



Finalist Reports

2021-2022

First Place

Team #47 Developing a Control Algorithm and Simulation for Thrust Vector Controlled Rockets

School **Los Alamos High**
Team Members **Daniel Kim,**
Andres Iturregui

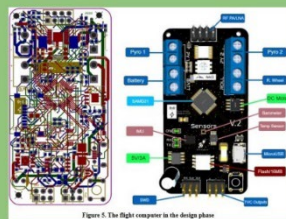
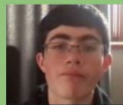


Figure 5: The flight computer in the design phase

WONDERFUL ACHIEVEMENT!

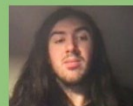


Second Place

Team #15 EL HAVEN (The Hearing, Exploring and Visualizing of Nubes)

School **New Mexico School for the Arts**
Team Members **Django Beaudoin, Elisea Jackson,**
Madelyn Kingston, Lexington Smith

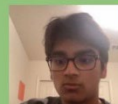
Teacher/Sponsor **Acacia McCombs**



Third Place

Team #34 Machine Learning - DNA Methylation & Alzheimer's Disease

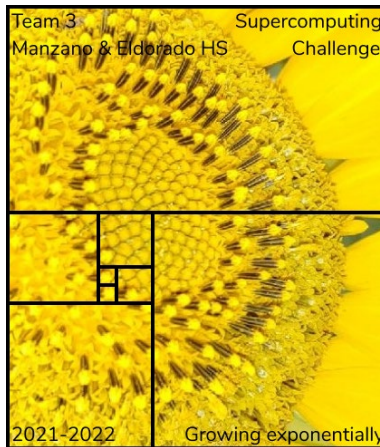
School **La Cueva High**
Members **Aditya Koushik, Abitpal Gyawali**
Teacher/Sponsor **Yolanda Lozano**



CONGRATULATIONS!



<https://supercomputingchallenge.org>



Cover: Top three teams in the 2021-22 Supercomputing Challenge which was held virtually via Zoom judging.



Notification: These final reports are presented in an abridged form, leaving out actual code, color, and appendices where appropriate. Complete copies of most of the final reports are available from the archives of Supercomputing Challenge web site:
<https://supercomputingchallenge.org> .

New Mexico Supercomputing Challenge

2021 – 2022 Finalist Reports

Table of Contents

About the New Mexico Supercomputing Challenge

For more information, please visit our website at

<https://supercomputingchallenge.org> 2

2021—2022 Challenge Awards 4

Sponsors 5

Scholarship winners 6

Participants 7

Judges 10

Finalist Reports 13

1. *Developing a Control Algorithm and Simulation for Thrust Vector Controlled Rockets*, Los Alamos High School Team 47
2. *Modeling Smoke Plume Dynamics from Imagery, El HEAVN': The Hearing, Exploring, And, Visualizing of Nubes*
New Mexico School for the Arts Team 15 (**Technical Writing Award winner**)
3. *DNA Methylation with Machine Learning: Prediction of Alzheimer's Disease and Novel Gene Therapeutics*,
La Cueva High School Team 34
4. *Robotic Air Quality Monitor: Snoopy*,
Capital High School Team 12
5. *A Ray of Hope*, Los Alamo High School Team 21
6. *Effects of Different Combinations of Halogen Additions to anti-B18H22 Molecule on Absorption and Emission Wavelength*,
La Cueva High School Team 24
7. *Designing Proteins with Quantum and Neuromorphic Computing*,
Los Alamos High School Team 27
8. *Classifying Mushroom Edibility*, La Cueva High School Team 30



Supercomputing Challenge Vision

The Vision of the Supercomputing Challenge is to be a nationally recognized program that promotes computational thinking in science and engineering so that the next generation of high school graduates is better prepared to compete in an information-based economy.

Supercomputing Challenge Mission

The Mission of the Supercomputing Challenge is to teach teams of middle and high school students how to use powerful computers to analyze, model and solve real world problems.

About the Supercomputing Challenge

The Supercomputing Challenge (the Challenge) is an exciting program that offers a truly unique experience to students in our state. The opportunity to work on the most powerful computers in the world is currently available to only a very few students in the entire United States, but in New Mexico, it is just one of the benefits of living in the "Land of Enchantment."

The Challenge is a program encompassing the school year in which teams of students complete science projects using high-performance supercomputers. Each team of up to five students and a sponsoring teacher defines and works on a single computational project of its own choosing. Throughout the program, help and support are given to the teams by their project advisors and the Challenge organizers and sponsors.

The Challenge is open to all interested students in grades 5 through 12 on a nonselective basis. The program has no grade point, class enrollment or computer experience prerequisites. Participants come from public, private, parochial and home-based schools in all areas of New Mexico. The important requirement for participating is a real desire to learn about science and computing.

Challenge teams tackle a range of interesting problems to solve. The most successful projects address a topic that holds great interest for the team. In recent years, ideas for projects have come from Astronomy, Geology, Physics, Ecology, Mathematics, Economics, Sociology, and Computer Science. It is very important that the problem a team chooses is what we call "real world" and not imaginary. A "real world" problem has measurable components. We use the term Computational Science to refer to science problems that we wish to solve and explain using computer models.

Those teams who make significant progress on their projects can enter them in the competition for awards of cash and scholarships. Team trophies are also awarded for: Teamwork, Best Written Report, Best Professional Presentation, Best Research, Creativity and Innovation,

Environmental Modeling, High Performance, Science is Fun and the Judges' Special Award, just to name a few.

The Challenge is offered at minimal cost to the participants or the school district. It is sponsored by a partnership of federal laboratories, universities, and businesses. They provide food and lodging for events such as the kickoff conference during which students and teachers are shown how to use supercomputers, learn programming languages, how to analyze data, write reports and much more.

These sponsors also supply time on the supercomputers and lend equipment to schools that need it. Employees of the sponsoring groups conduct training sessions at workshops and advise teams throughout the year. The Challenge usually culminates with an Expo and Awards Ceremony in the spring at the Los Alamos National Laboratory or in Albuquerque. In 2017 we started alternating the Expo and Awards Ceremony between Albuquerque and Los Alamos.

History

The New Mexico High School Supercomputing Challenge was conceived in 1990 by former Los Alamos Director Sig Hecker and Tom Thornhill, president of New Mexico Technet Inc., a nonprofit company that in 1985 set up a computer network to link the state's national laboratories, universities, state government and some private companies. Sen. Pete Domenici, and John Rollwagen, then chairman and chief executive officer of Cray Research Inc., added their support.

In 2001, the Adventures in Supercomputing program formerly housed at Sandia National Laboratories and then at the Albuquerque High Performance Computing Center at the University of New Mexico merged with the former New Mexico High School Supercomputing Challenge to become the New Mexico High School Adventures in Supercomputing Challenge.

In 2002, the words "High School" were dropped from the name as middle school teams had been invited to participate in 2000 and had done well.

In the summer of 2005, the name was simplified to the Supercomputing Challenge.

In 2007, the Challenge began collaborating with the middle school Project GUTS, (Growing Up Thinking Scientifically), an NSF grant housed at the Santa Fe Institute.

In 2013, the Challenge began collaborating with New Mexico Computer Science for All, an NSF funded program based at the Santa Fe Institute that offers a comprehensive teacher professional development program in Computer Science including a University of New Mexico Computer Science course for teachers.

2021—2022 Challenge Awards

32nd Annual New Mexico Supercomputing Challenge winners

First Place

Team #47 Developing a Control Algorithm and Simulation for Thrust Vector Controlled Rockets

School **Los Alamos High**
Team Members **Daniel Kim,**
Andres Iturregui

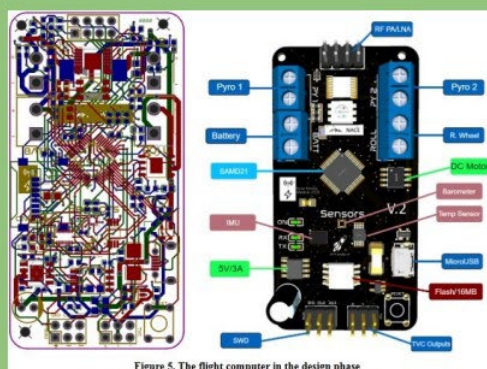
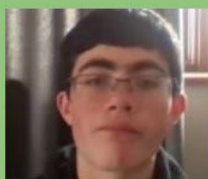


Figure 5. The flight computer in the design phase

WONDERFUL ACHIEVEMENT!



Second Place

Team #15 EL HAVEN (The Hearing, Exploring and Visualizing of Nubes)

School **New Mexico School for the Arts**
Team Members **Django Beaudoin, Elisea Jackson,**
Madelyn Kingston, Lexington Smith

Teacher/Sponsor **Acacia McCombs**



Third Place

Team #34 Machine Learning - DNA Methylation & Alzheimer's Disease

School **La Cueva High**

Members **Aditya Koushik, Abitpal Gyawali**

Teacher/Sponsor **Yolanda Lozano**



CONGRATULATIONS!



Supercomputing Challenge students across the state have completed their computational projects on topics such as Alzheimer's disease, datacasting, fire science, and environmental issues. This year the Supercomputing Challenge celebrated their 32nd annual Expo and Awards Ceremony on April 25 and 26, 2022 which featured student final presentations, judging and an award ceremony held on a virtual platform for the third year.

In this program, middle school and high school students are mentored by a community of volunteer scientists, computer programmers and professors. Several alumni also serve as volunteers. The Supercomputing Challenge partners include the [New Mexico Consortium](#), [Los Alamos National Laboratory](#), [Sandia National Laboratory](#), [Public Service Company of New Mexico](#), [Air Force Research Laboratory](#), [Westwind](#), and most New Mexico colleges and universities. A complete list of sponsors and supporters of the Challenge is on the website at <https://supercomputingchallenge.org/21-22/sponsors.php>.

"I am always impressed with the students in our state. We are so proud to be able to show their abilities," said David Kratzer, the executive director of the Supercomputing Challenge.

By participating in the Challenge, students learn to be prepared and successful in any career. They learn to be persistent by practicing their computer programming skills, completing research, and meeting deadlines to cross the finish line. The Challenge likes to refer to itself as an academic marathon, and these students should be recognized as critical thinkers, communicators, collaborators, and computer scientists.

Scholarships worth \$19,2000 were awarded to participating students planning to enroll in [New Mexico Tech](#), the [University of New Mexico](#) and [Central New Mexico Community College](#).

Many other awards were distributed ranging from \$50 per team member for finishing the academic marathon to team prizes of up to \$1000 for 1st and additional prizes for other categories such as teamwork, technical writing, programming ability, and community impact.

The Supercomputing Challenge is open to New Mexico middle and high-school students, including home-schooled students. Students work in teams and follow their own interests to choose a topic to computationally model. New students interested in becoming involved can make plans to start the next academic year, by contacting consult@supercomputingchallenge.org

Scholarships and Other Awards

A complete list of all winning student teams.

<https://supercomputingchallenge.org/21-22/expo/AllWinnersList.pdf>

Scholarship winners

Scholarships worth \$19,200 were awarded at the Supercomputing Challenge Awards Ceremony to 11 seniors: Nancy Avila, Candis Canaday, Gwenevere Caouette, Juan de la Riva, McLight Emma-Asonye, Madelyn Kingston, Mariah Rivera, Jack Sparrow, Joshua Tamarra, Angel Velasquez and Kyreen White.

All final reports are online.

<https://supercomputingchallenge.org/21-22/finalreports/submitted.php>

About the Supercomputing Challenge

“The goal of the yearlong event is to teach student teams how to use powerful computers to analyze, model and solve real-world problems,” said David Kratzer, Executive Director.

“Participating students improve their understanding of technology by developing skills in scientific inquiry, modeling, computing, communications, and teamwork.”

The New Mexico Supercomputing Challenge teaches written and oral communication, collaboration with peers and professionals, critical thinking including research and coding including computer modeling to middle and high school students throughout the state. Any New Mexico middle-school or high-school student, including home-schooled students are eligible to participate in the Supercomputing Challenge. Students follow their own interests to choose a topic to model.

Teams Finishing the Challenge and submitting final reports:

Team 3, Eldorado High School and Manzano High School,
How Climate Change Affects Power Outages and the Distribution of Power
Team Members: Nancy Avila, Candis Canaday, Gwenevere Caouette, Kyreen White
Sponsors: Sharee Lundsford, Karen Glennon, Patty Meyer

Team 4, Santa Fe Preparatory School, Capital High School, Santa Fe High School,
Optimizing the Geographic Location of Photovoltaic Panels in the Contiguous US
Team Members: Lucas Blakeslee, Ian Olson, Val Ornelas, Shrey Poshiya
Sponsors: Irina Cislaru, Jocelyn Comstock

Team 5, New Mexico Military Institute,
Economics Model with Wealth Distribution using Netlogo
Team Members: Ryan Lee, Jaejoon Lee
Sponsor: Mark Stone

Team 7, Capital High School, *Music On Mars:*
Using Fourier Analysis Spectra to Differentiate Martian Geology
Team Members: Juan De La Riva, Mariah Mejillas Rivera, Gilberto Morales, Jairo Salas Banda
Sponsors: Irina Cislaru, Barbara Teterycz

Team 8, Jackson Middle School, *Air pollution and How it Affects our Earth*
Team Members: Declan Neice, Antonio Espinosa
Sponsor: Sharee Lundsford

Team 9, Jackson Middle School, *Sea Ice Algae*
Team Members: Annabelle Brown, Sofia Cruz, Aiyana McCabe
Sponsor: Sharee Lundsford

Team 10, Taos High School, *Datacasting, an Opportunity in Educational Equity*
Team Members: Athanasios Bertin, Carlos Miller, Grace Goler, Lakai Tucker, Lola Shropshire
Sponsor: Tracy Galligan

Team 11, Santa Fe Preparatory School, *The Healing Process: A Simulation of Wounds*
Team Members: Isabel Voinescu, Caroline Moore
Sponsors: Jocelyn Comstock

Team 12, Capital High School, *Robotic Air Quality Monitor: Snoopy*
Team Members: Zachariah Burch, Joshua Mari Tamarra, Isel Aragon, Britny Marquez
Sponsor: Barbara Teterycz

Team 13, Taos High School, *Supercomputing Challenge Biofuel Proposal*
Team Members: Brian Hoang, Flynn Basehart, Chris Rivera, Izaiah Gonzales
Sponsor: Tracy Galligan

Team 15, New Mexico School for the Arts, *Modeling Smoke Plume Dynamics from Imagery, El HEAVN': The Hearing, Exploring, And, Visualizing of Nubes*

Team Members: Madelyn Kingston, Django Beaudoin, Lexington Smith, Elisea Jackson

Sponsors: Mohit Dubey, Acacia McCombs

Team 16, Cottonwood Classical and Del Norte High School, *A-Maze-Ing Algorithms*

Team Members: Ayyree Urrea, Kiara Onomoto, Violet Kelly

Sponsors: Karen Glennon, Patty Meyer

Team 17, Jackson Middle School, *Rapid Speed Of A Fire*

Team Members: Isaiah Biehle, Elijah Martinez

Sponsors: Sharee Lundsford, Karen Glennon

Team 18, Monte del Sol Charter School, *H2 Oh No: The Study of Water Vapor*

Team Members: Angel Martinez, Jack Sparrow

Sponsor: Rhonda Crespo

Team 19, Sandia Preparatory School, *A look into the collision of galaxies*

Team Members: Amaru Obrien, Uriah Armendariz

Sponsor: Nicholas Aase

Team 21, Los Alamos High School, *A Ray of Hope*

Team Member: Andrew Morgan

Sponsor: Nathaniel Morgan

Team 22, Justice Code/International/Harrison, *Machine Learning in Sports*

Team Members: Isaac Rankin, McLight Emma-Asonye

Sponsor: Caia Brown

Team 23, Justice Code/International/Harrison, *Maze Solving for Beginners*

Team Members: Mekhi Bradford, Kingesly Walker, Wellbeing Ogunsanwo, Hero Okhaifo

Sponsors: Caia Brown, Becky Campbell, Ryan Palmer

Team 24, La Cueva High School, *Effects of Different Combinations of Halogen Additions to anti-B18H22 Molecule on Absorption and Emission Wavelength*

Team Member: Melody Yeh

Sponsor: Creighton Edington

Team 26, Justice Code/International/Harrison,

The Correlation of Drug Ingestion and Brain Performance

Team Members: Ifunanya Egbo, Praise Ejimkonye, Sayra Medrano, Chinyere Offor,

Aileen Ukwuoma, Alexandrina Ukwuoma

Sponsors: Caia Brown, Becky Campbell, Ryan Palmer

Team 27, Los Alamos High School,
Designing Proteins with Quantum and Neuromorphic Computing
Team Member: Robert Strauss
Sponsors: Vikram Mulligan, Garrett Kenyon

Team 28, Truman Middle School, *Relationship Between Deforestation and Climate Change*
Team Members: Sebastina Puentes, Yahir Prieto, Camila Hernandez Delgado
Sponsor: Natali Barreto Baca

Team 29, Truman Middle School, *Amazon Deforestation*
Team Members: Briana Anaya, Sade Garcia, Yaritzel Castro
Sponsor: Natali Barreto Baca

Team 30, La Cueva High School, *Classifying Mushroom Edibility*
Team Member: Yana Outkin
Sponsor: Yolanda Lozano

Team 32, Justice Code/International/Harrison,
Can Visuals affect the way people experience sound?
Team Members: Chibuike Offor, Frances Emma-Asonye, Noel Emma-Asonye, Ifesinachi Egbo
Sponsors: Caia Brown, Becky Campbell, Ryan Palmer

Team 34, La Cueva High School, *DNA Methylation with Machine Learning: Prediction of Alzheimer's Disease and Novel Gene Therapeutics*
Team Members: Aditya Koushik, Abitpal Gyawali
Sponsor: Yolanda Lozano

Team 35, St. Thomas Aquinas School, *Forecasting Earthquakes*
Team Member: Ethan Ong
Sponsor: Eric Vigil

Team 37, St. Thomas Aquinas School, *Predator Prey Relationships*
Team Member: Trevor Wunsch
Sponsor: Eric Vigil

Team 45, Santa Fe High School, *Entropy in Chess*
Team Members: Aengus McGuinness, Armando Martinez-Brito, Roman Nappi, William Barral
Sponsor: Brian Smith

Team 47, Los Alamos High School,
Developing a Control Algorithm and Simulation for Thrust Vector Controlled Rockets
Team Members: Daniel Kim, Andres Iturregui

Team 48, Mountain Elementary School, *What is the best strategy for Master Mind?*
Team Member: Michael Petersen
Sponsor: Mrs. Hawkins

Team 51, Melrose High School, *There Is No Yellow*
Team Member: Lily Macfarlane
Sponsor: Alan Daugherty

Team 53, La Cueva High School, *Art With Tech*
Team Member: Victoria Outkin
Sponsor: Yolanda Lozano

Team 56, St. Thomas Aquinas School, *Understanding Sweetness*
Team Member: Julianna Matthews
Sponsor: Eric Vigil

Judges

Ed Angel, Professor Emeritus UNM
Joan Appel, New Mexico Community Data Collaborative, Supercomputing Challenge Alumni
Richard Barrett, Sandia National Laboratory/Retired
Jon Brown, Ohio State University
Hope Cahill, Santa Fe Public Schools
Angela Chrisman, Rio Rancho Public Schools
Ranjana Damle, University of New Mexico
Mike Davis, HPE/Cray Inc
Sharon Deland, Sandia National Laboratories/Retired
Mohit Dubey, Los Alamos National Laboratory, Supercomputing Challenge alumni
Juergen Eckert, Los Alamos National Laboratory
Creighton Edington, Rural Education Advancement Program
Drew Einhorn
Daniel Fuka, Virginia Tech University
Susan Gibbs, Project GUTS
Kaley Goatcher, TransCore
Stephen Guerin, Simtable
Harry Henderson
Christopher Hoppe, Hoppe's Art Studio, Supercomputing Challenge alumni
Clint Hubbard, Former CIO, City of Albuquerque

David Janecky, Los Alamos National Laboratory/Retired
Elizabeth Jimenez, University of New Mexico
Philip Jones, Los Alamos National Laboratory
Elizabeth Kallman, Databutterfly
Cole Kendrick, Oakridge National Laboratory, Supercomputing Challenge alumni
Amy Knowles, New Mexico Institute of Mining and Technology
Nicholas Kutac, Puget Sound Naval Shipyard, Supercomputing Challenge alumni
Peter Lamborn, Los Alamos National Laboratory
Maximo Lazo, Central New Mexico Community College
Joan Lucas, University of New Mexico-Los Alamos
Kasra Manavi, Simtable, Supercomputing Challenge alumni
Sam McGuinn, Cisco Systems, Supercomputing Challenge alumni
Monique Morin, Los Alamos National Laboratory
James Overfelt, Sandia National Laboratories
Veena Parboteeah, New Mexico Highlands University
Paige Prescott, Computer Science Alliance
David Price, New Mexico State University
Maureen Psaila-Dombrowski, Los Alamos National Laboratory
Dana Roberson, Central New Mexico College
Tom Robey, Gaia Environmental Sciences
Rod Sanchez, New Mexico Highlands University
Dorian Sims, Westwind
Darko Stefanovic, University of New Mexico
David Torres, Northern New Mexico College
Michael Trahan, Sandia National Laboratories
Geoff Valdez, PNM
Rachel Washington, Georgia Institute of Technology, Supercomputing Challenge alumni
James Wernicke, Los Alamos National Laboratory, Supercomputing Challenge alumni
Phil Wolfram, Los Alamos National Laboratory



Do you want to become a supporter of the Supercomputing Challenge?
Please email us at consult@supercomputingchallenge.org for details.

Developing a Control Algorithm and Simulation for Thrust Vector Controlled Rockets

New Mexico Supercomputing Challenge
Final Report
April 6, 2022

Team 47
Los Alamos High School

Team Members:

- Daniel Kim
- Andres Iturregui

Project Mentor:

- Chris Karr

Executive Summary

In the past five years, SpaceX has revolutionized the aerospace industry by introducing a new rocket that can propulsively land. At the heart of these rockets, and virtually all orbital and suborbital spacecraft is thrust vector control (TVC). TVC is where the engine of the rocket is moved using a gimbal to vector its direction and create torque forces to keep a rocket on its desired trajectory. The purpose of this project was to create a model rocket with a flight computer, develop a control algorithm for the TVC system, and program a simulation to model the flight of the thrust vector controlled rocket.

The model rocket was designed using CAD software and made using a 3D printer, and is capable of vectoring the rocket motor 5 degrees in any direction. The control system we decided to use is a PID, or proportional-integral-derivative controller. We chose this due to its simplicity, effectiveness, and ability to tune to any rocket. The flight computer we designed is able to run this PID algorithm, along with logging flight data and controlling the engine mount.

However, we needed a method to tune this PID controller. In order to do this, we developed a simulation in MATLAB Simulink so that we could find the optimal PID values for our real flight. The simulation allowed us to plug in variables such as the mass of the rocket and the amount of wind and receive a flight trajectory of the rocket.

Once both the real-life model and the simulation were finished, we did an actual test flight of the rocket. Unfortunately, the rocket quickly flipped out of control, so the flight was unsuccessful. This unsuccessful flight was caused due to a misalignment in the TVC mount that caused the rocket to immediately go off of its straight trajectory. We hope to fly again soon with an improved real-world model and simulation to back it up.

Introduction

Currently, there are over 10,000 companies in space technology development with a combined value of over \$4 trillion [5] dedicated to advancing technologies in navigation, tourism, national security, communication, and outer space research. This expansive growth prompts the need for new suborbital and orbital-class vehicles to transport these technologies into outer space. Although these vehicles, the majority being rockets, present multiple challenges, one of the biggest obstacles in developing a navigation and control system to guide them.

A majority of these spacecraft use a technique called thrust vector control (Figure 1). By angling, or vectoring, the direction of the thrusting component, the rocket has control over position and orientation, even in a non-atmospheric environment. Engineers have designed and implemented gimbals into the rocket engines to allow for this technique across a variety of different rockets, but the software to control these gimbals and the rockets become very complicated because of the high speeds, large masses, and precision required from spacecraft.

Furthermore, methods to tune these control algorithms also become very complex due to the high cost and a large number of variables involved in a rocket flight. Although a majority of rockets use the same technique as TVC for active control, the control system design and tuning present many challenges.

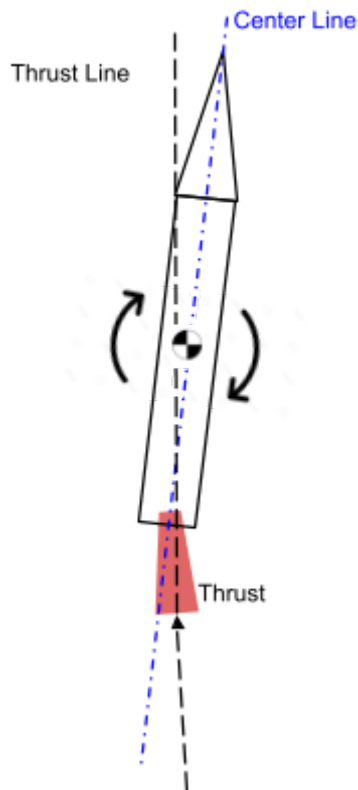


Figure 1. By vectoring the thrust, the rocket's orientation is rotated about the vehicle's center of mass. This method can be used to control the orientation and position of a model rocket.

The purpose of this research and engineering project was to design a versatile control system that is compatible throughout different dynamics of different rockets and to create a simulation to tune this control system and predict flight dynamics of different rockets. Implementation of this robust and dynamic control system and simulation offers a simpler way for the growing space industry to solve one of the most challenging problems of guidance, navigation, and control. In this project, a control algorithm, orientation scheme, and simulation were developed and tested in the form of a model rocket with a TVC system and control system implemented.

Control Theory

The control system that we decided to use and tailor to rocket guidance, navigation, and control, or GNC, was the very widely used Proportional-Integral-Derivative (PID) controller. This is a feedback controller that regulated steam engines during the industrial revolution, improved the yield from windmills and industrial processes worldwide, and lies at the heart of autopilots used in commercial airplanes. This control algorithm allows for a simple integration and tuning process for all rockets of various sizes and purposes by adjusting three values. The PID controller is defined as

$$u(t) = K_p e(t) + K_i \int e(t) \Delta t + K_d \frac{de}{dt} \quad (1)$$

Where $u(t)$ is the output, K_p , K_i , and K_d are the three gains, $e(t)$ is the error value and the input to the controller, and Δt is the change in time. There are three terms to this equation: the proportional, integral, and derivative of the error value $e(t)$, which are scaled using the adjustable gains K_p , K_i , and K_d . The proportional, integral, and derivative components allow for control over oscillations, overshoot, and small errors. This simplicity allows for use on lower-powered flight controllers, but the three components work together to produce a very robust, efficient, and versatile control system.

On a rocket, this method can be easily integrated. We can have three PID controllers, one for each axis to control orientation. The error, which is inputted into the controller, is equal to the setpoint subtracted from the current orientation. The setpoint is adjustable based on the desired angle. After performing the proportional, integral, and derivative components, we can then command these angles to the gimbal, or thrust vector control mount (Figure 2).

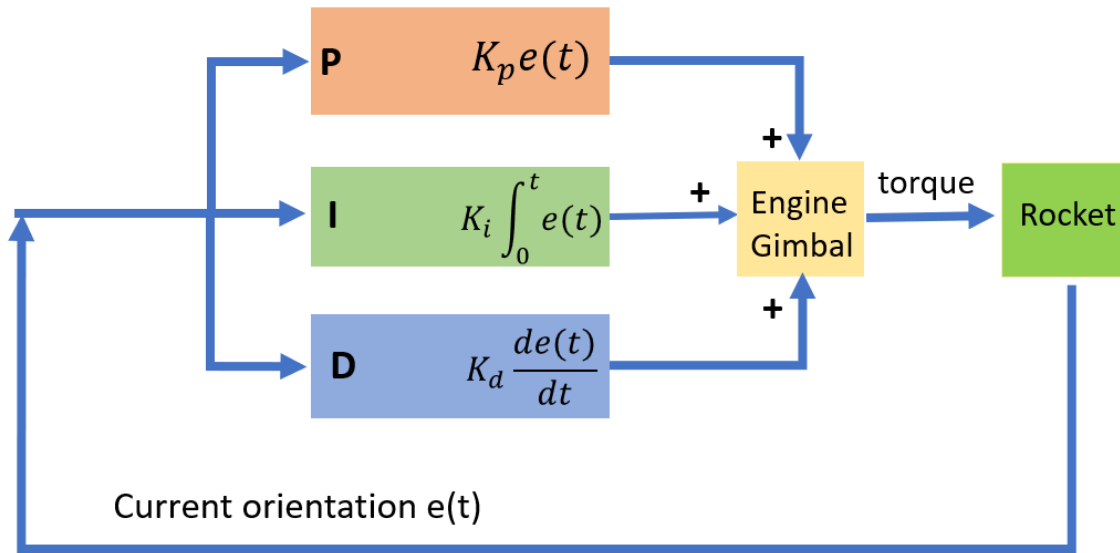


Figure 2. PID Controller for a rocket.

Methods

Real-World Model

To validate that the control system and simulation worked properly, we designed a model rocket and a TVC gimbal in computer-aided-design, CAD, and 3D printed components to replicate a real-life rocket.

The rocket itself was printed in PLA and supported by four carbon fiber rods. Parachutes that were stored in the upper body were deployed using a small pyrotechnic charge ignited by a load driver aboard the flight computer.

The thrust vector control mount was a two-axis (Figure 4) gimbal that was designed to vector the rocket motor using two servo motors that were controlled by the flight computer. The gear ratio of the motors and the mount was 2:1, which allowed for more accuracy and less error within the mount, and the mount had a range of ± 5 degrees in both axes.

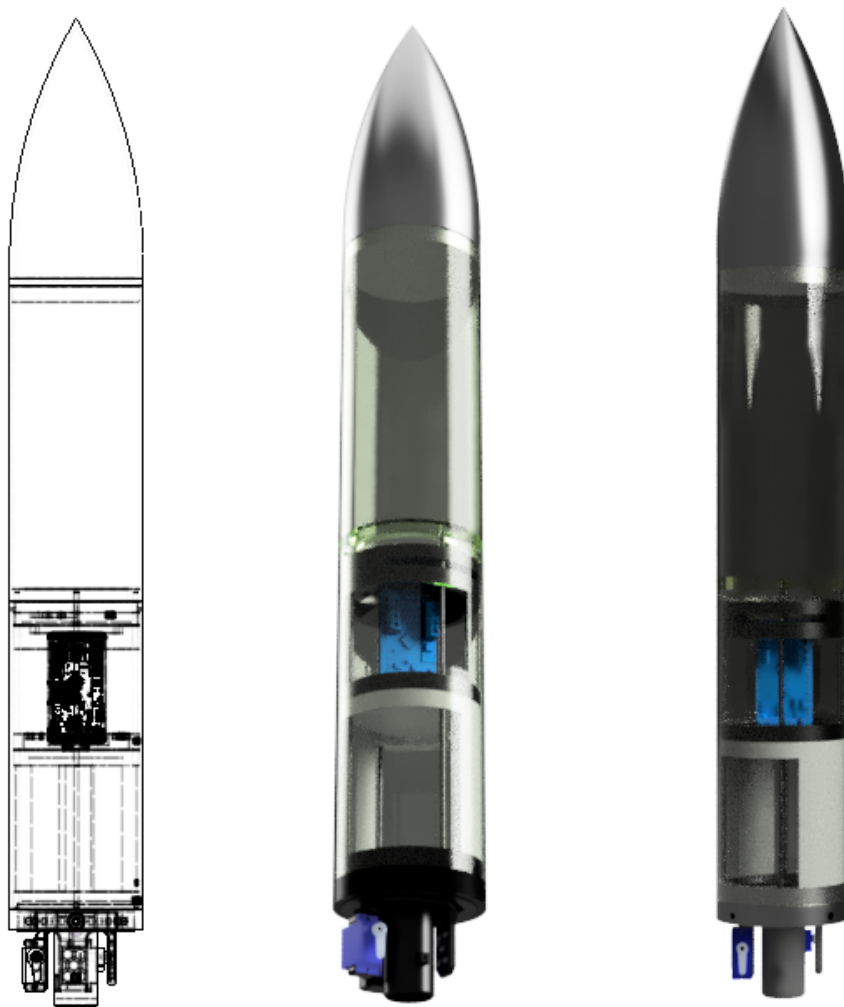


Figure 3. The model rocket design in CAD

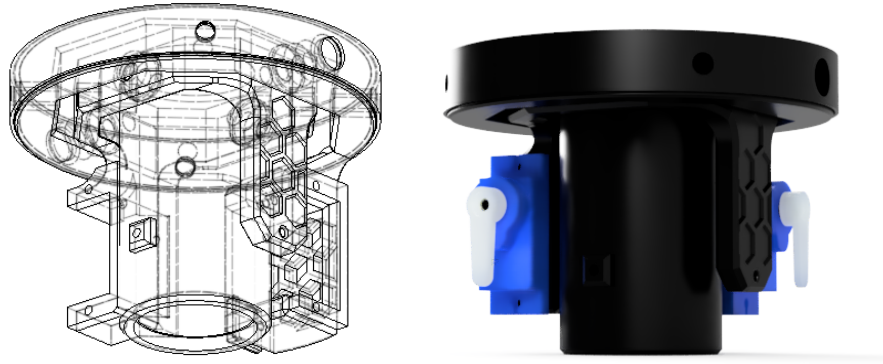


Figure 4. The thrust vector control gimbal

Flight Computer

To control the servo motors, calculate orientation, deploy recovery parachutes, log data, and execute the control algorithm, we designed and fabricated a custom flight computer (Figure 5). By collecting and analyzing sensor data, the flight computer was designed to control all aspects of our model.

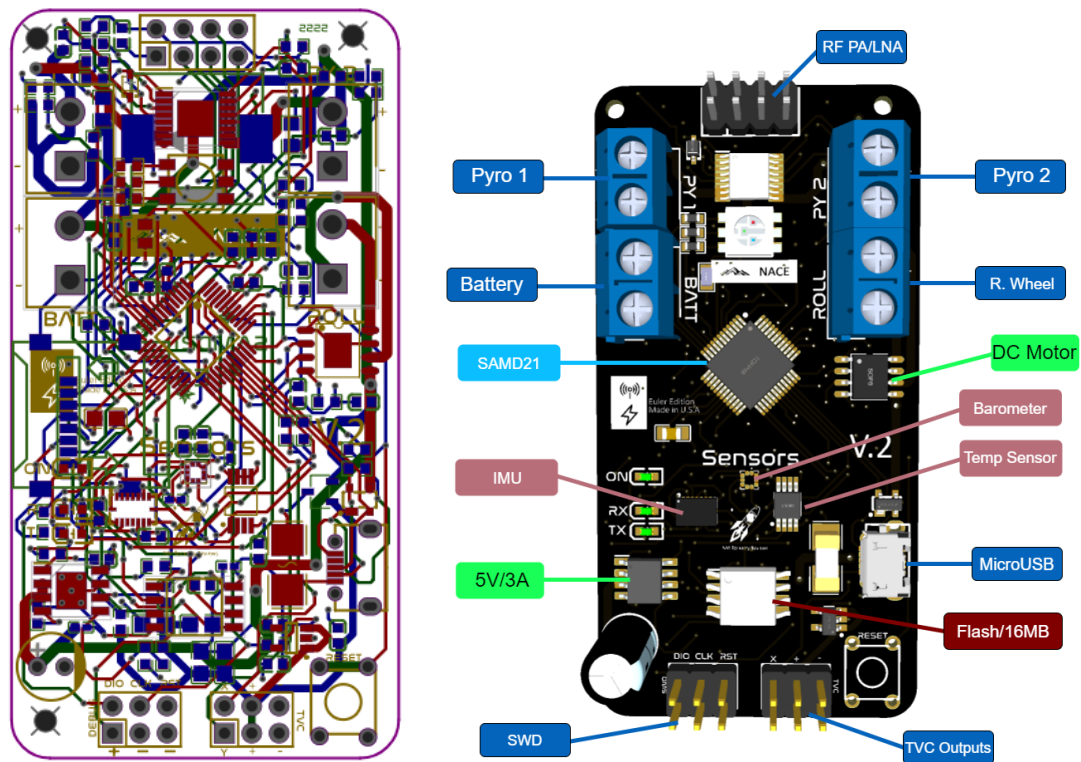


Figure 5. The flight computer in the design phase

Flight Software Design

The flight software was written in C++ and controlled all the components of the flight computer and control system. The flight computer ran a finite-state automation function to control parachute deployment and data logging tasks. To calculate the orientation of the craft using the angular rate measurements from the inertial-measurement-unit, or IMU, the flight computer used quaternions and a linear/1D Kalman Filter to estimate the orientation of the rocket.

Linear Kalman Filter

In our model, a linear Kalman filter is used to filter noise from sensor data [7]. A Kalman filter calculates an estimate of unknown variables within measurements caused by external factors. Since velocity and position are calculated by integrating and double-integrating acceleration respectively, even a small amount of noise will cause significant drift in the measurements.

We define the Kalman Gain, which is

$$G_n = \frac{p_n}{p_n + r_n} \quad (2)$$

Where:

- G_n is the Kalman Gain where $0 \leq G_n \leq 1$
- p_n is the estimated uncertainty
- r_n is the measurement uncertainty

We then calculate the current estimate X_t by

$$X_t = X_{t-1} + G_n \cdot (Z_n - X_{t-1})$$

(3)

Where:

- X_t is the current estimate at time step t
- X_{t-1} is the previous estimate
- Z_n is the measurement or the sensor data in our case

And the estimation uncertainty can be calculated using the Covariance Update Equation

$$p_n = (1 - G_n) \cdot p_n + |X_{t-1} - X_t| \cdot q$$

(4)

Where q is the process variance, which increases the accuracy based on how much the measurement moves. We can then set the previous estimate, X_{t-1} , to the current estimate

$$X_{t-1} = X_t \quad (5)$$

Quaternions

The simple integration of the angular rates provided by the IMU to Euler angles results in gimbal lock, where two axes align and differentiation between the two axes is not possible. Quaternions and quaternion algebra, which were invented by Sir William Rowan Hamilton, were used in this project to solve this issue and to allow for 3D rotations. Quaternions represent orientation in four dimensions in 3D space. Since the IMU outputs the angular rate of x , y , and z in rad/s so we can refer to those as

$$\omega_t = [0, x, y, z] \quad (6)$$

We can also refer to the base world reference frame, and since the rocket points upwards at launch, we can introduce Q_{init}

$$Q_{init} = [1, 0, 0, 0]$$

(7)

Next, we can calculate the quaternion derivative that describes the rate of change of orientation relative to the earth.

$$\frac{dQ_t}{dt} = \frac{1}{2} \cdot Q_{init} \otimes \omega_t$$

(8)

Where:

- $\frac{dQ_t}{dt}$ is the quaternion derivative at time step t
- Q_t is the orientation at time step t
- \otimes is the Hamilton Product operator (Appendix A)

We can integrate the quaternion derivative to determine the orientation at t with

$$Q = Q_{init} + \Delta t \cdot \frac{dQ_t}{dt} \quad (9)$$

And Δt is the time step. We now normalize the quaternion by first calculating the norm, and then dividing the orientation by the norm

$$Q = \frac{q}{\sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}} \quad (10)$$

Where Q_1, Q_2, Q_3, Q_4 are the individual elements of quaternion Q . This quaternion can be converted back to Euler angles without the risk of gimbal lock by equations 11-13.

$$\alpha = \tan^{-1} \left(\frac{2 \cdot Q_3 Q_1 - 2 \cdot Q_2 Q_4}{1 - 2 \cdot Q_3^2 - 2 \cdot Q_4^2} \right) \quad (11)$$

$$\beta = \sin^{-1} (2(Q_2 Q_3 + 2 \cdot Q_4 Q_1)) \quad (12)$$

$$\theta = \tan^{-1} \left(\frac{2 \cdot Q_2 Q_1 - 2 \cdot Q_3 Q_4}{1 - 2 \cdot Q_2^2 - 2 \cdot Q_4^2} \right) \quad (13)$$

Where α is the roll of the craft, β is the yaw of the craft, and θ is the pitch of the craft expressed in radians (Figure 6).

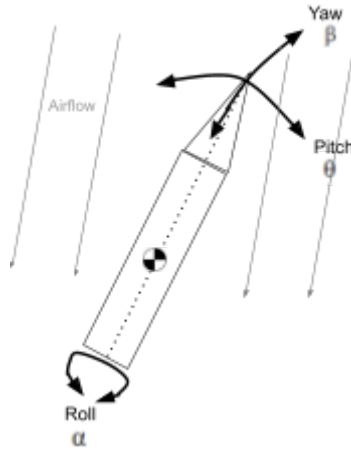


Figure 6. Rocket Rotations

To reduce unnecessary complexity, the real-world model did not include roll control, which is not controllable by the TVC mount. To account for any roll changes during the flight, θ , and β must be decoupled from α using

$$\theta = \cos(\alpha) \cdot \theta - \sin(\alpha) \cdot \beta \quad (14)$$

$$\beta = \cos(\alpha) \cdot \beta + \sin(\alpha) \cdot \theta \quad (15)$$

θ and β can then be inputted into the two PID controllers for both the pitch and yaw axis.

Quaternions are also extremely useful in rotating different orientations. The accelerometer unit on the IMU provides linear acceleration in the vehicle reference frame. This reference frame is not compatible with velocity and position measurements, as well as accelerometer measurements used to detect launch, burnout, and apogee for the finite state

automation function. By finding the Hamilton product of the orientation quaternions and the acceleration readings, we can convert the measurements to a world reference frame. We can refer to the accelerometer readings as

$$a_t = [0, a_x, a_y, a_z] \quad (16)$$

Then, we can find the world reference frame orientation by finding the first Hamilton product of the orientation quaternion and a_t .

$$A_{first} = Q \otimes a_t \quad (17)$$

Next, we find the Hamilton product of A_{first} and $-Q$ to flip the reference frame to the correct orientation

$$A = A_{first} \bullet -Q \quad (18)$$

Where A_2, A_3 , and A_4 the 3 of the 4 components of quaternion A , represent the x, y, and z linear accelerations in the world frame respectively.

In conclusion, the flight computer:

1. Reads raw data from sensors.
2. Filters the data using the linear Kalman filter.
3. Converts the raw gyro and time parameters into Euler angles using quaternions.
4. Inputs the Euler angles into the PID controller and commands the output of the PID controller to the servo motors of the TVC mount.
5. Rotates the filtered acceleration readings to the world frame using quaternion operators and uses these values for the state automation function.

Simulation

To tune the PID controller and verify that our real-life system would work properly, we have created a simulation of the rocket in MATLAB Simulink. We have worked to account for many real-life variables, including:

- Noise from sensor data
- Latency from flight computer and servo motor
- Wind disturbances

The most fundamental part of the simulation is the 3DOF block, as it utilizes all of the equations in three degrees of freedom motion to simulate an object in 2-dimensional space. This allows us to input x and z direction vector forces and torque, then receive x and z position and velocity, along with an angular position and velocity. real-life this to work properly, we needed to know the mass moment of inertia (MMOI) and mass of our actual rocket. To find the MMOI, we used the bifilar pendulum method, which involves hanging the rocket by two parallel strings equidistant from the center of mass and measuring the period of the swing rotating around the center of mass. The equation to find the MMOI is:

$$MMOI = \frac{m g p^2 r^2}{4 \pi^2 L} \quad (19)$$

Where $MMOI$ is the mass moment of inertia, m is mass, g is the gravitational constant, p is the period for one complete swing of the rocket, r is the distance from the center of mass and where the string is attached, and L is the length of the strings.

Engine Force and Gravity

The two most critical forces being applied to a rocket are gravity and the engine force. The 3DOF block already has a feature for applying gravity, which made the implementation extremely simple. As for the engine force, the data for the D12-0 engine we are using is already provided by Estes (Appendix C), so we were able to implement that data into the simulation. However, since we are using a TVC mount, we didn't just want the engine force to be applied in the z-direction, as it needed to be vectored based on the TVC mount's position. To solve this, we split the force into x and z vectors using the following two equations:

$$F_{motor(z)} = F_{motor} \times \cos(\sigma + e) \quad (20)$$

$$F_{motor(x)} = F_{motor} \times \sin(\sigma + e) \quad (21)$$

In these equations, $F_{motor(x)}$ and $F_{motor(z)}$ are the x and z direction vector forces respectively, F_{motor} is the total engine force, and σ is the angle the TVC mount. The motor mount misalignment, represented by e is to 1 degree to account for any real-life misalignments that may occur. We also use this equation

$$T_{motor} = F_{motor(x)} \times a \quad (22)$$

To calculate the torque produced by the TVC mount, where T_{motor} is the torque produced by the engine and a is the distance from the center of mass to the end of the rocket motor. The way the TVC mount position is calculated will be explained later in the simulation overview.

Noise Addition and PID Calculations

Since the accelerometer used on the real rocket does not yield perfect results, we wanted to simulate some of that noise. A random number generator generating values of +/- 0.1 degrees was added to the actual angular position of the rocket in degrees.

To calculate the PID values, we use the PID equation. To calculate the output, this equation must be the same in both the real-world model and the simulation because we want the K_p , K_i , and K_d gains found in our simulation to be used in the real-life model with optimal results.

Real-Life Limitations

Next, we need to account for some of the real-life limitations of the rocket, such as flight computer delay, servo speed limitations, and servo position limitations. The TVC control mount on the real rocket can only move the engine ± 5 degrees, so this is accounted for by saturating the output $u(t)$ to have a maximum of 5 degrees and a minimum of -5 degrees. In addition, the TVC servos have a limited angular rate. To measure the speed of the servo, we recorded a slow-motion video of the servo moving as fast as it can and calculated the maximum rotational speed to be 258 degrees per second. We used this value as the maximum rate at which the PID output could change. Finally, flight computer latency needs to be accounted for. Although the flight computer latency is estimated to be only 7.5 ms, we decided to increase it to 50 ms to have the simulation account for more imperfection. After all three of these limitations are applied to $u(t)$, the final resulting angle will be the position of the TVC mount, σ .

Wind Forces

To account for wind, we applied another torque in the form of a sine wave. Since we don't have any data on how much torque is applied for a given velocity of wind, we experimented with the amount of wind torque and set it to be as high as the rocket could handle. After some experimentation, we found that the most wind the rocket could handle was a sine wave going between 0 Nm and 0.1 Nm.

Aerodynamic Forces

The two most important aerodynamic forces that act on the rocket are the drag and lift force [4]. The drag force acts opposite to the direction the rocket is pointing towards, and the lift force acts perpendicular to the rocket but is applied from the center of pressure [4]. We were able to implement these forces with the help of a software called OpenRocket. OpenRocket is a software that allows for the simulation of model rocket launches and lets the user customize the rocket. We used this software and created our real-life rocket, ensuring that parameters such as the mass, center of mass, MMOI, and shape of the rocket were all the same. Then, we exported a large amount of data from OpenRocket for various amounts of wind, including the angle of attack, total velocity, drag force, lift force coefficient, and center of pressure position. The angle of attack is the angle at which the rocket is relative to the air flowing past it as opposed to the ground [10]. All of this data was then implemented into the simulation. However, we only received the lift force coefficient, but not the lift force. The equation to find the lift force is

$$F_l = \frac{1}{2} \times c_l \times d_{air} \times a_{ref}$$

(23)

Where F_l is the lift force, c_l is the lift force coefficient, d_{air} is the density of air, and a_{ref} is the reference area of the rocket [6]. The density of air is 1.225 kg/m³ at sea level, and the reference area is 5.6745•10⁻³ m² according to OpenRocket. To find the torque produced by the lift force, we multiply the lift force by the distance between the center of pressure and center of gravity, which varies based on the angle of attack of the rocket.

Tuning for Optimal Results

After each component of the simulation was completed, the next step was to tune the PID values K_p , K_i , and K_d . To begin tuning, we set K_i to 0 since its effect was not determined to be significant, and began experimenting with K_p and K_d values. We knew that low K_p gains would result in insufficient correction, and high K_p values would result in significant overshooting. K_d gains that were too low would result in overshooting even with low K_p values, and K_d values too high would result in extremely rapid oscillations. When we eventually found a K_p and K_d value combination that resulted in a successful flight, we began fine-tuning these values by moving each of them up and down by small increments individually and seeing if the flight had less or more error. Once we found the K_p and K_d values with the smallest amount of error, we added back the K_i term and increased it gradually until it hindered the results compared to lower values.

Results and Discussion

Simulation

Once tuning was complete, we got PID values of $K_p=0.5$, $K_i=1.4$, and $K_d=0.08$ inside of the simulation. As seen in Figure 7, the rocket coasts up to an altitude of about 18 meters before falling back down to the ground. Figure 8 shows the angle of the rocket as a function of time, which is much more important than the altitude. From 0-1.6 seconds, the TVC mount can handle the wind and prevents the rocket from exceeding an angle of 10 degrees, after this, the rocket motor burns out and is uncontrolled.

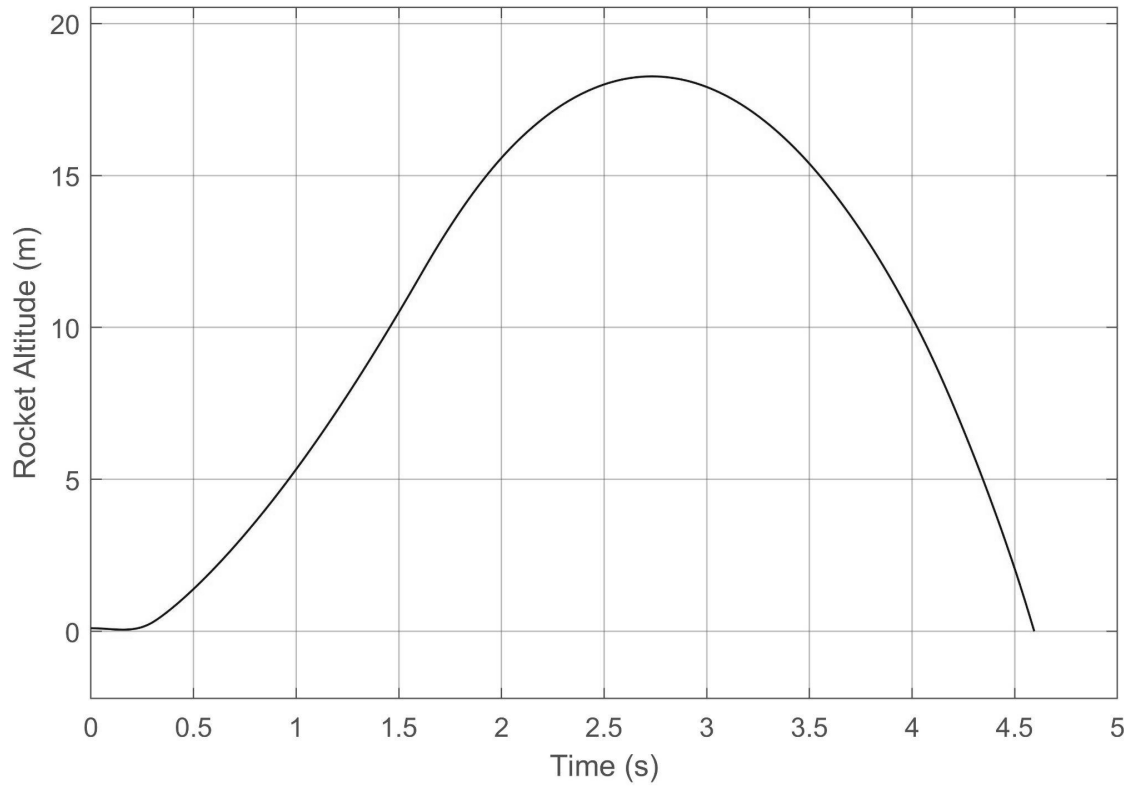


Figure 7. Shows the altitude of the simulated rocket as a function of time.

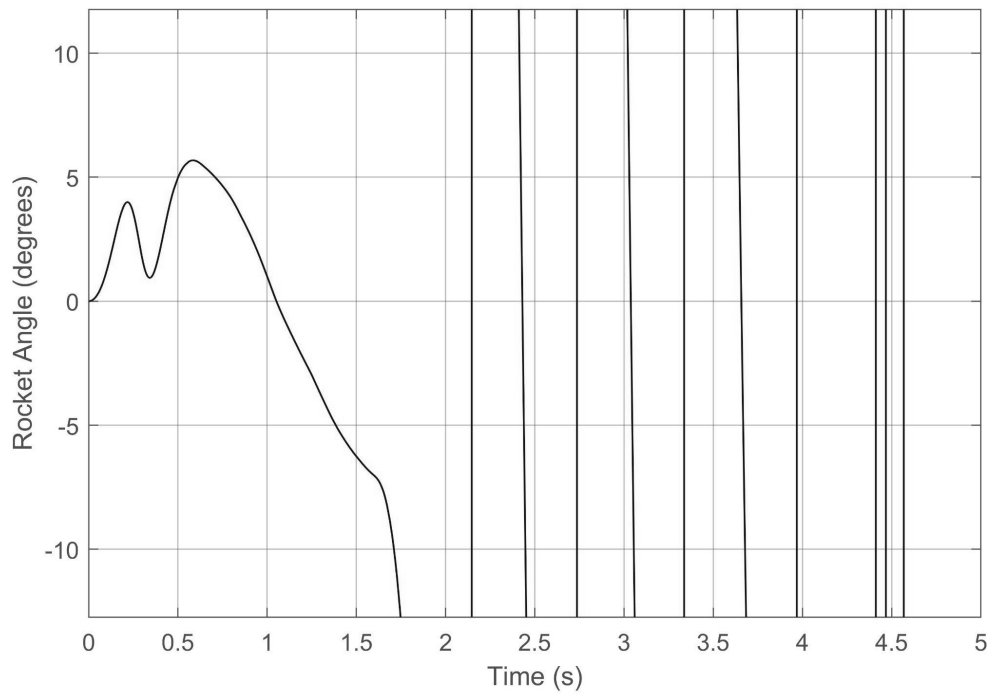


Figure 8. Shows the rocket's angle relative vertical in degrees as a function of time.

Unfortunately, the simulation was less advanced at the time that we flew the real rocket, so the PID values were instead $K_p=0.3$, $K_i=0.3$, and $K_d=0.05$. These PID gains were then inputted into the real-world model's flight software and flown on the Estes D12-0 rocket motor (Appendix C). The rocket had a mass of 0.605kg and a mass-moment-of-inertia of 0.01428 kgm². For simplicity purposes, the software had a constant setpoint of zero on both axes, meaning that the rocket would try to stay fully upright.

Real-World Model

The real-world model as discussed on page 5 was flown once using the parameters calculated from the simulation. The rocket was flown on an Estes D12-0 (Appendix C).

Figure 9 shows the estimated orientation of the rocket after liftoff was detected. The rocket almost immediately pitched over after liftoff. At around 0.372 seconds, data logging stopped and the PID loop was terminated because the flight software had an integrated abort system that triggered when the pitch or yaw surpassed +/- 30 degrees. This abort sequence saved all the data into the SD card, detached electrical connections to the servo motors, terminated all functions and deployed the parachutes. This sequence ensured that all relevant data would be saved and that damage to the rocket and avionics would be minimized when the rocket's orientation surpassed a point of no return.

Estimated Orientation during Flight

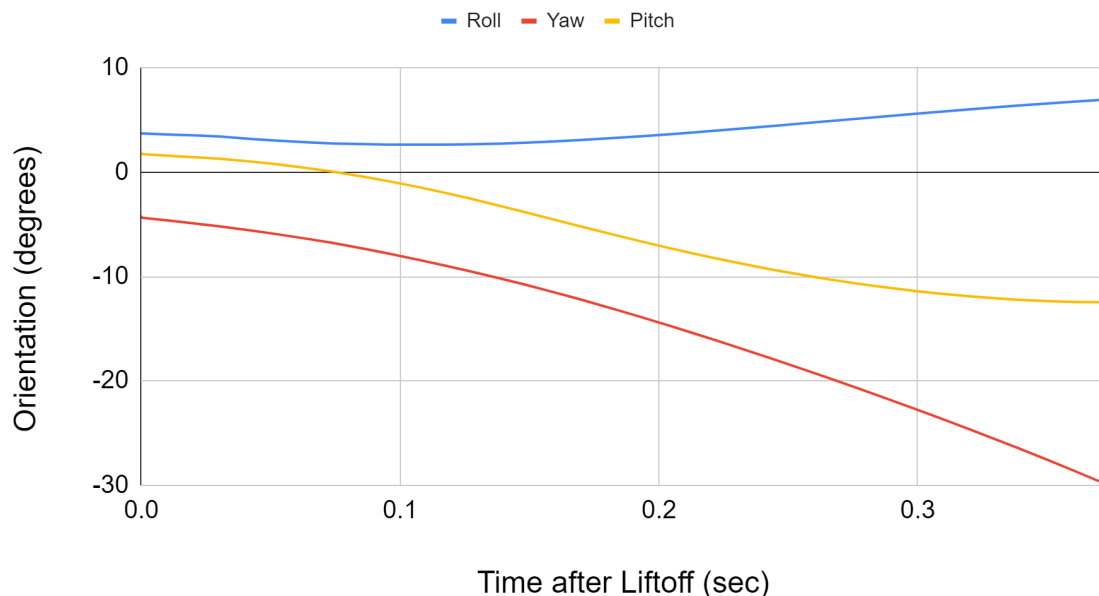
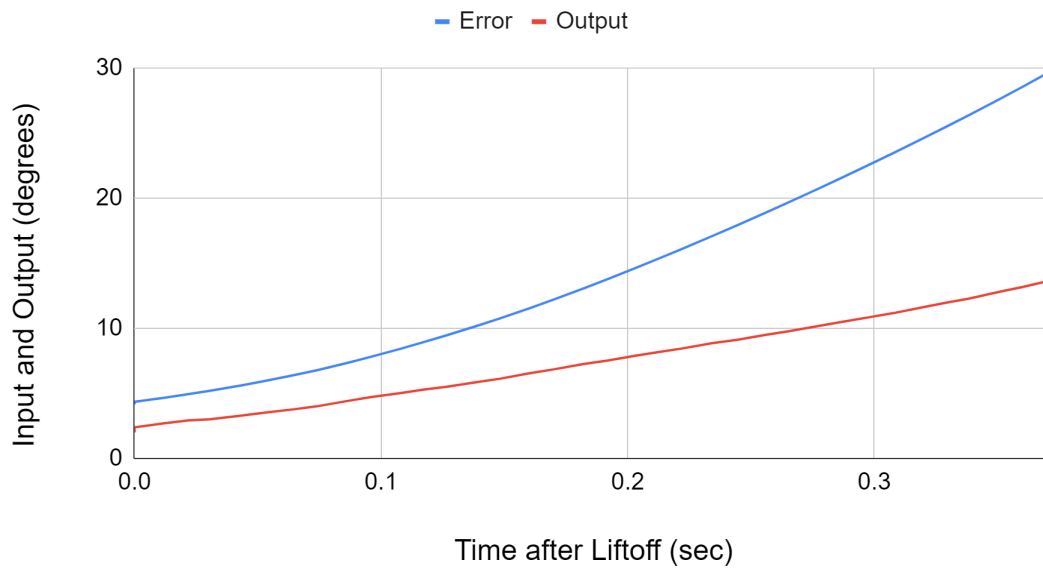


Figure 9. Orientation of the rocket in degrees as a function of time after liftoff was detected by the finite state automation function. Data was cut off at about 0.372 seconds by the abort sequence.

The large undesired orientation change was caused by several issues within the physical model. When assembling the thrust vector control gimbal, there were misalignments within the

two axes. So at liftoff, the rocket immediately pitched over, as seen in Figure 9. As the PID controller tried to correct this error, the misalignment caused larger errors to occur. As can be seen in Appendix B and Figure 10, the PID controller tried to output a value much greater than 5 degrees, and subsequently kept outputting values greater than 5 degrees. The TVC mount had an actuator limit of ± 5 degrees, which means it was not accurately reflecting the command output of the PID loop.

Yaw: Error vs Output of the PID Controller



Pitch: Error vs Output of the PID Controller

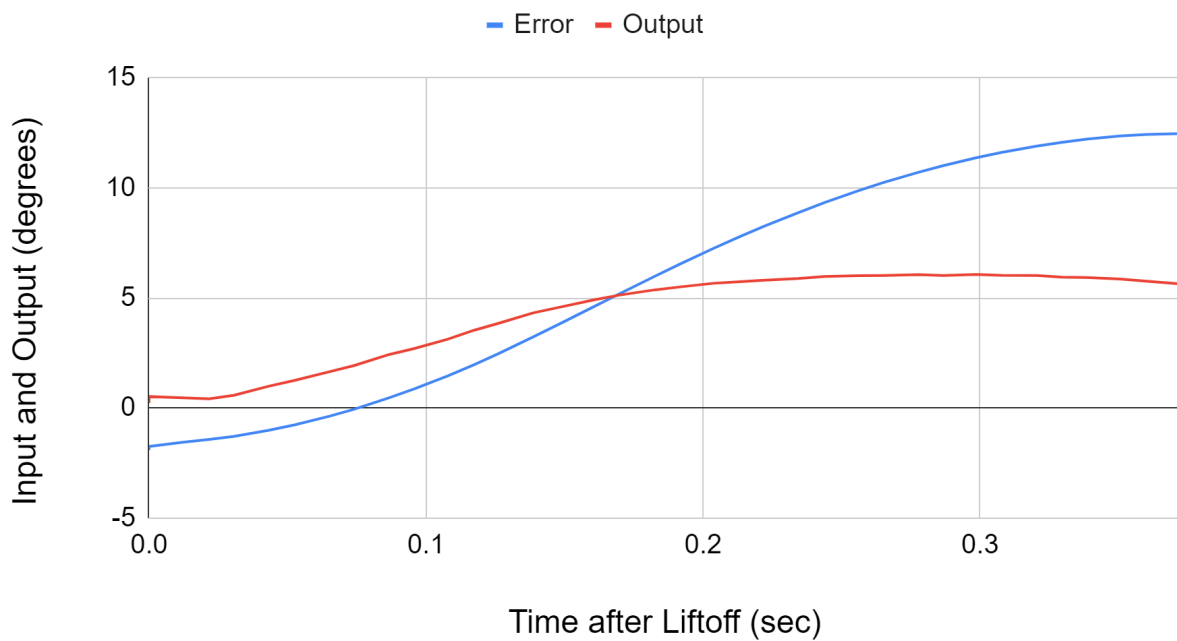


Figure 10. Error inputted to the PID controller versus output as a function of time detected after liftoff.

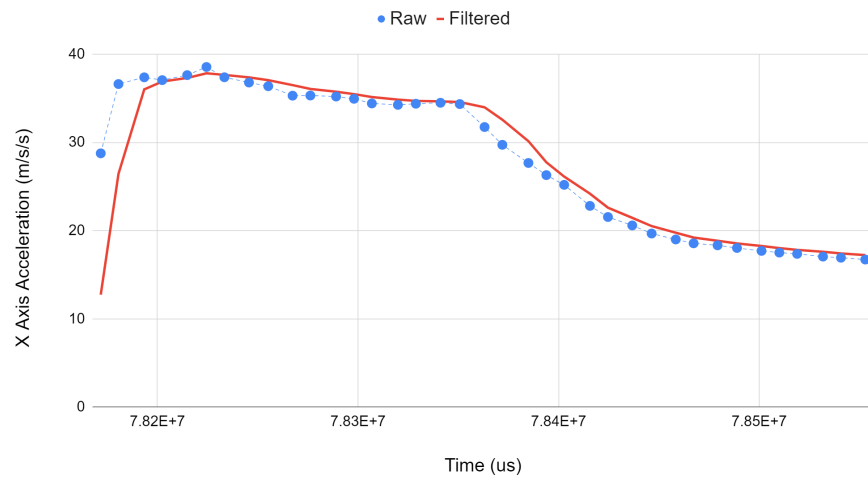


Figure 11. Different phases of the rocket flight are depicted in one photo. By the second and third freeze frames, the abort sequence had already fired and the rocket was uncontrolled.

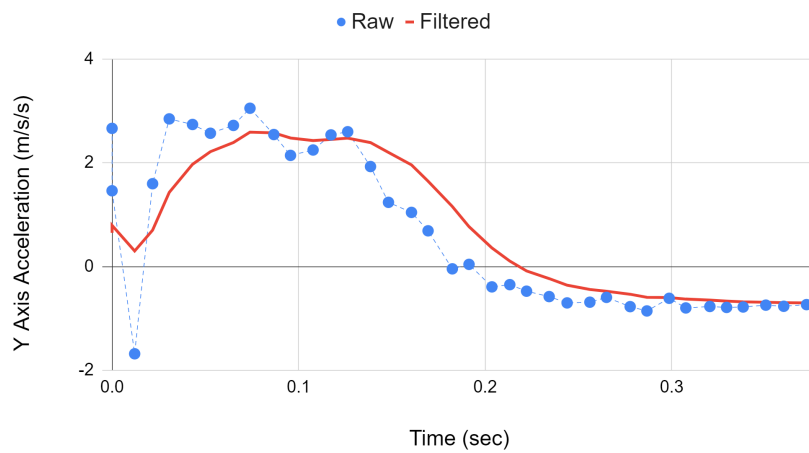
After the abort was called, the rocket flew completely out of control and collided with the ground. Although the rocket's outer body frame was shattered and the TVC mount was broken, the flight avionics, data, and servo motors were still intact. Figure 12 depicts other parameters measured during the flight, where X, Y, and Z are the roll, yaw, and pitch axes respectively. All measurements were plotted as a function of time detected after liftoff.

Figure 12 also depicts the raw versus filtered linear acceleration readings in the vehicle reference frame. The raw readings are passed into separate Kalman filters, which smooth the data.

X Axis: Raw vs Filtered Accelerometer Readings



Y Axis: Raw vs Filtered Accelerometer Readings



Z Axis: Raw vs Filtered Accelerometer Readings

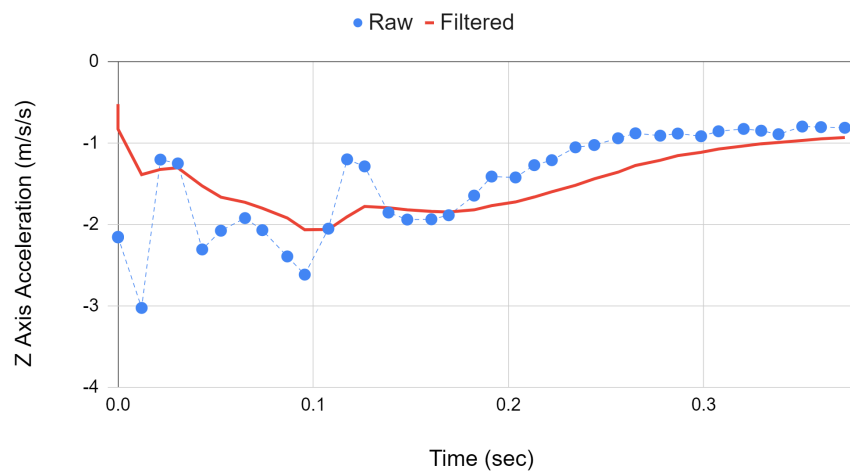


Figure 12. Raw and filtered accelerometer readings. Once again, the process variance in the Z-axis needs to increase since the filtered values are not accurately accounted for the large changes in acceleration.

X,Y,Z Gyroscopic Values

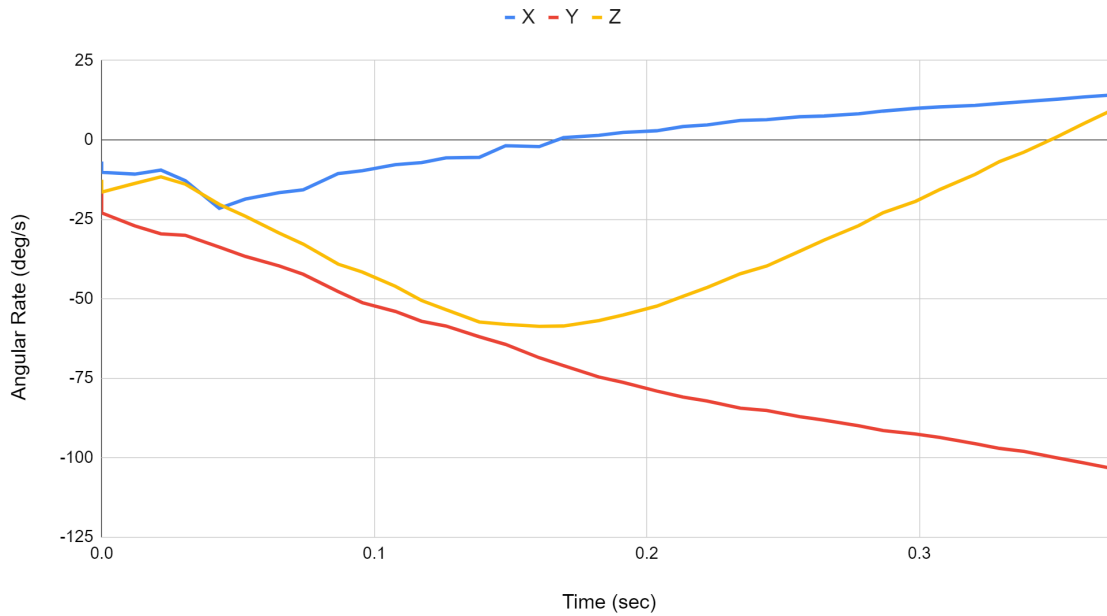


Figure 13. X, Y, Z (roll, yaw, pitch) readings from the gyroscope. These values along with time parameters were converted to Euler angles using quaternions.

Model and Simulation Verification

Although the flight did not take the desirable flight path due to errors in the physical model, we cross-referenced data from the real-world model's flight to the simulation that was run of that model. Figure 15 depicts the simulated flight orientation versus the actual flights in the Y and X axes (yaw and pitch respectively).

Y Axis - Simulation vs Actual Flight

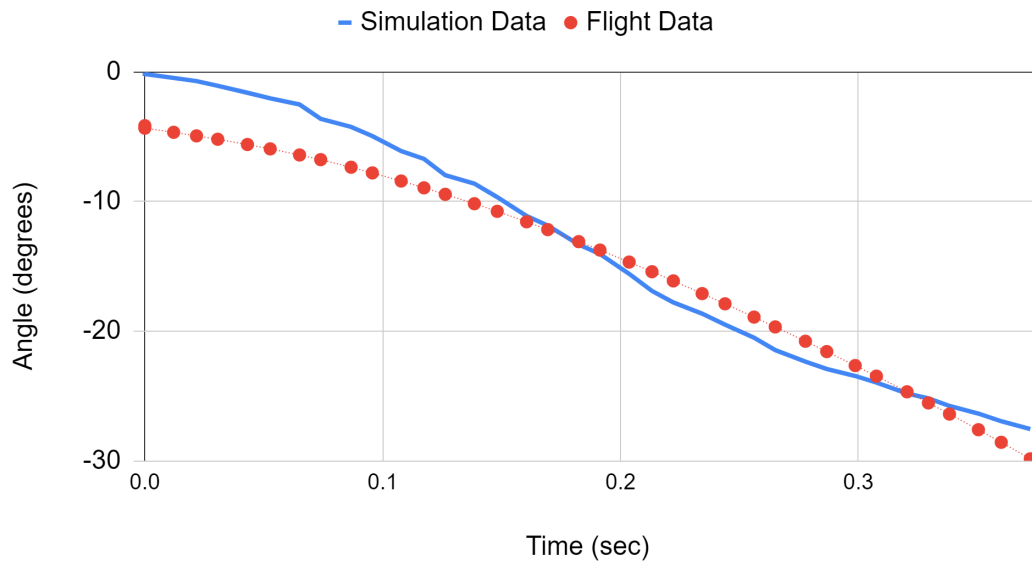


Figure 14. Y-axis (yaw) orientation of the simulated flight of the real-world model and the actual flight.

Z Axis - Simulation vs Actual Flight

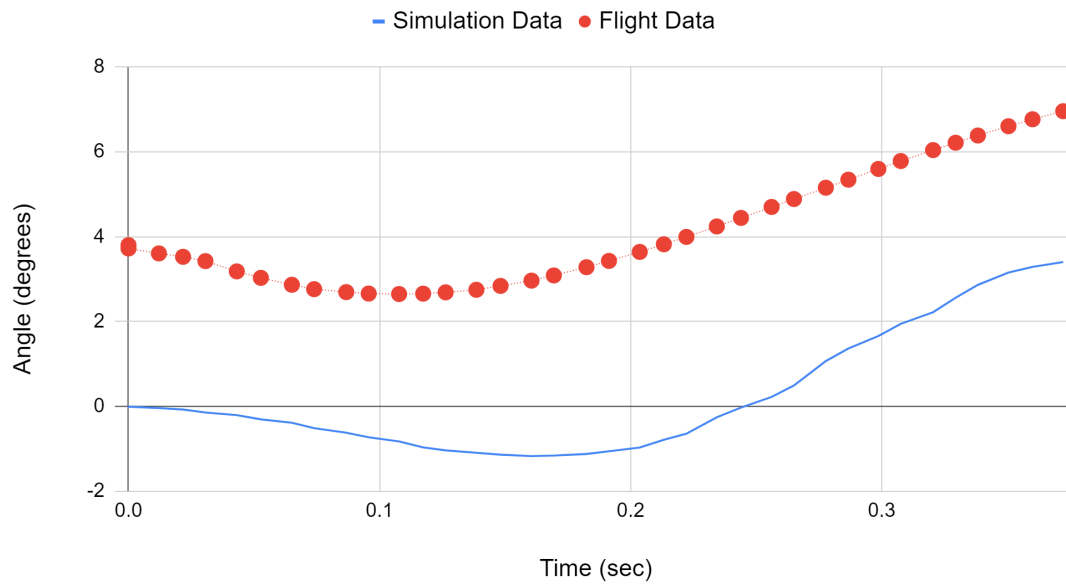


Figure 15. Z-axis (pitch) orientation of the simulation versus the actual flight. The real-world model's orientation values have drifted up about 4 degrees due to the integration of small noise values, causing an offset.

Conclusion

The results of the simulation and model indicate that a PID controller-based navigation system would be a viable option for rockets to use as a control system. Using an accurate simulation that matched real-world data (Figure 15), the three tuning parameters of the PID system could be calculated for an optimal flight. Furthermore, a quaternion-based orientation system was developed to solve gimbal lock and 3D space orientation problems, along with an implementation of a Kalman filter to reduce the effects of noise and increase the robustness of the overall system.

Although the real-world model did not fly as expected, the analysis showed that this was caused by misalignment of the parts and by design errors. We hope to fly another version of the real-world model soon after conducting a few improvements to the model and further improving the simulation for more accuracy.

Simulation Improvements and Future Implementations

The simulation worked very well at predicting the flight path and properties of the real-world model but also has some future implementations and improvements such as:

- Implementing the actual formulas for the fluid dynamic calculations
- Adding a visualization of the flight using animations.

Real-World Model Improvements and Future Implementations

Software

- Faster loop speeds by optimizing the available SRAM onboard the flight computer and creating more efficient functions.
- Add anti-windup to the integral term by clamping the output to the actuator limit of the TVC mount.
- Experiment with different methods of control such as model-predictive control or a linear quadratic regulator.
- Potentially implementing roll control with a reaction wheel, or adding position control.
- A dynamic setpoint that pitches the rocket in a certain direction.

Hardware

- Adding a self-alignment system in the form of a hex driver that inserts into the side.
- Moving avionics upward will lift the center of mass, giving the TVC more torque authority by increasing the lever arm.

Finally, the fact that this project is not the end cannot be overemphasized. The real-world model, control system, simulation, and flight software can have many new improvements, and with more research and development, we hope to see a larger growth within the space industry.

Appendix A

List of symbols, operators, and abbreviations

Symbol	Description
TVC	Thrust-vector-control
PID-Controller	Proportional-Integral-Derivative Controller
GNC	Guidance, navigation, and control.
IMU	Inertial-measurement-unit
3DOF	Three degrees-of-freedom
MMOI	Mass moment of inertia
CAD	Computer-Aided Design
PLA	Polylactic Acid
a	Distance from end of engine and center of mass
A	World reference frame linear acceleration
a_t	Linear acceleration
c_l	Coefficient of lift force
a_{ref}	Reference area of the rocket
d_{air}	Density of air
$e(t)$	Error
e	Motor mount misalignment
F_l	Lift force
F_{motor}	The total force from the rocket motor
$F_{motor(x)}$	x-direction vector of the engine force
$F_{motor(z)}$	z-direction vector of the engine force
G_n	Kalman gain
g	Gravitational constant in m/s ²

K_d	Derivative gain
K_p	Proportional gain
K_i	Integral gain
L	Length of string
m	Rocket mass
p	Period of the rocket swing in the two string pendulum
p_n	Estimate uncertainty
Q	Quaternion orientation
Q_{init}	Base world reference frame quaternion
Q_t	Quaternion orientation at time step t
q	Process Variance
r	Distance from center of mass and string
r_n	Measurement uncertainty
$u(t)$	PID controller output
X_t	Kalman filter estimate
Z_n	Measurement input
α	Roll
β	Yaw
θ	Pitch
σ	The angular position of the TVC gimbal
Δt	Change in time
ω_t	Angular rate

⊗ The Hamilton product operator is the noncommutative multiplication of two quaternions.
For quaternions a and b , $a \otimes b = i$ is defined as

$$i_1 = a_1 b_1 - a_2 b_2 - a_3 b_3 - a_4 b_4$$

$$i_2 = a_1 b_2 + a_2 b_1 + a_3 b_4 - a_4 b_3$$

$$i_3 = a_1 b_3 - a_2 b_4 + a_3 b_1 + a_4 b_2$$

$$i_4 = a_1 b_4 + a_2 b_3 - a_3 b_2 + a_4 b_1$$

Appendix B

Flight 1 PID controller outputs

Time (sec)	Yaw Axis Output	Pitch Axis Output
0	2.001	0.232
0	2.386	0.523
0.012112	2.704	0.468
0.021728	2.922	0.419
0.030656	3.009	0.579
0.043136	3.286	0.99
0.05272	3.514	1.26
0.065064	3.785	1.649
0.073936	4.004	1.929
0.086728	4.427	2.434
0.095664	4.713	2.699
0.107768	5.018	3.127
0.11736	5.298	3.536
0.12632	5.497	3.856
0.138552	5.859	4.319
0.148176	6.127	4.579
0.16052	6.557	4.911
0.169408	6.826	5.123
0.182416	7.267	5.367
0.191368	7.514	5.498
0.203632	7.911	5.666
0.213232	8.203	5.742
0.222208	8.457	5.811
0.23432	8.861	5.884
0.243952	9.104	5.976
0.256144	9.509	6.013
0.265056	9.769	6.022
0.277752	10.199	6.059
0.286728	10.496	6.023
0.298672	10.879	6.068
0.3076	11.168	6.033

0.32048	11.658	6.019
0.329456	11.983	5.948
0.33832	12.285	5.929
0.35048	12.799	5.861
0.36008	13.181	5.761
0.372304	13.723	5.642

Appendix C

Estes D12-0 Specifications

Flight 1 used the Estes D12-0 rocket motor.

Length	7cm
Diameter	24mm
Estimated Weight	40.5g
Total Impulse	20.00 N-sec
Estimated Max Lift Weight	32.90 N
Burn Duration	1.60 sec
Initial mass	40.4 g
Propellant Mass	23.8 g

Table C. Technical Specifications of the D12-0 provided by the manufacturer

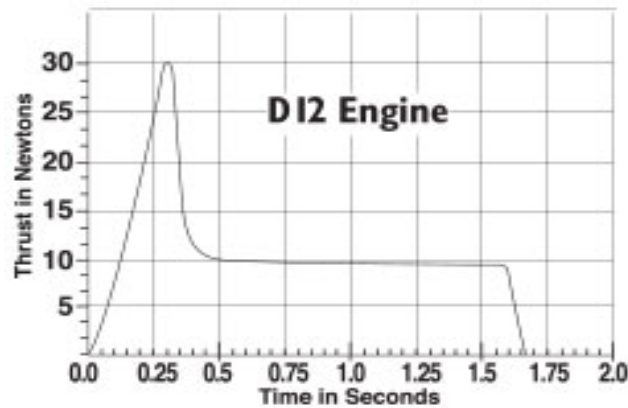


Figure C. Thrust curve of the D12-0 rocket motor provided by the manufacturer

Acknowledgments

We would like to express my gratitude to our mentor Dr. Robert Hermes for his invaluable advice and teachings in model rocketry. We thank our parents for their unending support and our sponsor Ms. Ombelli.

References

- [1] Becker, A. Online Kalman Filter Tutorial. <https://www.kalmanfilter.net/kalman1d.html> (accessed Feb 22, 2022).
- [2] Center of pressure. <https://www.grc.nasa.gov/www/k-12/airplane/cp.html> (accessed Feb 22, 2022).
- [3] D12-0 engines. <https://estesrockets.com/product/001565-d12-0-engines/> (accessed Feb 22, 2022).
- [4] Four forces on a model rocket.
<https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/rktfor.html>
(accessed Feb 22, 2022).
- [5] Koetsier, J. Space inc: 10,000 companies, \$4T value ... and 52% American.
<https://www.forbes.com/sites/johnkoetsier/2021/05/22/space-inc-10000-companies-4t-value--and-52-american/?sh=4861da5955ac> (accessed Feb 22, 2022).
- [6] The lift equation.
<https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/lifteq.html>
(accessed Feb 22, 2022).
- [7] Looney, M. Anticipating and managing critical noise sources in MEMS gyroscopes.
<https://www.analog.com/en/technical-articles/critical-noise-sources-mems-gyroscopes.html> (accessed Feb 22, 2022).
- [8] Maths - conversion quaternion to Euler.
<https://www.euclideanspace.com/maths/geometry/rotations/conversions/quaternionToEuler/> (accessed Feb 22, 2022).
- [9] Maths - quaternions.
<https://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/>
(accessed Feb 22, 2022).
- [10] Niskanen, S. OpenRocket technical documentation.
<http://openrocket.sourceforge.net/techdoc.pdf> (accessed Feb 22, 2022).
- [11] Vectored thrust. <https://www.grc.nasa.gov/www/k-12/airplane/vecthrst.html> (accessed Feb 22, 2022).

Modeling Smoke Plume Dynamics from Imagery

El HEAVN' : The Hearing, Exploring, And, Visualizing of Nubes

New Mexico

Supercomputing Challenge

Final Report

April 6, 2022

Team 15

New Mexico School For The Arts

Team Members

- Madelyn Kingston
- Django Beaudoin
- Lexington Smith
- Elisea Jackson

Teacher

- Mohit Dubey
- Acacia McCombs

Project Mentor

- Stephen Guerin

Table Of Contents

Executive Summary	3
Introduction	3
Forest Fires	3
Previous Approaches	4
Project Goal	4
Model Framework	5
Combustion	5
Fuel	5
Initial NetLogo Model	6
Real-World Data	7
AlertWildfire Imaging Data	7
Controlled Burn Data	8
Smoke Plume Model and Implementation	10
Burning Process	10
Plume Formation	10
Results	11
Conclusion	14
Takeaways	14
Limitations and Errors	14
Future Plans	15
Acknowledgements	15
References	16

Executive Summary

Forest Fires are natural processes that are necessary for healthy resilient forests. However, disasters can happen when wildfire spread is not understood and mitigated. Houses burn down, people are unable to evacuate, and wildlife and civilians can be trapped, injured, or killed. Mapping the spread patterns of wildfires makes it possible to efficiently contain active fires and predict future spread. In this project, we are observing, analyzing, and modeling smoke plumes from forest fires. The most cost-effective and safe method of locating the heat source of a fire is observing smoke and analyzing its behavior through computer modeling. A model such as the one we have created will help first responders react to forest fires in a more informed and effective manner based on camera observations of a smoke plume.

Introduction

a. Forest Fires

Forest fires are an increasingly common and unpredictable disaster. People who live in areas where forest fires are common are at great risk due to delayed evacuation alerts and information during a fire. In the United States in 2018 alone, there were 58,083 forest fires that destroyed over 25,000 structures (including 18,137 residences and 229 commercial structures) and burned 8,767,492 acres of forest (*Wildfires and Acres*, n.d.). These statistics have increased in the past 30 years, correlating to climate change across the United States.

In order for firefighters to successfully control a wildfire, they must have information about the fire location, movement, and areas that are at high risk. Unfortunately, this information is often unavailable to firefighters; they can find themselves going into a dangerous environment completely unaware of conditions and sometimes without the ability to communicate with their team members via radio in environments (Haynes & Madsen, 2017).

Using computer modeling to understand and visualize smoke plumes can be very beneficial when it comes to trying to predict the path, location, and attributes of fire. Specifically, plume imaging is a way to predict the behavior of a fire from afar. With this technology, firefighters will be safer and better able to protect at-risk areas in a forest fire.

b. Previous Approaches

When we began creating this model, the first thing we had to do was understand what a smoke plume is, how it forms, and what factors are most important to predicting the movement of a fire. We began by reading *Stormscapes: Simulating Cloud Dynamics in the Now* (Hadrach et. al.), which gave us an idea of how clouds form. From this paper, we were able to gain insight into the buoyancy, density, and mass that our modeled plume must have in order to accurately represent smoke from a forest fire.

Initially, we wanted to use this model as a basis for our project and create a simpler and more computationally efficient version directly for smoke plumes that could be used for field deployment. However, we soon discovered that each parameter within the model constituted its own complex derivation and simulation; therefore, any level of simplification would not be viable. Instead, we used the Stormscapes model to understand the principles for smoke and cloud formation and inform our agent-based model.

c. Project Goal

The goal of this project is to accurately model smoke plums, specifically the condensation of water vapor onto atmospheric debris, in order to map a wildfire through remote camera imagery. As trees are consumed by a moving wildfire, they combust. This means heat, sugar, and oxygen become carbon dioxide, water, soot, and ash in the fire. The soot and ash are cloud condensation nuclei (CCN). When combined with water vapor, CCN makes a smoke plume visible. By studying this process and modeling it with agent-based modeling in AgentScript (Densmore, 2012), we are able to gain an understanding of the behavior of smoke plumes and how they might reflect the location and movement of the wildfire.

Additionally, if the smoke plume from a real-world wildfire can be replicated by our model, then the model can be used in real-time to determine the behavior of an active fire.

Model Framework

a. Combustion

The most important aspect of a smoke plume model is the main molecular processes involved with a wildfire, which is combustion. The combustion of wood is a complex process that can vary with wood type, water content, and the temperature at which combustion is occurring. We used the following equation as the basis for combustion in the model:

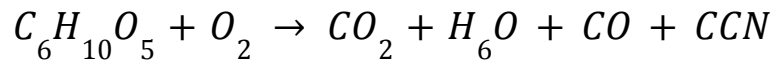


Figure 1. The unbalanced equation for the combustion of a tree

This assumes that the main component of wood is cellulose [1] and that some of the combustions will be incomplete, leaving behind carbon monoxide (*Chemical Composition*, n.d.).

b. Fuel

Combustion occurs at different rates based on the type of forest that is burning. By accounting for this, we are able to approximate the effects of wood type, water content, and temperature on combustion.

The vegetation composition of the Santa Fe National Forest is as follows:

Classification	%	Dominating Tree Types
Timberland	64	Ponderosa-Pine, Douglas Fir
Woodland	36	Pinion, Juniper, Oak

Figure 2. Santa Fe National Forest Composition (Lambert, 2004, #)

Additionally, ponderosa-pine trees make up 22% of all land in the Santa Fe National Forest. Based on these statistics, our model assumes that the forest is entirely ponderosa-pine trees, as this will give an accurate approximation.

The average biomass of a ponderosa-pine tree for trees between 11 and 40 years old is 15.765 kilograms (Grulke & Retzlaff, 1999). Since ponderosa-pine is a softwood (Hardwoods, n.d.), most of the tree is composed of cellulose or $C_6H_{10}O_5$ and will burn in accordance with our combustion approximation (Rowell et al., 2012). The model is tuned to burn in accordance with these data points and can incorporate other fuel types as well given these parameters.

c. Initial NetLogo Model

The first iteration of our model was written in NetLogo 3D. It consists of a basic 3D space with oxygen (blue spheres) moving in Brownian motion. At random, trees combust (turn red), consuming oxygen and releasing water (orange spheres), which rises at an arbitrary rate.

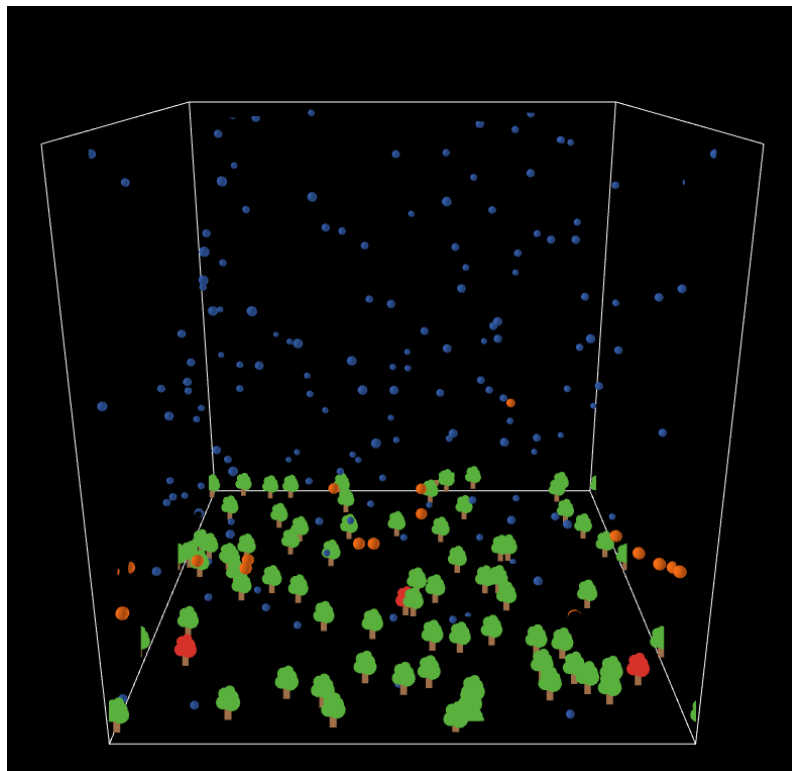


Figure 3. The original framework made in NetLogo 3D that we used as a base for our AgentScript Model

This served as a basic outline from which we could implement greater functionality; however, we quickly found that NetLogo may not be ideal for deployment. Though the framework is great for prototyping, the lack of typical programming conventions proved frustrating and inefficient. These pitfalls resulted in a model that would be difficult to expand. Additionally, despite its relative simplicity, the model is computationally demanding, indicating that it lacks scalability. These factors indicated that switching to a different framework may be the best approach for our project. With that in mind, we pivoted to writing the model in AgentScript. Using the framework of the original NetLogo3D model, we were able to make a more accurate forest and begin constructing the smoke plume.

Real-World Data

a. AlertWildfire Imaging Data

AlertWildfire is an imaging system with cameras all across the Southwest and the Western United States that aims to locate wildfires based on the location of their smoke plumes. The smoke plume is located in space and in relation to the surrounding territory through triangulation as facilitated by multiple camera angles.





Figures 4 and 5: AlertWildfire camera images of the Aztec Springs Prescribed Burn

The current technological capabilities of the AlertWildfire imaging system are to manually draw lines between observed points from the images (features on the smoke plume) and known points shared with a model (geographical features like mountains or stars). Each camera is calibrated by this process and the error is corrected using the Levenberg-Marquardt algorithm. (*Alert Wildfire*, n.d.)

However, the imaging system is limited to manually relating known geographical points in a model of the area to observed points. Our work is to automate the process by which a smoke plume is correlated with a ground location that will increase the response time of such an imaging system. A model like ours will facilitate a more rapid determination of the location of a fire in order to provide that information to first responders and reduce damage and casualties.

b. Controlled Burn Data

After using the AlertWildfire Data to understand the process by which smoke plumes can be observed and measured, one of the obstacles we came across was figuring out how quickly smoke rises, which is an unpredictable and varying number. We completed a controlled burn of our own to gather specific data about fire and smoke behavior. We used a combination of dead needles and live branches of

a ponderosa pine tree in the school parking lot to simulate a forest fire in a very small and controlled manner. The wind during this experiment was moving at a relatively average speed for Santa Fe (7mph) and the setting for the experiment was in an unsheltered location (*US Wind Climatology*, n.d.).



Figure 6: Controlled burn still images

We took measurements of the velocity of smoke with camera footage shot in 4K at 30fps by analyzing several different sections of burn videos in Davinci Resolve 17. Using this tool we were able to visually and manually identify how many distinct features in the plume moved per frame. The speed at

which the smoke moved was approximately 60cm/s (2cm/frame). In creating a physical experiment, we were able to implement the data for how fast the smoke rose on average into our computer model.

Smoke Plume Model and Implementation

a. Burning Process

The trees in our model are all based on ponderosa-pine trees as discussed in the Fuel section. These are softwood trees, are the most common trees in New Mexico forests, and are commonly burned in wildfires. The patches in the model are each 100m^2 and the fire spreads from patch to patch. Based on the fire spread model developed in the 2019 New Mexico School for the Arts Supercomputing Challenge project, “It’s ‘Bout To Get Lit Up In Here: Modeling Forest Fire Risks in Northern New Mexico,” the fire in our model takes 15,000 seconds (250 minutes) to spread from one patch to the next given that the slope between each of the two patches is 0. Each fire begins on the center patch and will ignite any neighboring patch of trees that have not yet caught fire.

This framework makes it very simple to implement wind and slope as factors that affect fire spread; however, we moved on to focus on modeling the smoke plume.

b. Plume Formation

After forming a rudimentary model of the burning forest, we set out to accurately construct our plume. A study on the “Characterization of Combustion Aerosols for Haze and Cloud Formation” (Hallett et al., 2007) found that ponderosa-pine wood burns at a rate of about 10g/100s. This indicates that the average ponderosa-pine tree (15.765kg) will take approximately 43 hours to burn entirely and will release smoke for that entire duration (Hallett et al., 2007). The concentration of condensation nuclei (CN), all particulate matter in clouds that includes hydrophilic and hydrophobic particles in the smoke from a ponderosa-pine tree, is 50,000 to 500,000 CN / cm^3 (Schmale et al., 2017). The ratio of CCN/CN in

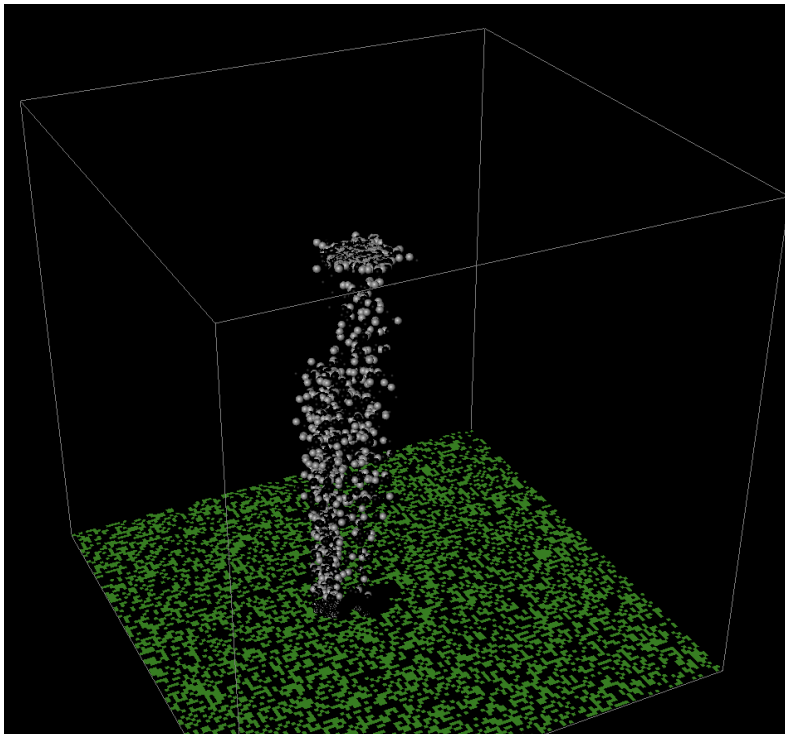
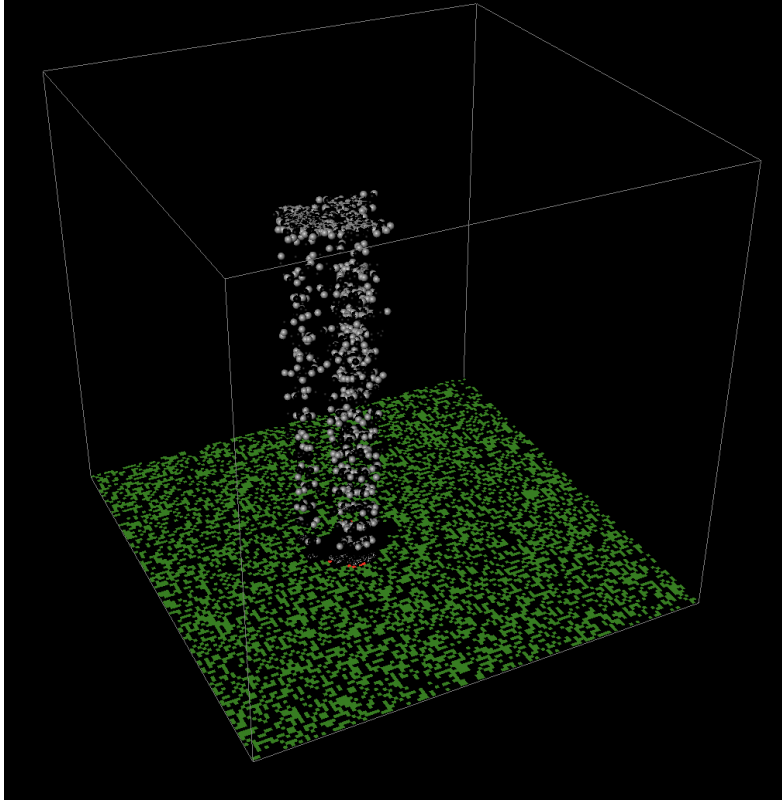
smoke is 0.72; therefore, the density of CCN in smoke can be estimated at 162,000 CCN/cm³. This smoke, with a CCN density of 162,000/cm³, rises at a rate of 1 Liter/second (Hallett et al., 2007).

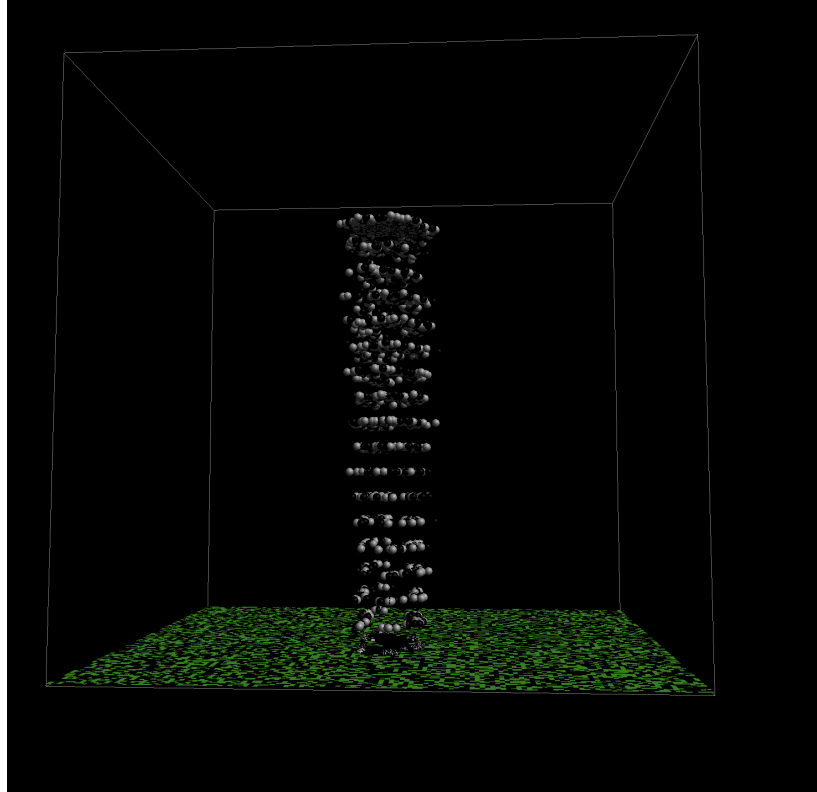
Scaling all of this information to the size of the model produces the following values: each tick is 1,000 seconds, burning patches will release 2,430,000,000,000 CCN particles each tick, and burning patches will release 0.7166 liters of water each tick. These quantities of CCN and water are represented by a single black sphere and a single gray sphere respectively in the model. The smoke rises at a rate of 600m/tick, as informed by our controlled burn.

The fire smoke cools at 1 degree Celsius per 100 meters of altitude (Curtis, n.d.). The smoke stops rising when the temperature reaches the dew point of -2 degrees Celsius, which is the yearly average for Santa Fe (*Climate & Weather Averages in Santa Fe, New Mexico, USA*, n.d.). Currently, we assume that the temperature of the smoke plume begins at 100 degrees Celsius immediately after the water within the tree vaporizes and is released as smoke. The implementation of these parameters into our model creates a functional smoke plume in our model that resembles the behavior of smoke plumes for real-world wildfires.

Results

Our final plume model correlates to the spread of a forest fire. We were able to test the applicability of our model framework by altering parameters, such as the density of the forest (percentage of trees, or number of 100m² patches that contain trees), the vertical velocity of the smoke, the initial temperature of the smoke (how hot the fire is burning), the dew point (to correspond to a particular day), the characteristics of the trees in the model (for example, juniper instead of pine trees), and the speed of fire spread.





Figures 7, 8, and 9: Three different angles of the smoke plume from a ponderosa pine tree forest where CNN are black spheres and Water is gray spheres

After successfully being able to burn our forest and model a plume, we are able to begin analyzing its movement and try to decipher the heat source. When a smoke plume rises above a fire, the concentration, visual appearance, and movement of the plume correlate to the location of the fire and its movement. Just as we were able to determine the location of a plume by beginning with information about the fire, the model can be reverse engineered to begin with information about a plume and determine the characteristics of a fire. Being able to analyze this backwards in the model proved challenging; therefore, our ability to obtain data about the effectiveness of the model in this particular application was hindered. Many of the environmental and computational aspects of our plume were more complex than we had anticipated. As a result, we plan to continue working to analyze our plume and forest fire in order to make this possible.

Even so, we were able to implement all of the computational structures for this model and have created a useful structure for fire spread prediction in the field. The model can be used as an educational

tool for understanding fire spread and can be tuned to fit specific conditions in other forests.

Conclusion

a. Takeaways

In conclusion, we were able to make a functional plume and model a forest fire in AgentScript that can be used as an educational resource for discussing and learning about wildfire spread and a framework for automating the translation of smoke plume imagery into real-time data about the location of a wildfire. Through several iterations of our model, we were able to identify and model the most impactful parameters for smoke formation: tree composition, atmospheric dew point, smoke particle temperature, and vertical velocity of smoke particles. Finally, the work that we completed fostered a greater understanding of scientific and computational practices, laying a strong foundation for future work and research.

b. Limitations and Errors

Complex systems such as wind, condensation onto CCN, and smoke plume dispersion could not be feasibly implemented due to the scope and timeline of our project. Currently, there is no representation for wind in the model, and water condensation is approximated by ceasing the vertical motion of particles when said particles reach the dew point. Modeling the dispersion of smoke particles is computationally demanding and further optimization will be required to account for all the particles produced during a wildfire. At the moment, the number of smoke particles visible in the model is limited to maintain functionality. With further research and work, we hope to implement these systems.

c. Future Plans

As our timeline permits it, we intend to implement the systems mentioned in our limitations and errors section. These systems of wind, condensation, and particle dispersion will increase the accuracy and applicability of our model. Additionally, we hope to implement more ground fire spread aspects into our model based on the framework developed in “It’s ‘Bout To Get Lit Up In Here: Modeling Forest Fire Risks in Northern New Mexico.” These aspects include terrain slope, GIS compatibility, and wind speed effect on ground fire spread. This will allow us to simulate smoke plumes in real-world environments, such as the Santa Fe National Forest.

We would also like to improve the current visualization abilities that our model provides. While the model currently visualizes the structure of the smoke plume functionally, this model would best develop into a more computationally advanced version that could more accurately predict movement and hear heat sources within the smoke.

Finally, we want to eventually add sound to our model through sonification, perhaps as a project in future years. We looked at several different examples where scientists are already using sonication to display data. One example of this is the solar wind radio made with parameter mapping (*Sonification: Data Like You've Never Heard Before*, 2013). Parameter mapping refers to relating specific aspects of the data back to a specific variable in the music. The size and spread of the fire could be directly related back to the tempo and pitch of the music and then communicated in the form of music to firefighters in the field (*What Is Sonification?* | *Arts and Architecture Research*, n.d.).

Acknowledgements

We would like to thank our teacher sponsor, Acacia McCombs, for providing the physical resources and support that were necessary to complete this project, our mentors, Mohit Dubey and Stephen Guerin, for their expertise and guidance with the code and scientific research involved in our project, and the Supercomputing Challenge, for their support and resources throughout the project.

References

1. Alert Wildfire. (n.d.). ALERT Wildfire: Home. Retrieved April 5, 2022, from <https://www.alertwildfire.org>
2. *Chemical Composition*. (n.d.). *Chemical composition • Learning Content • Department of Earth Sciences*. Retrieved April 5, 2022, from https://www.geo.fu-berlin.de/en/v/iwm-network/learning_content/watershed-resources/ressource_biomass/chemical-composition/index.html
3. *Climate & Weather Averages in Santa Fe, New Mexico, USA*. (n.d.). Time and Date. Retrieved April 5, 2022, from <https://www.timeanddate.com/weather/usa/santa-fe/climate>
4. Curtis, G. (n.d.). *Atmospheric Thermodynamics Cloud Condensation Nuclei - ppt download*. SlidePlayer. Retrieved April 5, 2022, from <https://slideplayer.com/slide/12441091/>
5. Densmore, O. (2012). *AgentScript*. AgentScript. <https://agentscript.org/>
6. Grulke, N. E., & Retzlaff, W. A. (1999, August 30). Changes in the physiological attributes of ponderosa pine from seedling to mature tree. *Tree Physiology*, 21.
7. Hallett, J., Hudson, J. G., & Rogers, C. F. (2007, June 7). Characterization of Combustion Aerosols for Haze and Cloud Formation. *Aerosol Science and Technology*. <https://www.tandfonline.com/doi/pdf/10.1080/02786828908959222>
8. Hardwoods, N. (n.d.). *Ponderosa Pine - North American Hardwood Lumber Manufacturing and Distribution*. Northwest Hardwoods. Retrieved April 5, 2022, from <https://northwesthardwoods.com/products/ponderosa-pine/>
9. Haynes, H. J.G., & Madsen, R. S. (2017, January). Wildland/Urban Interface: Fire Department Wildfire Preparedness and Readiness Capabilities. *National Fire Protection Association*, 77. <https://www.nfpa.org/-/media/Files/News-and-Research/Fire-statistics-and-reports/Emergency-responders/osWUI2017.pdf>

10. Lambert, D. (2004, December). Forest Resources of the Santa Fe National Forest. *Rocky Mountain Research Station*, 13.
https://www.fs.fed.us/rm/pubs_series/forest_resources/santa_fe.pdf
11. Rowell, R. M., Pettersen, R., & Tshabalala, M. A. (2012, September). Cell Wall Chemistry. *Routledge Handbooks Online*. <https://www.routledgehandbooks.com/doi/10.1201/b12487-5>
12. Schmale, J., Henning, S., & Henzing, B. (2017, March 14). Collocated observations of cloud condensation nuclei, particle size distributions, and chemical composition. *National Library of Medicine*.
[https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5349251/#:~:text=Cloud%20condensation%20nuclei%20\(CCN\)%20are,well%20as%20radiative%20budgets1](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5349251/#:~:text=Cloud%20condensation%20nuclei%20(CCN)%20are,well%20as%20radiative%20budgets1)
13. *Sonification: Data like you've never heard before*. (2013, July 23). Earthzine. Retrieved April 5, 2022, from <https://earthzine.org/sonification-data-like-youve-never-heard-before/>
14. *US Wind Climatology*. (n.d.). National Centers for Environmental Information. Retrieved April 5, 2022, from <https://www.ncdc.noaa.gov/societal-impacts/wind/>
15. *What is Sonification? | Arts and Architecture Research*. (n.d.). Arts and Architecture Research | Research at Penn State's College of Arts and Architecture. Retrieved April 5, 2022, from <https://aaresearch.psu.edu/artist/sonifications-of-the-universe-and-more/what-is-sonification>
16. *Wildfires and Acres*. (n.d.). National Interagency Fire Center. Retrieved April 5, 2022, from <https://www.nifc.gov/fire-information/statistics/wildfires>

***DNA Methylation with Machine Learning: Prediction of Alzheimer's Disease
and Novel Gene Therapeutics***

Team 34, La Cueva High School

April 6, 2022

New Mexico Supercomputing Challenge Final Report

Team Members:

- Aditya Koushik
- Abitpal Gyawali

Teacher:

- Yolanda Lozano

Table of Contents

Executive Summary	3
Introduction	4
Our Project - Project Idea	4
Background - Epigenetics, DNA Methylation, Alzheimer's Disease	4
Epigenetics	5
DNA Methylation	5
Differentially Methylated Regions (DMRs)	6
Purpose	6
Materials	6
Methods	9
Code Snippets - Data Filtering	9
Identifying the top 10 DMRs: Mutual Information	9
Identifying the top 10 DMRs: ANOVA F-test	10
Code Snippets - Machine Learning Classification	12
Random Forest & Decision Tree:	13
Logistic Regression:	14
Gradient Boosting:	15
Code Snippets - Hyperparameter Tuning	16
Machine Learning - Hierarchical Clustering	18
Results	20
Exploratory Data Analysis	23
Machine Learning Classification Results	24
Machine Learning Hierarchical Clustering Results	28
Gene Predictions	33
Gene Interaction Network	33
Conclusion	37
Acknowledgements	38
Bibliography	39

Executive Summary

Alzheimer's Disease (AD) is the most common form of dementia (loss of memory and other cognitive abilities) as it accounts for 60-80% of dementia cases. Over 50 million people worldwide live with Alzheimer's Disease and currently there is no cure/treatment, except a recent controversial monoclonal antibody therapy (Aduhelm). Therefore, Alzheimer's Disease serves as an important condition to target with various bioinformatics and machine learning methods for both prevention and therapy. Currently, Alzheimer's Disease can be diagnosed by MRI-PET and biomarkers in Cerebrospinal fluid (CSF). However, these methods can be expensive and invasive. Therefore, this project aims to offer an alternative of diagnosing Alzheimer's Disease using blood-based genomic DNA methylation levels and machine learning (Classification and Hierarchical Clustering). We downloaded a publicly available dataset from <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE144858> which contained DNA methylation samples profiled for 300 patients (111 with Mild Cognitive Impair or MCI, 96 with No Alzheimer's Disease - Control, and 93 with Alzheimer's Disease). Using Mutual Information (Information gain), the top 10 Differentially Methylated Regions (DMRs) for only Alzheimer's Disease and Control patients were identified. Next, we trained seven machine learning classification models (Random Forest, K-Nearest Neighbor, Decision Tree, Support Vector Machine, Naive Bayes, Gradient Boosting, and Logistic Regression) on the data of the top 10 DMRs to evaluate the model accuracy in predicting the occurrence of Alzheimer's Disease. We then identified another set of DMRs which includes Mild Cognitive Impairment (MCI - an early stage of Alzheimer's Disease) data using ANOVA F-Test. We built and tested a hierarchical clustering model on the new set of DMRs to see if it could predict the occurrence of Alzheimer's Disease as well as MCI. With classification (only predicted Alzheimer's Disease and Control), the highest scoring model was Gradient Boosting with 90% model accuracy. In addition, our hierarchical clustering model could predict Alzheimer's Disease, Control, and MCI with 86% accuracy. Finally, we used the Cytoscape software and MCODE to predict the interactions of genes methylated by the top 2000 DMRs and to identify genes (RPL23, RPS13, VARS, and MINK1), that can be used for prediction and therapeutic targeting for Alzheimer's Disease.

Introduction

Our Project - Project Idea

Purpose - Using machine learning models to predict the severity of the patient's Alzheimer's Disease with the DNA Methylation levels, and to predict genes that can be used for therapeutic targets for Alzheimer's Disease.

Background - Epigenetics, DNA Methylation, Alzheimer's Disease

Epigenetics

What is epigenetics?

Epigenetics is a study of changes in the DNA's behavior that are not caused by alterations of the DNA sequence. Rather than causing changes in the DNA sequence, epigenetic processes alter the way the DNA is read. The simplest example of epigenetics can be observed in the difference in the functionalities of muscle cells in comparison to the functionalities of nerve cells. Muscle cells and nerve cells both contain the same replica of the DNA, however their functionalities differ by a large margin. This is because of epigenetics. Different epigenetic changes occur between the muscle and nerve tissues that cause all the genes that create proteins for nerve cells to turn off in the muscle cells and vice versa. In other words, epigenetic processes allow cells to have different functionalities by controlling which genes get expressed.

What causes epigenetic processes to occur?

Epigenetic processes are caused and changed due to environmental factors, behavioral factors, and aging factors. Here's an image from

<https://www.sun.edu.ng/knowledge-update/epigenetic-processes-and-human-health>

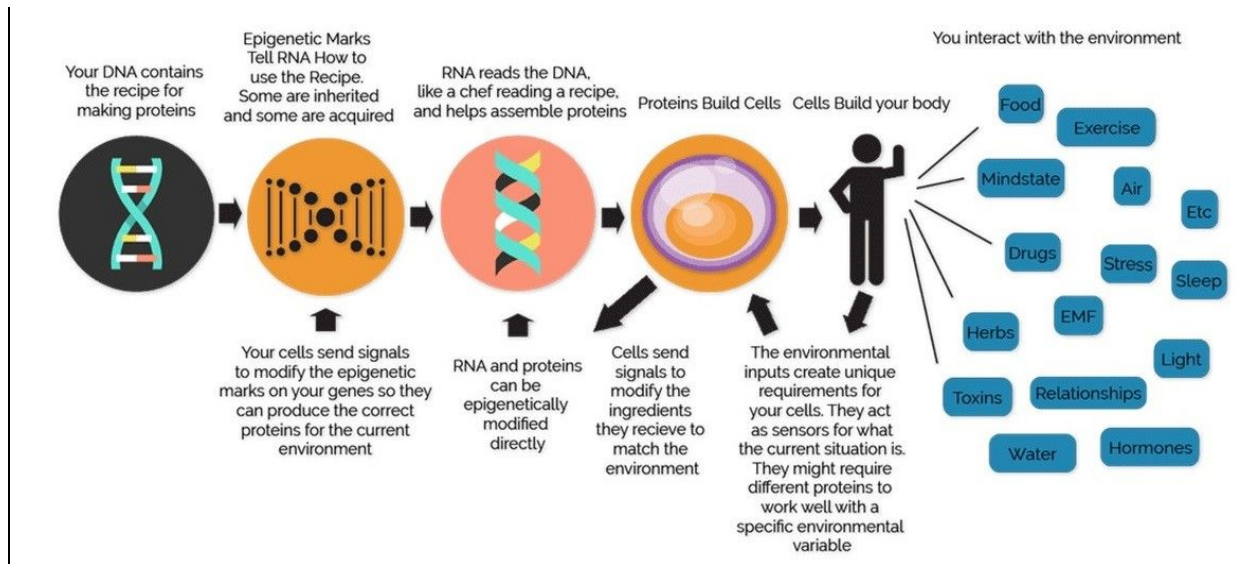


Figure 1

Significance of epigenetic processes

As said earlier, epigenetic processes control which genes are turned ‘on’ and ‘off’. So what happens if an epigenetic process accidentally turns on a gene that’s supposed to be off? This can sometimes be harmless, however, it can also be crucial to the body's ability to live. For example, irregular epigenetic processes are known to cause various diseases such as Alzheimer's Disease, and various forms of cancer.

DNA Methylation

What is DNA Methylation?

DNA Methylation is one of the epigenetic processes that regulates which genes are read/transcribed to make proteins. This is done through addition of methyl groups to the gene to repress its transcription, hence turning ‘off’ the gene. If genes are methylated without control, it can cause irregularities in expression of genes, and can cause different diseases (including Alzheimer’s Disease).

Differentially Methylated Regions (DMRs)

Differentially methylated regions are regions in DNA that have a different methylation level in a healthy patient compared to an Alzheimer's Disease. Methylated regions are methylated locations on genes denoted by cg##### (Eg. cg01813033). DMRs are methylated regions that are only significant to Alzheimer's Disease since they are differentially methylated. For example, if a certain region in the gene has a high methylation in people with Alzheimer's Disease compared to healthy people, it is a DMR. This also applies to low methylation.

Purpose

The purpose of this project is to predict the occurrence of Alzheimer's Disease using Classification and Hierarchical clustering models, and to predict gene interactions of 2000 DMRs for novel gene therapeutic targets for Alzheimer's Disease.

Materials

- Jupyter notebook running Python 3.8
- <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE144858> for data on ~411,000 DNA methylation values across the **whole human genome** in 300 people (publicly available from GEO – gene expression omnibus)
- Cytoscape to predict gene interactions
- MCODE to predict gene interactions clusters
- BioRender to make models of the concepts
- Python Modules used include SciPy, Pandas, NumPy, Matplotlib, Sklearn and Seaborn.

Methods

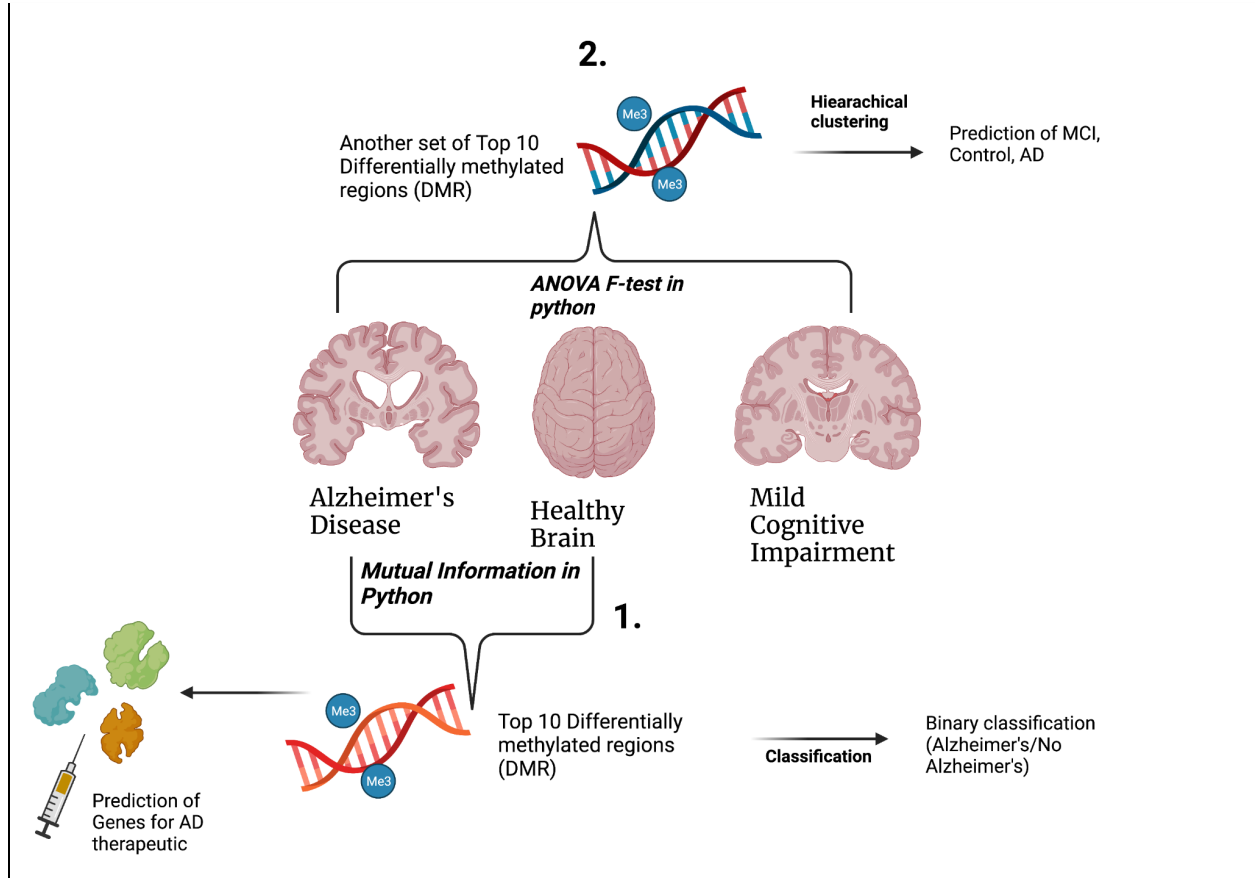


Figure 2

Github: <https://github.com/pauldbd/Supercomputing-challenge-2021-2022>

(1851 Lines of Code in total)

1. Gather materials
 - a. Data was publicly available from GEO - <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE144858>
2. Using Mutual Information, find the top 10 Differentially Methylated Regions (DMRs).
This is only for Alzheimer's and Control (MCI is excluded for classification)
3. Load the data of the top 10 DMRs into a new Jupyter Notebook. Shuffle the data to prevent bias.
4. Split the data into training and testing with 70% - 30% split (70 for training, 30 for testing)

5. Perform machine learning analysis for the data with seven models (K-nearest neighbor, Random Forest, Decision Tree, Support Vector Machine, Naive Bayes, Gradient Boosting, and Logistic Regression).
6. Evaluate the best machine learning model on accuracy, area under ROC curve (True positive vs False positive rate), and confusion matrix
7. Hyperparameter tuning was performed to improve prediction accuracies.
8. Perform Stratified K-fold Cross Validation and compare accuracies with train-test split (normal machine learning).
9. Using the ANOVA F-test, find another set of top 10 Differentially Methylated Regions (DMRs). This is for Alzheimer's, Control, and Mild Cognitive Impair (MCI). This data will be used for hierarchical clustering.
10. Build a hierarchical clustering model with a dendrogram to predict the occurrence of not just Alzheimer's, Control, and Mild Cognitive Impair occurrence using the data of the 10 DMRs obtained with ANOVA.
11. Using Mutual Information for a second time, find the top 2000 DMRs instead of the top 10.
12. Find the corresponding genes that these 2000 DMRs are located in using the original dataset.
13. Using cytoscape, graph the interactions of these genes in a gene interaction network.
14. Use the MCODE algorithm in Cytoscape to find the strongest gene interaction cluster to predict new genes for a potential Alzheimer's Disease therapeutic.
15. Build a Python program to find which genes are most common among DMRs for another way to predict genes.

NOTE: We used hierarchical clustering to predict Alzheimer's Disease, MCI and Control because we found classification was unable to do so (could only predict Alzheimer's Disease vs Control)

Code Snippets - Data Filtering

Identifying the top 10 DMRs: Mutual Information

```
#information gain
from sklearn.feature_selection import mutual_info_classif as MIC
from sklearn.feature_selection import SelectKBest
best = SelectKBest(MIC, k=10)
X_kbest = best.fit_transform(X, y)
print(X_kbest)
cols = best.get_support(indices=True)
features_df_new = X.iloc[:,cols]

print('Original number of features:', X.shape)
print('Reduced number of features:', X_kbest.shape)
```

```
mi_score = MIC(X_kbest,y)
print(mi_score)

[0.19264773 0.17243542 0.17439579 0.16723334 0.16790445 0.16615951
 0.17311981 0.17650831 0.17988855 0.16598824]
```

Figure 3

The Mutual Information (MI) algorithm was used to identify the top 10 DMRs for the binary classification. Figure 3 shows the implementation of the algorithm in Python (we used SelectKBest to get the top 10 DMRs out of the original dataset). Mutual Information is a quantity that measures the effect of one variable on another and is a useful method of feature selection. In this project, Mutual Information was used to identify which Methylated Regions were DMRs and therefore had a significant effect on Alzheimer's Disease. A high mutual information shows the two variables are highly dependent (highly correlated) and a low mutual information shows the two variables are independent of each other. In this project, we are testing the mutual information between a Methylation site and the occurrence of Alzheimer's Disease. Mutual Information is :

$$I(X; Y) = \sum_{x,y} P_{XY}(x, y) \log \frac{P_{XY}(x, y)}{P_X(x)P_Y(y)} = E_{P_{XY}} \log \frac{P_{XY}}{P_X P_Y} .$$

http://www.scholarpedia.org/article/Mutual_information

where x and y are variables in question, and $PXY(x,y)$ is the joint probability distribution between them. In more simple terms, Mutual Information can be defined as:

$$MI(\text{feature}; \text{target}) = \text{Entropy}(\text{feature}) - \text{Entropy}(\text{feature}|\text{target})$$

where entropy is a measure of uncertainty. Mutual information is essentially a measure of the loss of entropy, or uncertainty. The Mutual Information Score is a value between 0 and 1, where a higher value will indicate a better score and therefore a better DMR. In the screenshot, the mutual information scores are fairly low, which is one of the biggest limitations to this project. The data we originally downloaded may have flaws which could have reduced the overall MI score. However, machine learning models may still perform well with less ideal data through hyperparameter tuning (discussed later). The top 10 DMRs identified through MI were fed into the 7 classification models (see results).

Identifying the top 10 DMRs: ANOVA F-test

```
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import SelectKBest
fvalue_Best = SelectKBest(f_classif, k=10)
X_kbest = fvalue_Best.fit_transform(X, y)
print(X_kbest)
cols = fvalue_Best.get_support(indices=True)
features_df_new = X.iloc[:,cols]

print('Original numbermm of features:', X.shape)
print('Reduced number of features:', X_kbest.shape)
```

Figure 4

After classification prediction, the next step of our project was to predict not just Alzheimer's Disease/Control, but also predict Mild Cognitive Impair (3 classes) using hierarchical clustering. To do this, Mutual Information cannot be used since it only works for binary data (Alzheimer's vs No Alzheimer's). For multi-class data (Alzheimer's Disease, Control, Mild Cognitive Impair), the ANOVA F-test was used as a measure of feature selection (finding DMRs) as shown in Figure 4. The ANOVA F-Test finds the level of variance between Alzheimer's Disease, Control, and Mild Cognitive Impair samples to identify significant DMRs. In order for a DMR to be significant (cause Alzheimer's Disease/Mild Cognitive Impair) the ANOVA F-value must be

higher than the F-critical value and the p-value (significance) should be less than 0.01. The results are summarized below in Figure 5:

```
#f crit
import scipy.stats
dfn = 2
dfd = 297
scipy.stats.f.ppf(q=1-0.05, dfn=dfn, dfd=dfd)

3.0261533685653728

fit = fvalue_Best.fit(features_df_new,y)
print(fit.scores_)
print(fit.pvalues_)

[10.07311789 10.54066674 10.95856387 10.86863145 10.76073261 11.73166742
 11.29114208 11.98888219 10.00637878 10.27968937]
[5.85177876e-05 3.77930977e-05 2.55960459e-05 2.78327519e-05
 3.07776216e-05 1.24809625e-05 1.87846149e-05 9.83564781e-06
 6.22926562e-05 4.82312585e-05]
```

Figure 5

To calculate the F-critical value, the degrees of freedom numerator (dfn) and degrees of freedom denominator (dfd) must be calculated (Figure 5).

This can be calculated with:

$$\text{dfn} = a - 1$$

$$\text{dfd} = N - a$$

where a is the number of groups (3 in our case for Alzheimer's Disease, CL, MCI) and N is the number of samples across all groups (300 in our case for 300 patients). The calculated f-critical value was 3.026. Next, we calculated the actual F-value for our own data. The F-values of the top 10 DMRs were in the range of 10-11 which is well above our F-critical value of 3.026 and therefore shows that the DMRs are statistically significant. The associated P-values are also shown in Figure 5 and most of the p values are extremely low (10^{-5} and 10^{-6}) which also demonstrates significance of the 10 DMRs. These 10 DMRs were fed into a hierarchical clustering model, and the findings of the model will be discussed in the Results section.

Code Snippets - Machine Learning Classification

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 28, leaf_size = 1, p =1)
knn.fit(x_train, y_train)

print("{} NN Score: {:.2f}%".format(28, knn.score(x_test, y_test)*100))
knn_predictions = knn.predict(x_test)
scoreList.append(knn.score(x_test, y_test))
acc = max(scoreList)*100
accuracies['KNN'] = acc
confusion_matrix(y_test, knn_predictions)
#better but still bad - cross validation should be able to improve it

28 NN Score: 78.95%

array([[15,  2],
       [ 6, 15]])
```

Figure 6

Figure 6 is an example of building a machine learning model in Python. This classification model is K-Nearest Neighbor. First, data is split into 70% for training the model and 30% for testing using the train-test split from sklearn (a python library). The KNN model from sklearn is imported, and the model is fit (learns patterns in data using the KNN algorithm) for the training data. The trained model is tested on the other 30% of the data, and the prediction accuracy is displayed and the confusion matrix is displayed (explained in detail in the results section). The K-nearest neighbor classifies data by looking at similarity based on Hamming distance, or the neighboring points (if other points near the point at question fall under a certain class, then the point at question may fall into that class). Below is the Hamming Distance function (used for categorical prediction):

Hamming Distance

$$D_H = \sum_{i=1}^k |x_i - y_i|$$

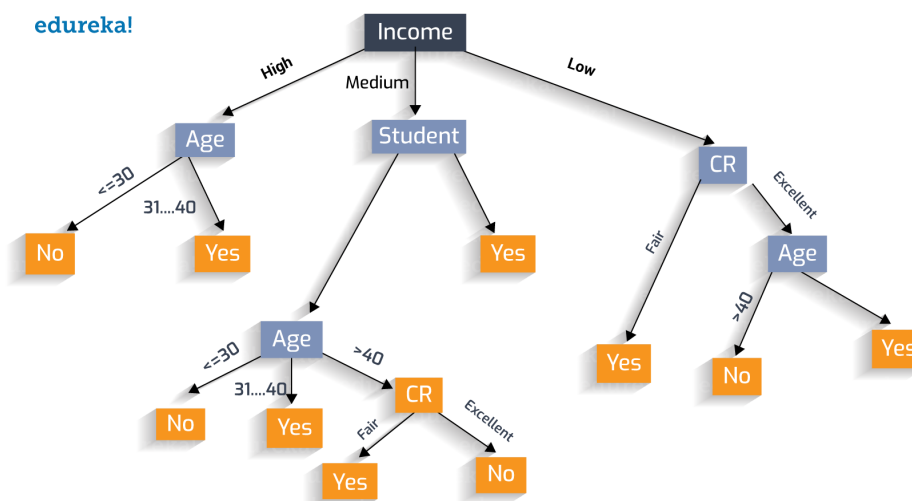
$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

where K determines the number of neighbors. The process for building machine learning models is repeated for all other classification models the same way.

Random Forest & Decision Tree:

Decision trees learn by splitting the dataset into smaller subsets to predict a target value (each condition is called a node, and possible outcomes are called “branches”), hence forming a tree. Random forest consists of many individual decision trees that operate as an ensemble (multiple learning algorithms). Decision trees work by creating a series of cases to aid the model into an accurate prediction. The image below is a great example of how it works.



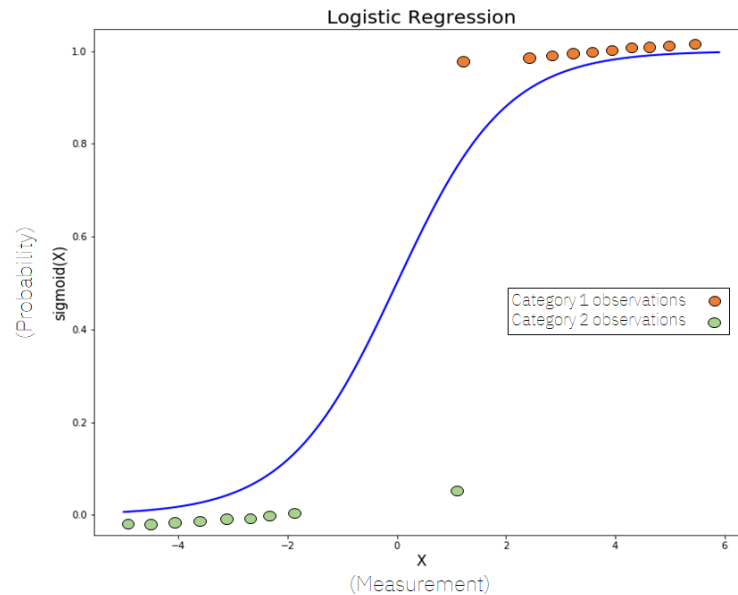
<http://blog.davidvassallo.me/2019/08/06/3-uses-for-random-decision-trees-forests-you-maybe-didnt-know-about/>

Logistic Regression:

Logistic Regression solves binary classification problems although it's a regression function. The basis of logistic regression is the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

The function can take any real value number and map it to a value between 0 and 1 (DNA methylation values can be mapped in this way). The sigmoid curve can be graphed as:



<https://towardsdatascience.com/logistic-regression-explained-9ee73cede081>

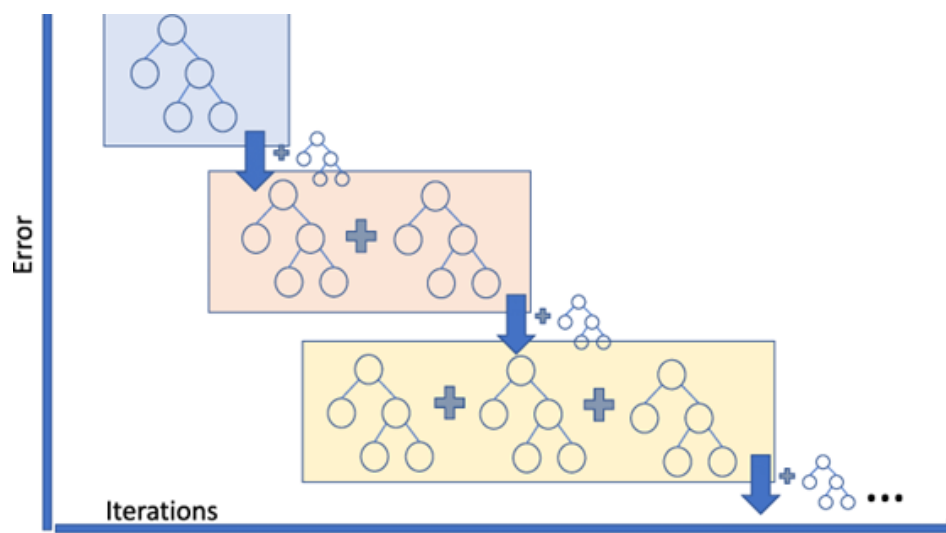
With this method, the location of the point on the curve can determine the classification of that point. In this study, the x-axis would be the DNA Methylation values, and the y-axis would be the probability of Alzheimer's Disease vs Control (No Alzheimer's Disease).

Gradient Boosting:

Examples and Definition

Gradient Boosting is a common way of minimizing the bias error in a classification problem. The system consists of several models that will work recursively to minimize the error. In most cases, the first model of the system is the model that predicts the output based on an input, and the models following the first model will predict the error of the predicted output based on the previous model, and apply that onto the previous output to create a new output.

<https://medium.com/analytics-vidhya/what-is-gradient-boosting-how-is-it-different-from-ada-boost-2d5ff5767cb2>



As in the image above, the error of the system decreases significantly as more and more models are added. However, this can hold a high risk of overfitting, so a learning rate is applied to the models to make sure the model doesn't overfit the train database, and splitting the train and test cases is used to check if the model is overfitting.

Code Snippets - Hyperparameter Tuning

```
gbc = ensemble.GradientBoostingClassifier()
parameters = {
    "n_estimators": [5, 50, 250, 500],
    "max_depth": [1, 3, 5, 7, 9],
    "learning_rate": [0.01, 0.1, 1, 10, 100]
}
gridGB = GridSearchCV(gbc, param_grid = parameters, cv=6, refit = True, verbose = 3)
gridGB.fit(x_train, y_train)
[CV 6/6] END learning_rate=0.01, max_depth=1, n_estimators=500; total time= 0.2s
[CV 1/6] END learning_rate=0.01, max_depth=3, n_estimators=5; total time= 0.0s
[CV 2/6] END learning_rate=0.01, max_depth=3, n_estimators=5; total time= 0.0s
[CV 3/6] END learning_rate=0.01, max_depth=3, n_estimators=5; total time= 0.0s
[CV 4/6] END learning_rate=0.01, max_depth=3, n_estimators=5; total time= 0.0s
[CV 5/6] END learning_rate=0.01, max_depth=3, n_estimators=5; total time= 0.0s
[CV 6/6] END learning_rate=0.01, max_depth=3, n_estimators=5; total time= 0.0s
[CV 1/6] END learning_rate=0.01, max_depth=3, n_estimators=50; total time= 0.0s
[CV 2/6] END learning_rate=0.01, max_depth=3, n_estimators=50; total time= 0.0s
[CV 3/6] END learning_rate=0.01, max_depth=3, n_estimators=50; total time= 0.0s
[CV 4/6] END learning_rate=0.01, max_depth=3, n_estimators=50; total time= 0.0s
[CV 5/6] END learning_rate=0.01, max_depth=3, n_estimators=50; total time= 0.0s
[CV 6/6] END learning_rate=0.01, max_depth=3, n_estimators=50; total time= 0.0s
[CV 1/6] END learning_rate=0.01, max_depth=3, n_estimators=250; total time= 0.2s
[CV 2/6] END learning_rate=0.01, max_depth=3, n_estimators=250; total time= 0.2s
[CV 3/6] END learning_rate=0.01, max_depth=3, n_estimators=250; total time= 0.2s
[CV 4/6] END learning_rate=0.01, max_depth=3, n_estimators=250; total time= 0.2s
[CV 5/6] END learning_rate=0.01, max_depth=3, n_estimators=250; total time= 0.2s
[CV 6/6] END learning_rate=0.01, max_depth=3, n_estimators=250; total time= 0.1s
[CV 1/6] END learning_rate=0.01, max_depth=3, n_estimators=500; total time= 0.3s

gridGB.best_estimator_

GradientBoostingClassifier(learning_rate=1, n_estimators=250)
```

Figure 7

When training our classification models, we initially found the accuracies were fairly low in predicting Alzheimer's vs Control (could be from substandard training data with low MI scores). Hyperparameter tuning was used in order to compensate for this, and to boost model accuracies (Figure 7). We used GridSearchCV for hyperparameter tuning, which runs the model in an iterative process with many parameters to evaluate which parameters would make the model have the highest accuracy. The last line of code in Figure 7 shows what parameters to use to get the best model performance. These parameters are re-applied to the model as shown in Figure 8. Figure 8 below shows the change in accuracy after hyperparameter tuning. This process was repeated for all 7 machine learning models.

Before Hyperparamter tuning for Gradient Boosting model (accuracy is 84.21%)

```
#gradient boost
from sklearn import ensemble
gb = ensemble.GradientBoostingClassifier()
gb.fit(x_train, y_train)
acc = gb.score(x_test,y_test)*100
gb_predictions = gb.predict(x_test)
print("Accuracy of Gradient Boosting: {:.2f}%".format(acc))
confusion_matrix(y_test, gb_predictions)
```

Accuracy of Gradient Boosting: 84.21%

```
array([[15,  2],
       [ 4, 17]])
```

After Hyperparamter tuning for Gradient Boosting model (accuracy is 89.47%)

```
gb = ensemble.GradientBoostingClassifier(learning_rate=1, n_estimators=250)
gb.fit(x_train, y_train)
acc = gb.score(x_test,y_test)*100
accuracies['Gradient Boosting'] = acc
gb_predictions = gb.predict(x_test)
print("Accuracy of Gradient Boosting: {:.2f}%".format(acc))
confusion_matrix(y_test, gb_predictions)
```

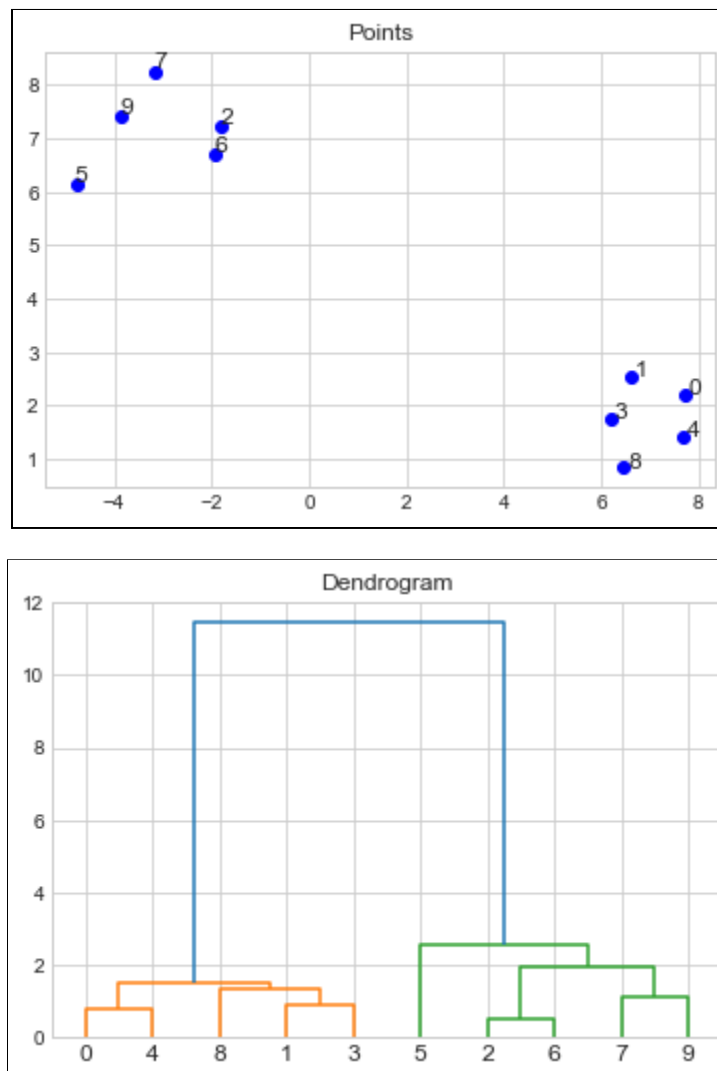
Accuracy of Gradient Boosting: 89.47%

```
array([[16,  1],
       [ 3, 18]])
```

Figure 8

Machine Learning - Hierarchical Clustering

Hierarchical Clustering is a clustering method that aims to cluster a dataset based upon finding the closest points and joining them, and repeating this until every single point on the dataset is in a single group. There are two main types of hierarchical clustering: agglomerative and divisive clustering. Agglomerative clustering is the clustering method described above, in which the model finds the two closest groups/points and joins them to create a new group. However, in divisive clustering, the model starts with a single group consisting of all the points in the dataset, and continually splits them into groups that are the farthest until there are only points left (it is the opposite of agglomerative clustering).



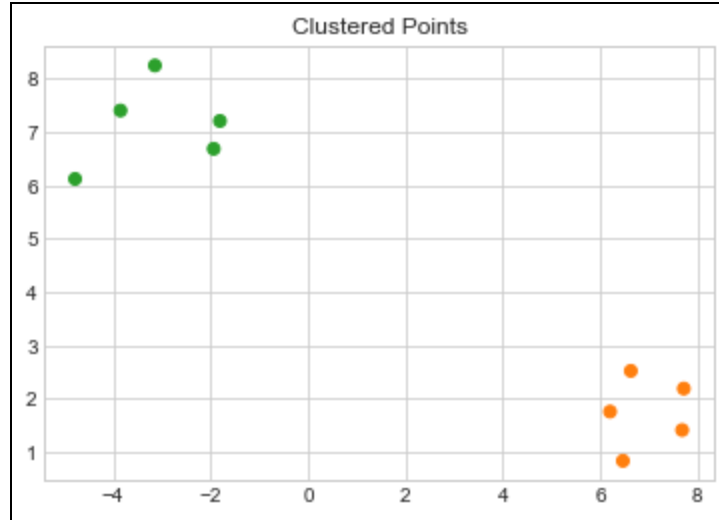


Figure 9

Example

Figure 9 shows examples of how hierarchical clustering works. As you see the graph labeled ‘Points’ consists of 10 points. Additionally, for the purpose of demonstration, the graph is made so that it consists of two separated groups of points (one on the top right, and another on bottom left). The dendrogram shows how the clustering model joined the points to create the final clusters. The closest pair points are joined together to make the first group (points 0 and 4, points 2 and 6, etc). These pairs of points are joined to create a new point (the x and y values of the new point is the average of the two points). Then, this new point is put through to be compared by other groups of points. The length of each line on the dendrogram shows how close the groups/points are (points 2 and 6 have the shortest lines because they were the closest pairs out of the dataset). Then the final graph titled “Clustered Points” shows the final graph with color coordinated points according to their clusters calculated by the model.

Results

Exploratory Data Analysis

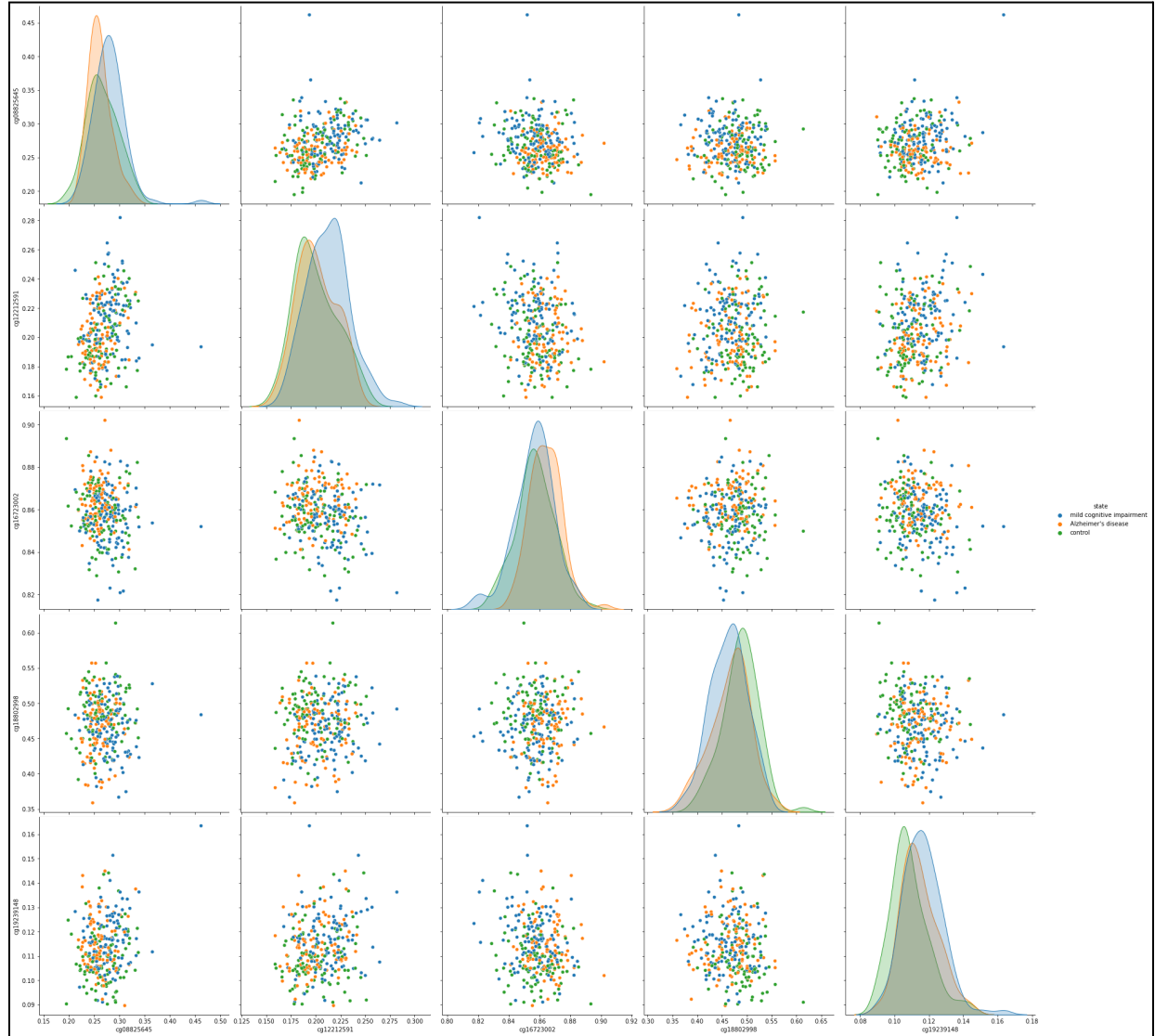


Figure 9

Figure 9 shows a seaborn scatterplot created in Python of the top 5 DMRs which were identified through ANOVA F-test (detailed in methods). The most important graphs are in the diagonal, since they show the distribution of methylation levels for each DMR and what class each distribution curve is classified under. For example, in the last methylated region (cg19239148), the No Alzheimer's curve (shown in green) is behind the Mild cognitive Impair which is shown

in blue. This means that a high methylation level correlates with mild cognitive impair. This can also be visualized with a violin plot and box plot shown below.

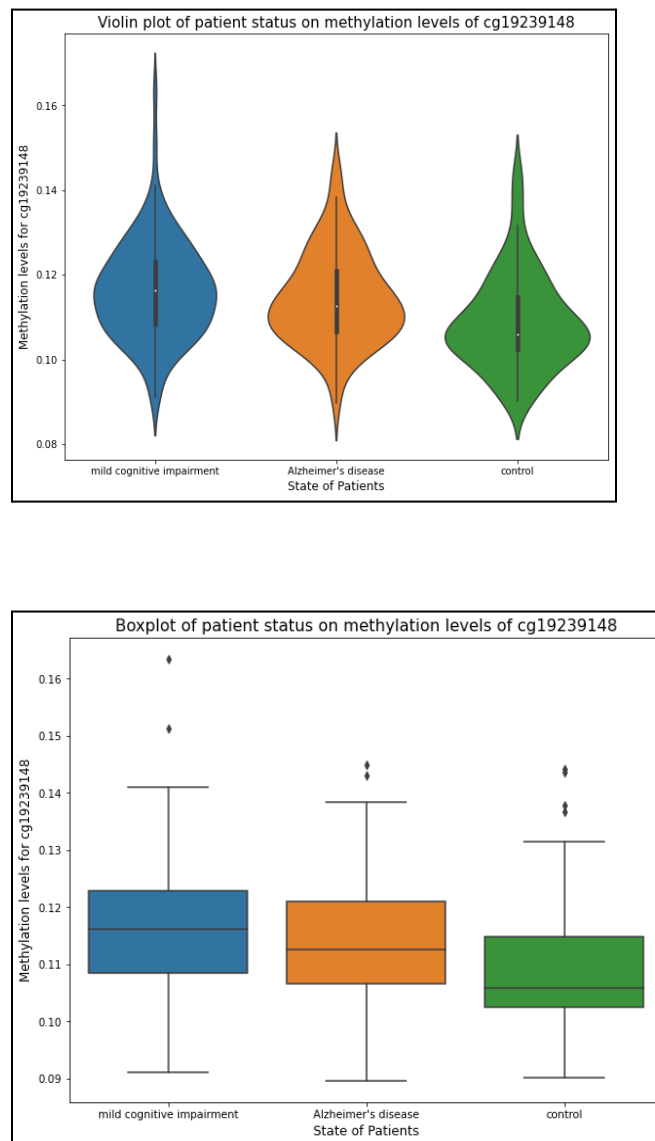


Figure 10

Figure 10 shows the violin plot and the boxplot for the most differentially methylated region in our dataset, **cg19239148** (identified with ANOVA F-test). The violin plot graphs the general distribution of the data as well as the median methylation levels for Alzheimer's Disease, Mild cognitive Impair, and Control (No Alzheimer's Disease). The shape of each plot follows a probability density function, where the widest point indicates a high probability that each sample

will have the given methylation value. A uniform shape shows a normal distribution, where data points are concentrated around the median. From the plots, patients with mild cognitive impairment seem to have the highest DNA methylation levels, then Alzheimer's patients, and finally healthy patients have the lowest DNA methylation levels.

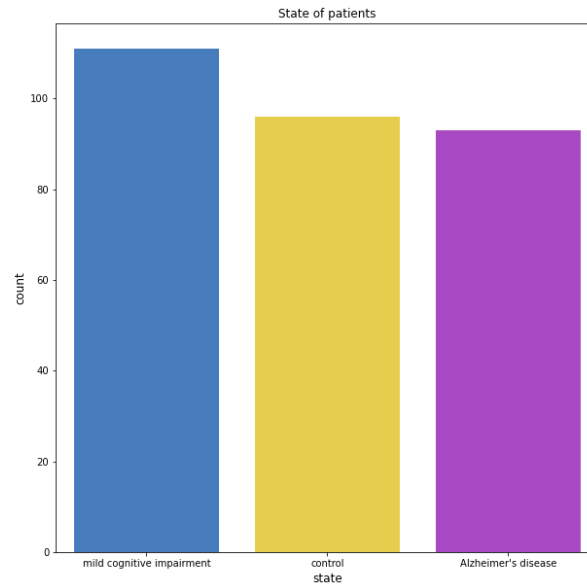


Figure 11

Figure 11 shows the distribution of the patient status in our original dataset. 111 had Mild Cognitive Impair or MCI, 96 had No Alzheimer's Disease - Control, and 93 had Alzheimer's Disease. Imbalances of classes in the dataset will be accounted for with Stratified K-fold Cross Validation (explained in results).

Machine Learning Classification Results

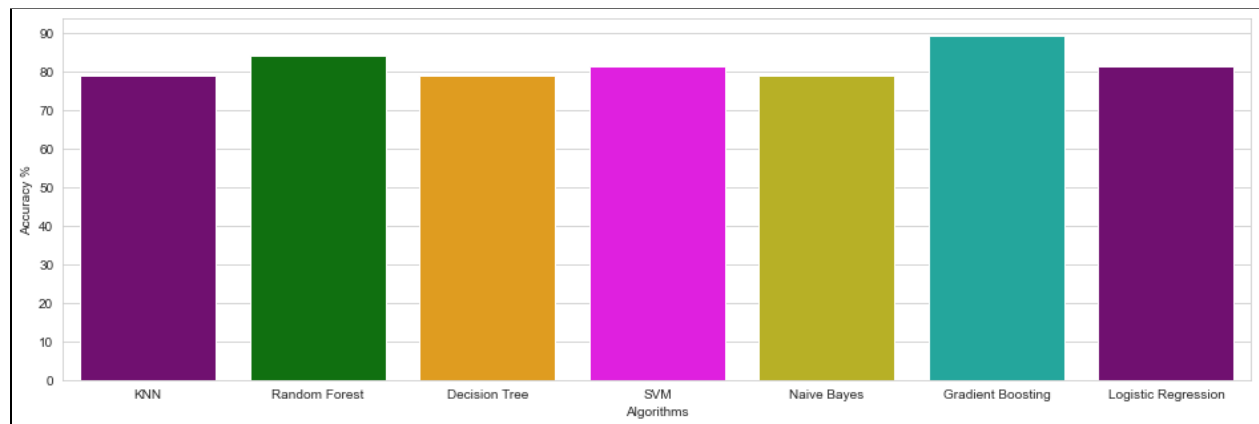


Figure 12

Model accuracies:

'KNN': 78.95, 'Random Forest': 84.21, 'Decision Tree': 78.95, 'SVM': 81.58, 'Naive Bayes': 78.95, 'Gradient Boosting': 89.47, 'Logistic Regression': 81.58

Figure 12 shows the accuracies of 7 machine learning classification models in predicting Alzheimer's Disease vs Control after being trained on the top 10 DMRs (identified with MI). The average accuracy was 82%, and the highest scoring model was Gradient Boosting with 90% accuracy. Note that these were accuracies measured after hyperparameter tuning. Model performance can also be measured through confusion matrices and ROC curves.

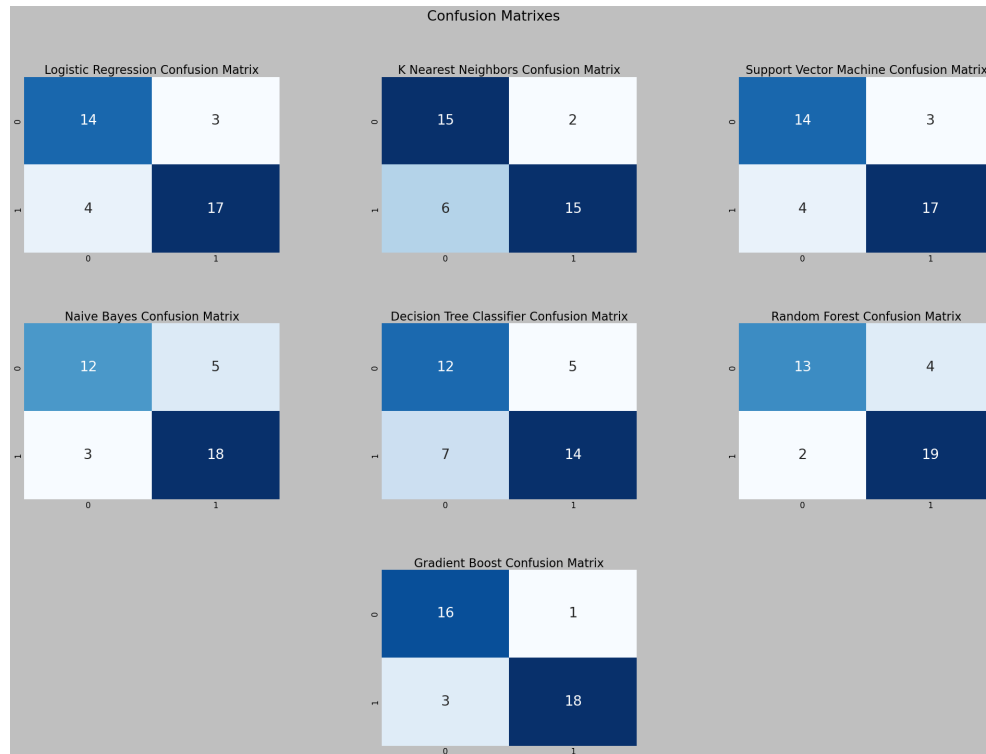


Figure 13

Figure 13 shows the confusion matrix for each of the 7 machine learning models. Confusion matrices evaluate model performance by showing the number of predictions of each class. The table below shows what each number means in each quadrant of the confusion matrix:

	Predicted: Alzheimer's Disease	Predicted: No Alzheimer's Disease
Actual: Alzheimer's Disease	True positive	False Positive
Actual: No Alzheimer's Disease	False Negative	True negative

In order to better visualize the confusion matrix, an ROC plot was created (shown below)

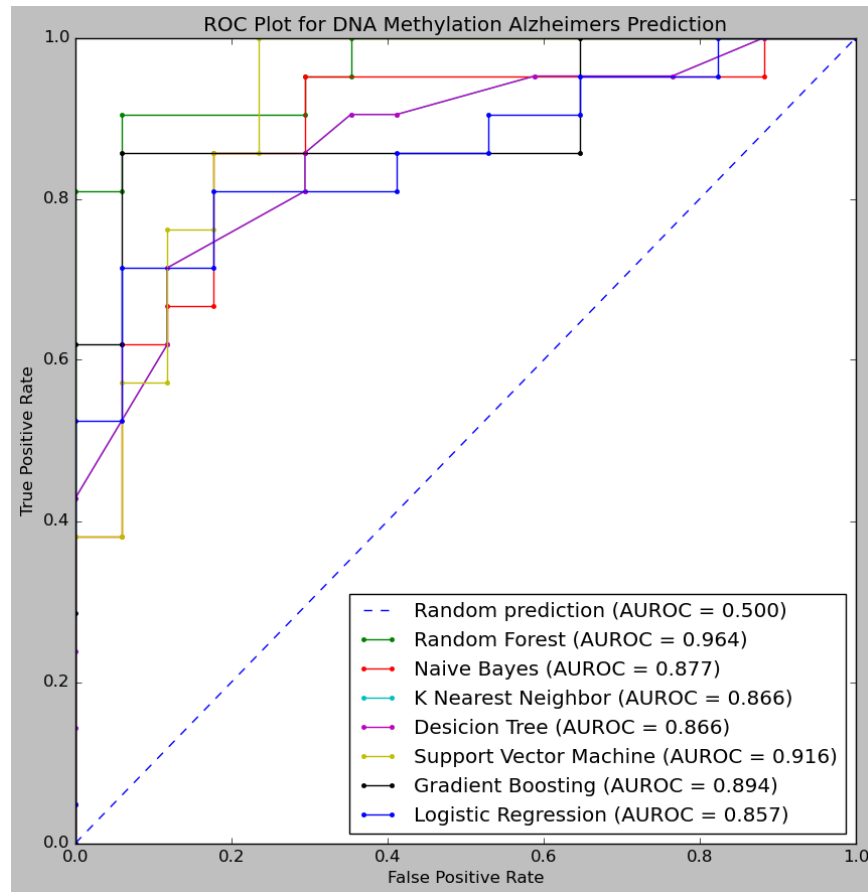


Figure 14

Model performance can also be analyzed using ROC curves (Receiver Operating Characteristic) and AUROC (Area under ROC curve). The ROC curve plots the True positive rate (TPR) vs False positive rate (FPR).

$$\text{TPR / Recall / Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

TPR = (# of True positives)/(True positives + False Negatives)

$$\text{FPR} = 1 - \text{Specificity}$$

$$= \frac{\text{FP}}{\text{TN} + \text{FP}}$$

(False positive rate = (# of False Positives)/(True Negatives + False Positives)

The AUROC is the area under the ROC curve. A good AUROC is near 1, which shows a good measure of separability (Alzheimer's Disease vs Control). A poor model will have an AUROC near 0, which indicates the model gets every prediction incorrect. A model with AUROC of 0.5 cannot make any separation (cannot differentiate between Alzheimer's Disease and CL). **Figure 14 shows the ROC curve for all models. The random forest model had the highest AUROC of 0.964. The average AUROC was 0.891.**

Stratified K-Fold Cross Validation

Next, Stratified K-fold Cross validation was performed in Python. In regular machine learning, the data was split using the train test split (70% of the data used to train models, and the other 30% used to test the model accuracy). Splitting the data with a train-test split has more bias and can lead to overfitting, so Stratified K-fold splits the data in a different manner. K-fold cross validation involves splitting data into k equal folds (shown by Figure 15 below). The first k-1 folds are used for training, and the remaining are used for testing. This is repeated for all k-folds, and the mean of the accuracies of each k-fold is returned. Stratified k-fold is similar, but involves splitting the data into folds not randomly, but based on the number of each class (if fold 1 has 15 tumor samples and 15 non-tumor samples, then fold 2 should have a roughly equal amount). This helps eliminate biases that come with randomly splitting the data.

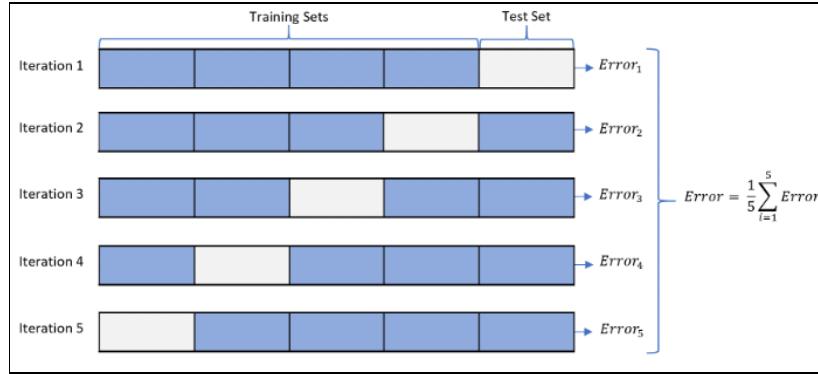


Figure 15 <https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b>

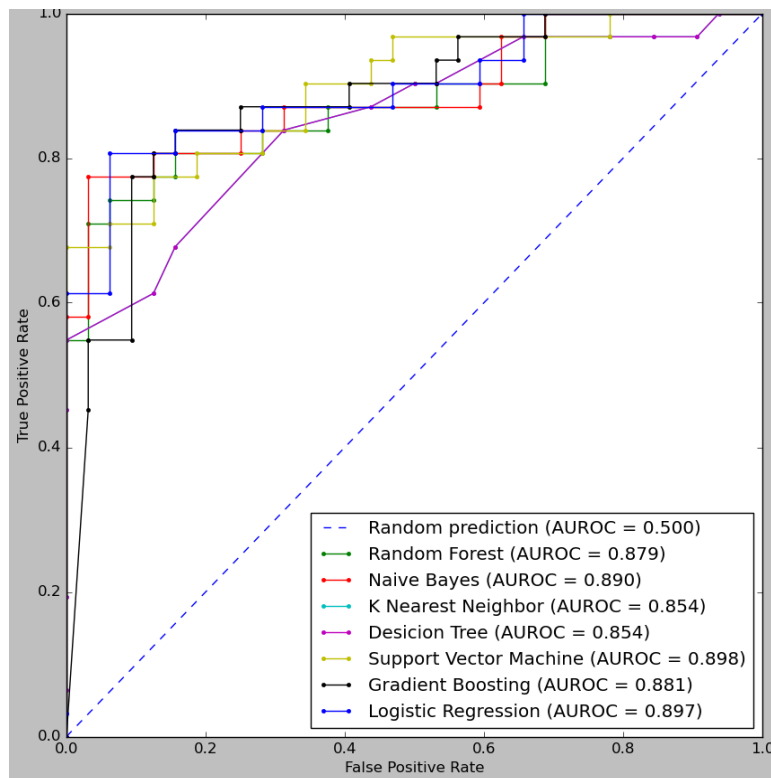


Figure 16

Figure 16 shows the AUROC scores for each of the 7 models after performing Stratified K-Fold Cross Validation. The model with the highest AUROC was the Support Vector machine with a score of 0.898. The average AUROC score was 0.879. This was a little lower than the average for regular machine learning, which was 0.891. This could mean that the models before were slightly overfitting. However, most of the AUROC scores are still fairly high, and random forest

had an AUROC of 0.96 - which has promising potential to predict Alzheimer's Disease in a medical setting.

Machine Learning Hierarchical Clustering Results

Hierarchical Clustering with Alzheimer's Data

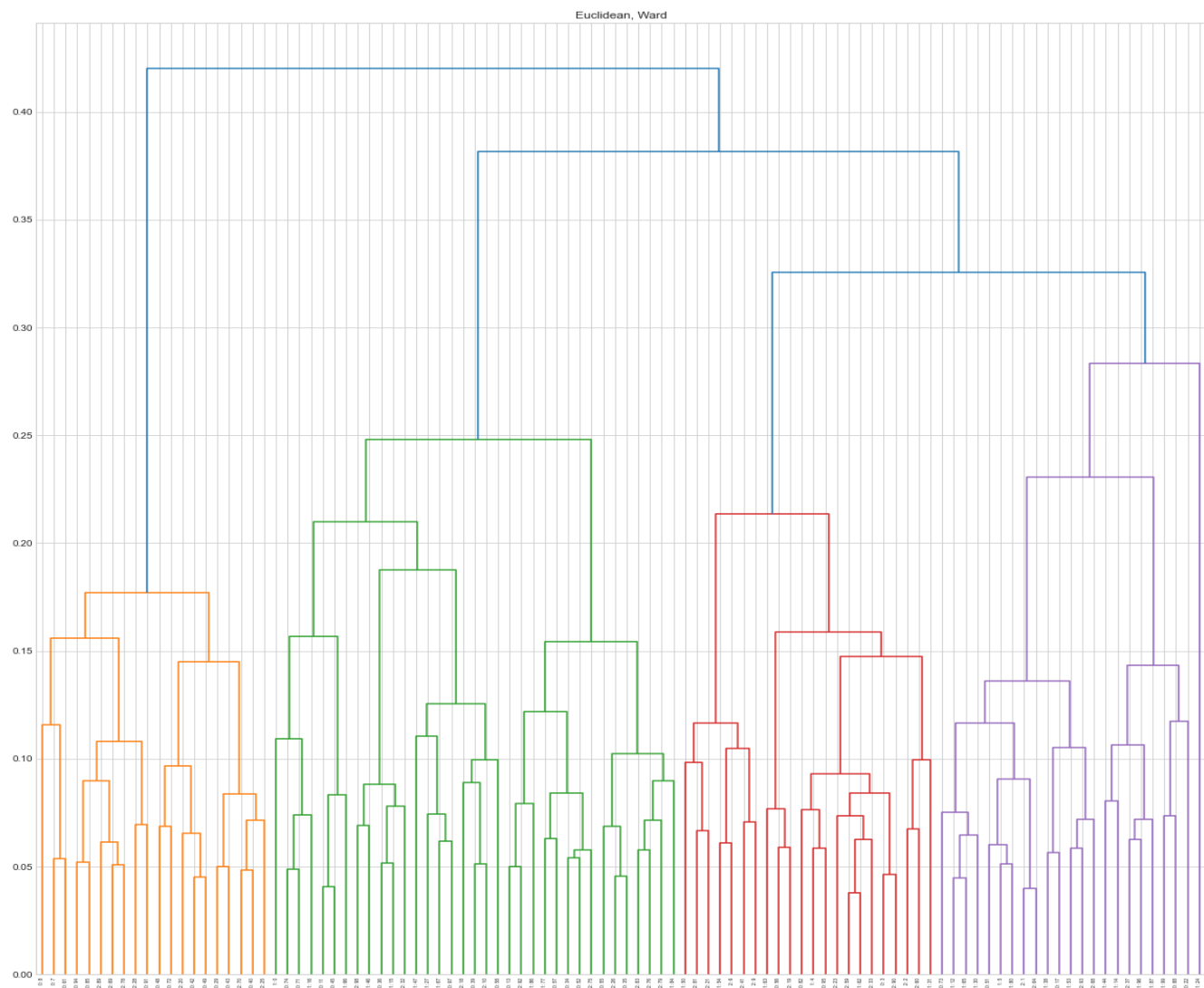
When creating a hierarchical clustering model, the three most important parameters are the following:

- The distance function: which function/formula to use to compute the distance between points (Eg. euclidean, manhattan distance, etc.)
- The joining algorithm: the method used to join/group two points
- The distance threshold OR number of clusters: the distance threshold determines when to cut off the clusters (the distance is the y-axis of the dendrogram), and the number of clusters is the target number of clusters to make

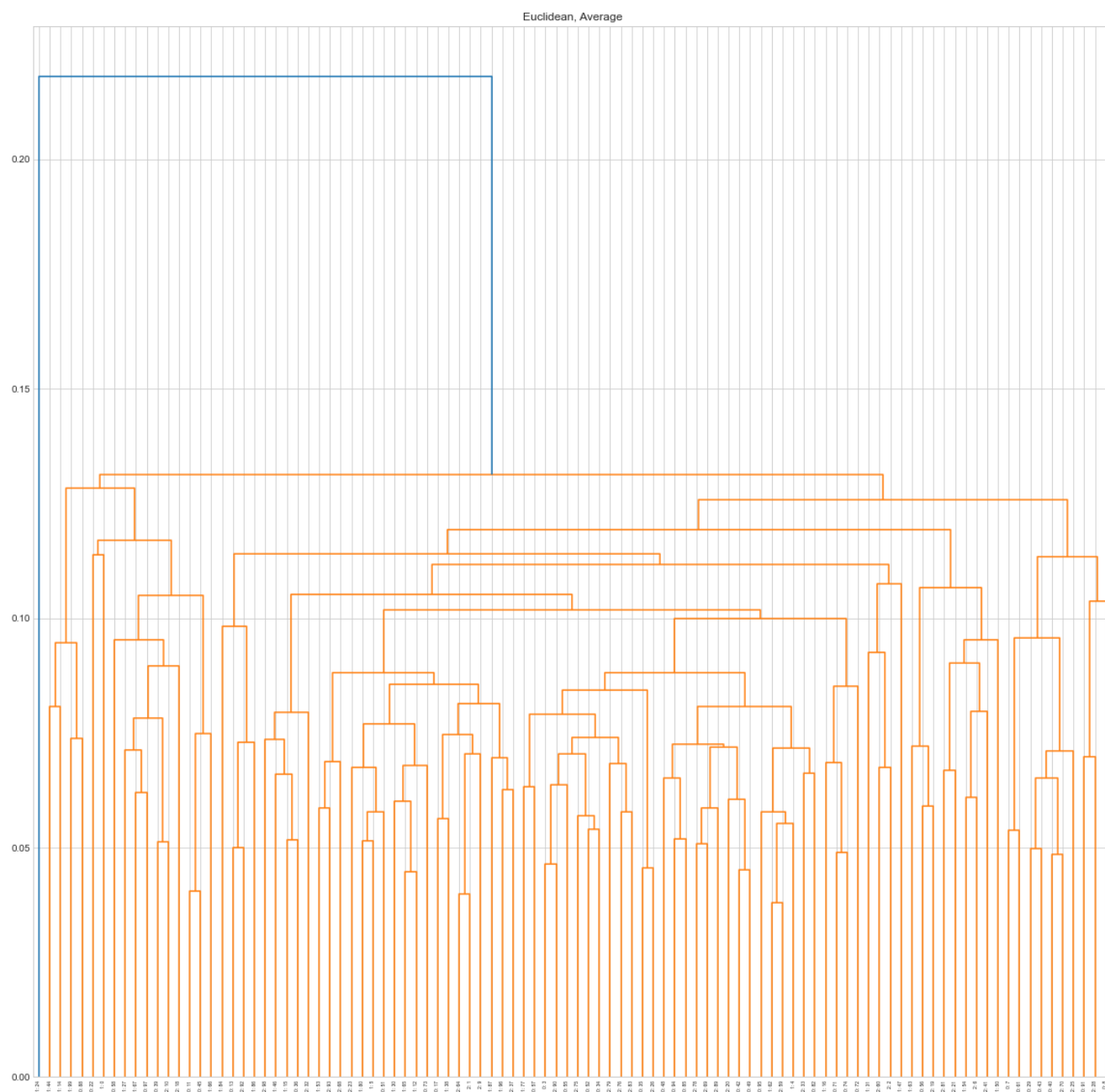
We assigned hierarchical clustering to the task of predicting between the control case, mild imperative impairment, and Alzheimer's (unlike in other models where we let it predict between control and Alzheimer's).

First thing we did with the model was to graph the dendrogram with different joining methods and distance functions.

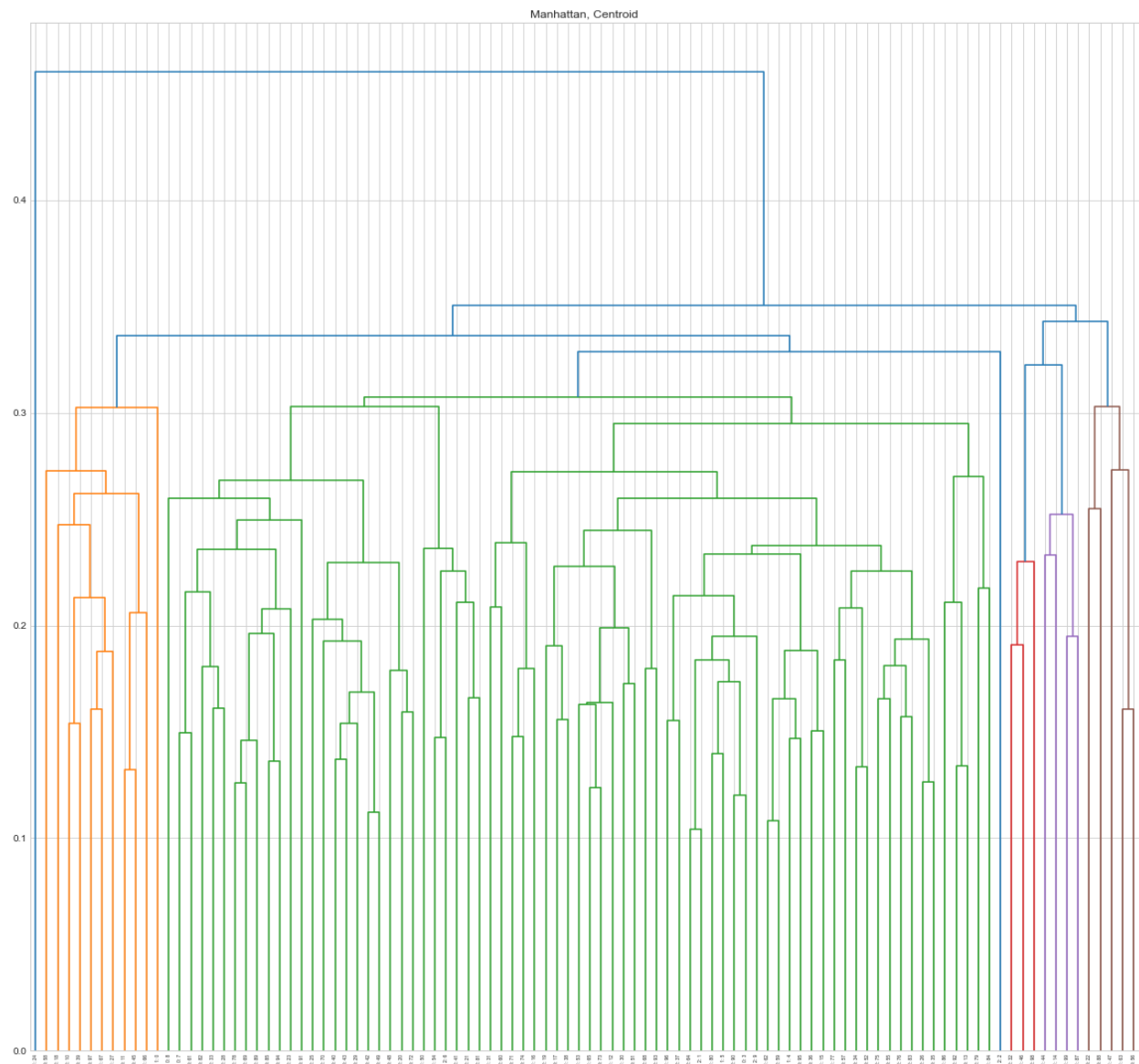
Distance: Euclidean, Joining: Ward



Dist: Euclidean, Join: Average



Dist: Manhattan, Join: Centroid



The dendrograms above were a couple of the dendrograms we collected before creating the final model that included the distance threshold and number of clusters. Based on observations, we decided to use the distance function of euclidean and the merge function of ward to create the final model. The observations were:

- With ward and euclidean, we had an even split between the clusters
- The first merges were consistent, meaning that most of the samples were able to find the first partner at a relatively similar distance

In the final model, we tried restricting the model to a distance threshold and a certain number of clusters (60), and after exploring different parameters with different values, we came to the conclusion that the distance threshold of 0.07 would create a model that created the minimal number of clusters while maintaining the highest accuracy.

```
xs, ys = createData(100)
cluster = AgglomerativeClustering(affinity='euclidean', linkage='ward', n_clusters=None, distance_threshold=0.07)
y_pred = cluster.fit(xs)
def calc_score(y_pred, y):
    arr = {}
    for i in range(0, len(y_pred)):
        arr[y_pred[i]] = [0, 0, 0]
    for i in range(0, len(y_pred)):
        arr[y_pred[i]][ys[i]] += 1
    avg = 0
    right = 0
    for i in range(0, len(y_pred)):
        tot = arr[y_pred[i]][0] + arr[y_pred[i]][1]
        avg += tot
        if (arr[y_pred[i]][ys[i]] > tot - arr[y_pred[i]][ys[i]]):
            right += 1
    return right / (len(y_pred))

print("Accuracy: " + str(100 * calc_score(y_pred.labels_, ys)) + "%");

Accuracy: 86.0%
```

Overall, the accuracy of our hierarchical clustering model in predicting Mild Cognitive Impair, Control, and Alzheimer's Disease was 86%.

]

Gene Predictions

The next step of our project was to predict the top 2000 DMRs using Mutual Information (we only wanted the methylated regions that cause Alzheimer's, so no Mild Cognitive Impair). We found which gene each of the 2000 DMRs correspond to using <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GPL13534>. Because the methylation data is profiled in the whole human genome, we wanted to find which genes our 2000 DMRs are located in. Next, we obtained the name of the genes that the DMRs are located in and graphed the gene interactions in the Cytoscape software. We graphed a gene interaction network since that would allow us to predict genes that cause Alzheimer's Disease and can be used for a potential pharmacogenomic therapeutic for Alzheimer's Disease. It is important to identify new genes that can cause Alzheimer's Disease since that can provide more options for therapeutics. Our objectives to predict genes were twofold:

Objective 1: Predict genes that are known to cause Alzheimer's Disease (using MCODE, Python)

Objective 2: Predict genes that have not yet been explored (using MCODE, Python)

Gene Interaction Network

Before either objective could be completed, first a gene interaction network had to be constructed in cytoscape. The interaction network would show the interactions of genes that the top 2000 DMRs are located in. It is important to look at gene interactions since 1 gene may not have the ability to cause Alzheimer's Disease, but a series of many genes that interact with each other can have the potential to cause Alzheimer's Disease. Additionally, the MCODE algorithm needs a gene interaction network built to make predictions. Below is the gene interaction network for the genes of the top 2000 DMRs. This shows a small glimpse of the center of the network (most important) since the network was too large to attach.

can result in excessive methylation which can cause AD). We are essentially finding the most frequently occurring gene in the list of genes of the 2000 DMRs. The findings are shown below:

[('PTPRN2', 13), ('MINK1', 13), ('IQCE', 12), ('MADL1', 11), ('HPGD', 11)]

The number shows the number of DMRs located in that gene. For example, the PTPRN2 gene has 13 DMRs located in the gene. If a lot of DMRs are found in a certain gene, it has a high probability to cause Alzheimer's Disease. In total, we predicted 15 genes (from MCODE and the Python program). In order to find out which of these 15 genes cause Alzheimer's Disease and which ones cause other disorders, the Gene Cards website was used. The results are summarized in the table below.

Blue Highlight: MCODE genes

Green Highlight: Python Predicted genes

Predicted Genes	Gene Cards Validation (What disorder is the gene known to play a role in)
PLEC	Muscular Dystrophy, Limb-Girdle, Autosomal Recessive, Epidermolysis
TSR1	Epithelial Malignant Thymoma, Dendritic Cell Thymoma, Geleophysic Dysplasia, Sick Building Syndrome
RPL5	Diamond-Blackfan Anemia
RPL23	Brain Glioblastoma Multiforme
SERPBI	Contagious Pustular Dermatitis, Ovarian Cancer
PELO	No disorders in Gene Cards
RPL13A	Sclerosteosis, Spermatogenic Failure

RPS13	Gaucher Disease, Neuroblastoma, Autonomic Nervous System Neoplasm
ATP5F1A	Combined Oxidative Phosphorylation Deficiency 22
VARs	Neurodevelopmental Disorder With Microcephaly, Seizures, And Cortical Atrophy
PTPRN2	Developmental Coordination Disorder
MINK1	Alzheimer's Disease
IQCE	Retinal Degeneration
MADL1	Prostate Cancer
HPGD	Digital Clubbing, Isolated Congenital

Some of these genes have no significant relation to Alzheimer's or any brain disorder. The most important genes (involved in Alzheimer's or other brain diseases) are **RPL23, RPS13, VARs, and MINK1**. MINK1 is proven to cause Alzheimer's, and we were able to predict it- which validated our methodology. RPL23, RPS13, and VARs aren't known to cause Alzheimer's Disease outside of a few published papers, but can still be novel genes that could be targeted via therapeutics. The most significant finding to this project however was that MINK1, one of the genes we predicted, is known to cause Alzheimer's Disease. The link to the gene card validation is here: <https://www.genecards.org/cgi-bin/carddisp.pl?gene=MINK1&keywords=MINK1>.

Conclusion

Our project featured various machine learning models to predict the severity of patients' Alzheimer's Disease with DNA methylation levels, and to predict genes that can be used as therapeutic targets for Alzheimer's Disease. We initially cleaned the data to only include the top 10 most differentially methylated regions, in order to ensure efficiency during our process. This later proved to be useful because some of our models, such as hierarchical clustering, proved to be time inefficient, and by trimming the data down, we were able to save time and remain efficient with our process. Throughout model building, we experimented with various methods to create models with high accuracy (without overfitting). Such methods included hyperparameter tuning, which finds the best parameter for the models to use to achieve the best accuracy. We also decided to make the models predict between control (no Alzheimer's Disease) and Alzheimer's cases rather than control, mild cognitive impairment, and Alzheimer's disease. We did so because during our process, we learned that classification models could not distinguish between mild cognitive impairment and Alzheimer's. Knowing the difficulties of distinguishing between the three states, we explored hierarchical clustering to see if it was possible, and it was proved that hierarchical clustering can distinguish between the three states with 86%.

The main findings of this project is 1) Machine Learning Classification can predict Alzheimer's/No Alzheimer's with 82% accuracy on average; 2) The highest scoring classification model was Gradient Boosting with 90% accuracy; 3) Hierarchical Clustering can predict Alzheimer's, Control (No Alzheimer's), and Mild Cognitive Impair with 86% accuracy; 4) RPL23, RPS13, VARS were predicted using the MCODE algorithm and are known to cause many brain disorders. **More importantly, the MINK1 gene contains the most DMRs upon Python analysis and is proven to cause Alzheimer's Disease, which makes it the best target for an Alzheimer's therapeutic.** In the future, we would like to make more machine learning models such as neural networks to see how that might affect model accuracy and maybe try to improve the model accuracy of hierarchical clustering. We would also like to look into larger datasets and DNA methylation profiles in more diseases. The main limitation to our project is with the model accuracies, since they are still slightly low, and we would like to try to improve them in the future. Another limitation is the data. Our dataset initially showed fairly low mutual information scores which doesn't show a strong correlation of DMRs with Alzheimer's disease,

even though it is a biologically known fact. This could be from substandard data which was downloaded, so in the future we would like to find other sources of data.

This project is important because it allowed us to explore the non-hereditary causes of Alzheimer's Disease such as epigenetics processes, which can be changed by environmental, lifestyle, and aging factors, and can play such a large role in causing diseases. It showed us that we have some control over the occurrences of diseases such as Alzheimer's, and how we have control over many aspects of our health and body. Additionally, our machine learning models can offer an alternative to current diagnostic methods such as MRI or CSF tests for a safer, cheaper method in predicting Alzheimers. Finally the genes (especially MINK1) we predicted in this project can be used as a pharmacogenomic (drug targeting of genes) therapy for Alzheimer's Disease.

Most Significant Achievement

In this project, we were able to use machine learning classification to predict Alzheimer's Disease with 90% accuracy. Additionally, we were able to successfully use hierarchical clustering to predict Mild Cognitive Impair and Alzheimer's disease with 86% accuracy. Finally, we predicted the MINK1 gene which is known to cause Alzheimer's Disease using Python analysis of DMRs, and we also predicted the genes RPL23, RPS13, VARS which could be used for a novel AD therapeutic.

Acknowledgements

We would like to thank Ms. Yolanda Lozano, our teacher, for supporting and guiding us through the project and our parents for their encouragement and support. We would also like to thank the Supercomputing Challenge Judges for giving us feedback during the midterm interview.

Bibliography

1. Centers for Disease Control and Prevention. (2020, August 3). *What is epigenetics?* Centers for Disease Control and Prevention. Retrieved April 5, 2022, from <https://www.cdc.gov/genomics/disease/epigenetics.htm>
2. Choudhury, A. (2020, December 24). *What is gradient boosting? how is it different from Ada Boost?* Medium. Retrieved April 6, 2022, from <https://medium.com/analytics-vidhya/what-is-gradient-boosting-how-is-it-different-from-ada-boost-2d5ff5767cb2>
3. *DNA methylation*. What is Epigenetics? (2019, September 5). Retrieved April 5, 2022, from <https://www.whatisepigenetics.com/dna-methylation/>
4. KNN classification. (n.d.). Retrieved April 6, 2022, from https://www.saedsayad.com/k_nearest_neighbors.htm
5. Latham, P. E., & Roudi, Y. (2009, January 21). *Mutual information*. Scholarpedia. Retrieved April 6, 2022, from http://www.scholarpedia.org/article/Mutual_information
6. Pal, D. S. K. (2019, July 15). *Epigenetic processes and human health*. Skyline University Nigeria. Retrieved April 6, 2022, from <https://www.sun.edu.ng/knowledge-update/epigenetic-processes-and-human-health>
7. Patro, R. (2021, February 1). *Cross validation: K Fold vs Monte Carlo*. Medium. Retrieved April 6, 2022, from <https://towardsdatascience.com/cross-validation-k-fold-vs-monte-carlo-e54df2fc179b>
8. U.S. National Library of Medicine. (n.d.). *Geo accession viewer*. National Center for Biotechnology Information. Retrieved April 6, 2022, from <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GPL13534>
9. U.S. National Library of Medicine. (n.d.). *Geo accession viewer*. National Center for Biotechnology Information. Retrieved April 6, 2022, from <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE144858>
10. U.S. National Library of Medicine. (n.d.). *Geo accession viewer*. National Center for Biotechnology Information. Retrieved April 6, 2022, from <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE144858>
11. Vassallo, D. (2019, August 6). *3 uses for Random decision trees / forests you (maybe) didn't know about*. David Vassallo's Blog. Retrieved April 6, 2022, from <http://blog.davidvassallo.me/2019/08/06/3-uses-for-random-decision-trees-forests-you-maybe-didnt-know-about/>
12. z_ai. (2021, September 26). *Logistic regression explained*. Medium. Retrieved April 6, 2022, from <https://towardsdatascience.com/logistic-regression-explained-9ee73cede081>

SUPERCOMPUTING CHALLENGE

ROBOTIC AIR QUALITY MONITOR:

SNOOPY

Team 12 members:

Joshua Mari Tamarra

Zachariah Burch

Isel Aragon

Britny Marquez

Teacher Sponsor:

Barbara Teterycz

Mentors:

David Ritter

Chris Karr

March, 27 2022

Table of Contents

Abstract/Executive Summary	3
Hypothesis	4
Identify the Problem	4
Problem Background and Research	4
Carbon Dioxide (CO ₂)	4
Total Volatile Organic Compounds (TVOC)	6
Pollen	7
The Size of Particles	7
Work Done By Others	9
Constraints	9
Goal	10
Brainstorming	10
Idea Generation / Selected Approach	10
Model/Prototype	11
Description of Model	11
Test model and evaluate	16
Troubleshooting, Testing & Redesigning	16
Using Data to Improve Air Quality	21
Computational/Mathematical Model	22
Conclusion	40
Collaboration	41
Roles / Responsibilities	41
Contributions	41
Works Cited	43

Abstract/Executive Summary

Due to current events, such as global warming causing the increase of carbon dioxide in the air as well as pandemic and the risk of catching deadly viruses, we were curious if there was a way to monitor air quality in their surrounding areas, such as school and homes. We strongly believed that knowing what's in the air could then lead to the best possible solutions and a much safer and healthier environment.

For this reason, we were meeting with a retired professional from Silicon Valley, and together we developed a robotic air quality monitoring system, called Snoopy. In order to build such a robot, we needed the following hardware: Arduino Mega microcontroller board, two stepper motors, two drive wheels, pivot wheel, BlueFruit drive for Bluetooth, Real Time Clock, SGP30 chemical sniffer/sensor, PMSA003 dust & pollen sensor, SD driver, SD card, battery holder with connector and plug, batteries, and wall-mounted power supply. After connecting all these parts together, we then worked with our mentor on programming them in an Arduino coding environment using C++ language.

Thanks to all these efforts, intense learning, and wonderful mentorship, each of us took our own Snoopy home in order to measure the quality of the air and to collect data, which we then analyzed and compared with each other as well as presented graphically using another programming language, Python. In order to better understand our data, we also did research about the particles in the air, and we learned that increased amounts of some of them may have very harmful effects on human's health.

As a result of this project, we found out that there is very good air quality at school, but not at our homes. Because of this, we propose the following solutions to improve the quality of air: open the window to allow CO₂ and TVOC escape from the house, replace carpet with tiles, use ecological paints, use air purifiers with HEPA filters, and invest in a good ventilation system.

Hypothesis

Due to current events, such as global warming causing the increase of atmospheric carbon dioxide as well as pandemic and its constantly increasing COVID cases, there is a big chance that the air quality isn't sufficient, and should be improved. In order to confirm this hypothesis, a programmable air quality monitoring system should be used to identify the circumstances that contribute to either worsening or improving the quality of air. We predict that simply opening windows and doors as well as investing in a good air purification and ventilation systems should contribute to decreasing the level of carbon dioxide and other toxic gasses in the air, but at the same time these remedies (especially opening the window) might increase the level of pollen.

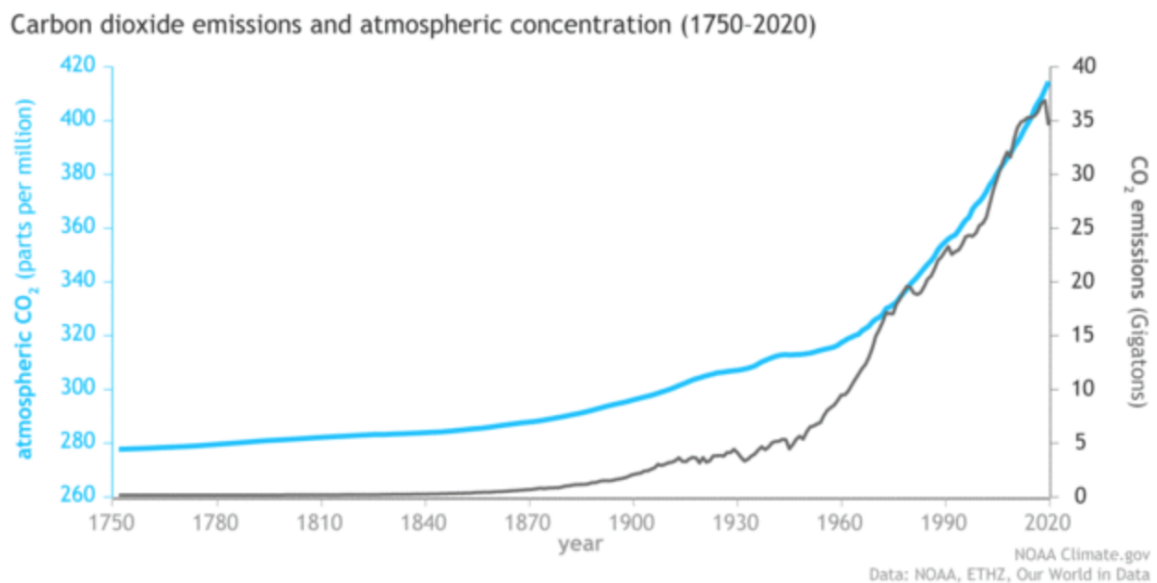
Identify the Problem

Problem Background and Research

Carbon Dioxide (CO₂)

Carbon dioxide is made of one carbon atom and two oxygen atoms, and is a naturally occurring gas in the atmosphere, without which plants would not be able to survive, and our planet would be too cold after the sunset. Carbon dioxide, as a greenhouse gas, traps heat in the atmosphere keeping our planet warm (Smith, 2019). However, due to anthropogenic CO₂ emissions, caused by human activities, such as burning fossil fuels for energy, recent outdoor carbon dioxide levels are the highest they've ever been.

According to the article published at climate.gov, “fossil fuels like coal and oil contain carbon that plants pulled out of the atmosphere through photosynthesis over many millions of years; and we are returning that carbon to the atmosphere in just a few hundred years” (Lindsey, 2020). As shown on the chart below, since the start of the Industrial Revolution in 1750, the amount of carbon dioxide in the atmosphere (blue line) has increased along with human emissions (gray line) and currently it has already exceeded 400 ppm (particles per million parts of air). That is why the lowest possible level of carbon dioxide that has been detected by Snoopy never dropped below 400 ppm.



In addition to atmospheric carbon dioxide, there is also indoor CO₂, which is produced by our own bodies as a byproduct of respiration when we exhale as well as during cooking, burning candles, incense, wood in fireplace, and smoking. And if there is poor ventilation in our houses that are tightly sealed to save energy, CO₂ is trapped and builds up to unhealthy levels. At around 1000 ppm, the residents may start to experience fatigue, sleepiness, and problems with concentration. With prolonged exposure to higher concentration of CO₂, people will experience additional and more severe health issues, such as headache, drowsiness, tiredness, dizziness, sweating, difficulty breathing, and even seizures and loss of consciousness when the level of

indoor CO₂ is very high. The table below shows the examples of health effects caused by the specific levels of CO₂ concentration (Smith, 2019).

CO₂ Concentration	Health Effects
< 1000 ppm	Limited or no health effects
1000 ppm - 2500 ppm	Fatigue, loss of focus and concentration, uncomfortable 'stuffy' feeling in the air
2500 ppm - 5000 ppm	Headache, drowsiness, tiredness
5000 ppm - 40000 ppm	Violates OSHA requirements, severe headaches, slight intoxication depending on the exposure time
40000 ppm - 100000 ppm	IDLH (Immediately dangerous to life or health), dizziness, increased heart rate, sweating, difficulty breathing; seizures and loss of consciousness after prolonged exposure
> 100000 ppm	Loss of consciousness within minutes, coma, risk of death

Total Volatile Organic Compounds (TVOC)

Total Volatile Organic Compounds are multiple organic chemicals that become a gas at room temperature. Many VOCs come from cleaners and disinfectants, air fresheners, fragrances, paints and solvents, glue, plywood, candles and fires, cooking fumes, new furniture and carpets, electronic devices, etc. TVOC can be measured in micrograms per cubic meter ($\mu\text{g}/\text{m}^3$) of air (or milligrams per cubic meter (mg/m^3), parts per million (ppm) or parts per billion (ppb)). Past recommendations from the USEPA (United States Environmental Protection Agency, 2021) and indoor environment rating schemes is that a recommended limit of 500 ppb TVOC and less than 250 ppb of any one VOC is appropriate in average office environments (The World Green Building Council, 2020). The table below shows the recommended actions that should be taken for a given ppb level of TVOC:

0 - 250 ppb	The VOC contents in the air are low.
250 - 2000 ppb	Look for VOC sources if this average level persists for a month.
> 2000 ppb	The VOC contents are very high - consider taking action/ventilating right now.

Some VOCs are bad for health, especially during long-term exposure in large doses. Immediate symptoms that some people have experienced soon after exposure to VOCs are eye and respiratory tract irritation, headaches, dizziness, visual disorders and memory impairment. Some VOCs as formaldehyde (used in making building materials) can cause cancer (Advanced Solutions Nederland B.V., 2020).

Pollen

Some plants, including various kinds of trees, grasses, and weeds, make a fine powder called pollen that's light enough to travel through the air in order to reproduce. More than 25 million Americans are allergic to pollen, which means that their white blood cells release a chemical, called histamine, when their immune system is defending against allergen. This can result in allergic reactions, such as itchy throat; red, itchy, watery eyes; runny or stuffy nose; sneezing; wheezing or coughing (Fields and DerSarkissian, 2021).

In addition to getting medical help, many people try to avoid going out and stay indoors instead. However, as our Snoopy found out, keeping the window always closed, traps CO₂ and TVOC inside and makes the indoor air even more dangerous and toxic.

The Size of Particles

Due to the microscopic size of the particles in the air, in order for Snoopy to detect and correctly classify them, it was necessary to install two kinds of sensors, one for CO₂ and TVOC

and the other one for pollen and dust.



As shown here, pollen, salt, and sand are significantly larger than viruses or bacteria. Because of their higher relative sizes, our body is usually able to block them out—a particle needs to be smaller than 10 microns before it can be inhaled into our respiratory tract. Because of this, pollen or sand typically get trapped in the eyes, nose, and throat, before they enter our lungs. The smaller particles (e.g. viruses or wildfire smoke) however, are able to slip through more easily and cause even more dangerous health risks.

While allergies to pollen worsen the quality of life and are very inconvenient (sometimes even making it difficult to see), the smaller particles, like viruses, are even more dangerous for human health. However, air pollution caused by particulate matter (such as dust, dirt, soot, and smoke particles) has even more chances to enter human lungs. For example, wildfire smoke at just a fraction of the size between 0.4-0.7 microns, has been identified as the key factor in not just respiratory issues, but also cardiovascular and neurological problems (Ang et al., 2020).

Because of all these harmful air particles, it's very important especially now, during the time of pandemic and global warming resulting in frequent wildfires, to constantly care for the

best air quality possible in both public and private places.

Work Done By Others

Since building the robot and programming all its components required thorough knowledge of C++ language and even some electrical engineering, such as designing electrical circuits and connectors, we were not able to build it on our own. That is why we were working with a professional mentor with almost 40 years of experience working at Silicon Valley as an electrical engineer and also C/C++ programmer.

After collecting and analyzing data, we then worked with another mentor, a professional computer programmer, who helped us apply linear regression in a computational model for the purpose of further statistical analysis.

Constraints

Because of the risk with COVID, we were working with our engineering mentor remotely, which wasn't always easy, especially when we had to connect the physical parts together. In addition, there were many electrical concepts, which we were not familiar with, and which were hard for us to comprehend without professional experience.

Another limitation, which we had, was caused by the COVID-related guidelines, which required us to always have air purifiers on and windows open during school hours. Because of this, we were unable to measure how air quality was affected by the presence of students in class when the air purifier was turned off and windows were closed.

Goal

The goal of this project is to have a correctly working air quality monitoring system that can detect and record increased levels of particulates, such as pollutants and allergens in the air. This would let us know how safe/unsafe the environment, in which we live, work, study, and sleep, is. For example, if there is a low level of CO₂, it means that there is very good ventilation, which is especially important and beneficial now, during the pandemic. Knowing what's in the air might also help people suffering from headaches, dizziness, sleepiness, problems with concentration and allergies identify the source(s) of their health issues.

Brainstorming

Idea Generation / Selected Approach

As high school students with no prerequisites in computer science, we first wanted to build a robot. We then realized that for this kind of a challenge, our robot should be able to do something practical and useful rather than just only move around. Since our teacher-sponsor was suffering from allergies to pollen, we came up with the idea to have our robot detect it. Our mentor also advised us to add another sensor that would detect the level of carbon dioxide and other chemicals, such as volatile organic compounds. This way, our robot would measure the overall air quality, not just pollen. Thanks to all these ideas, we strongly believed that knowing what's in the air could lead to a much safer and healthier environment.

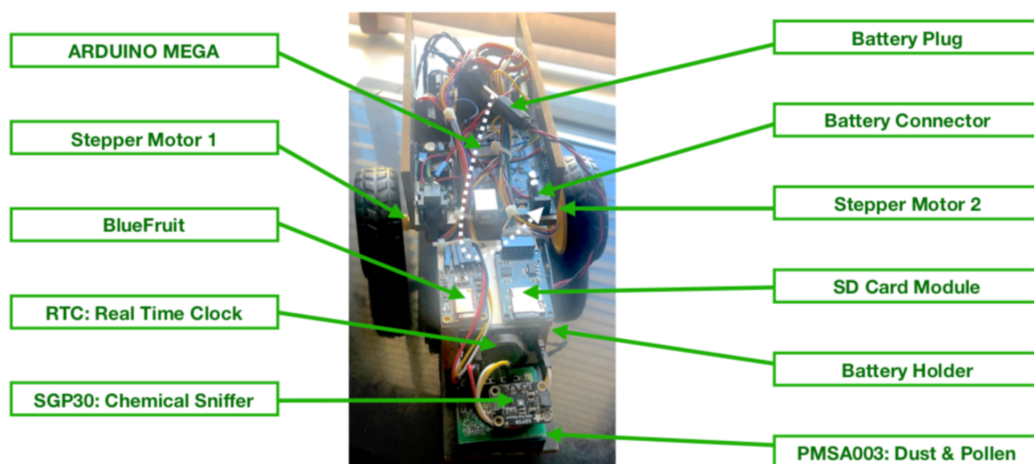
Model/Prototype

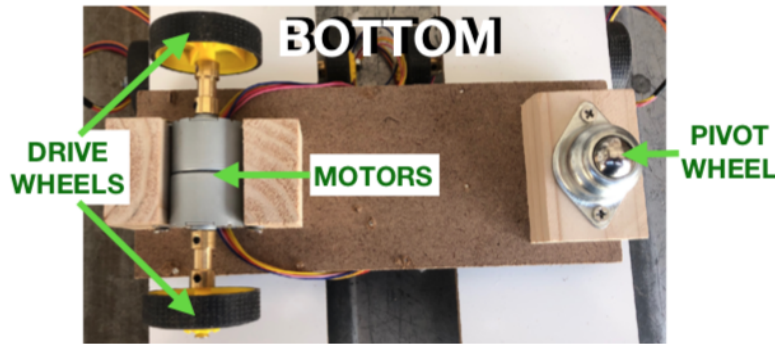
Description of Model

While working with our mentor from Silicon Valley, we have developed a robotic air quality monitoring system, called Snoopy:



In order to build such a robot, we needed the following hardware:





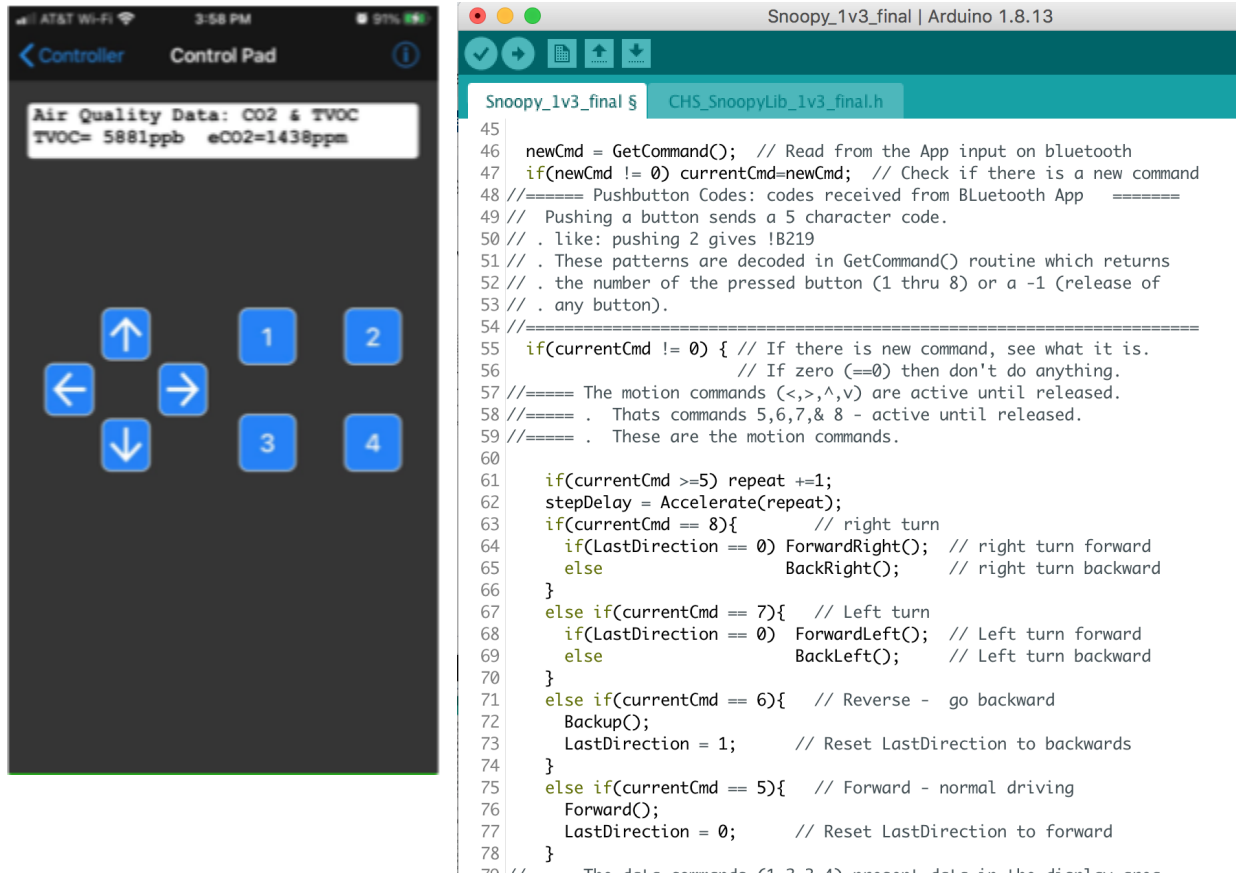
After connecting all the parts of Snoopy together, some of us worked in create.arduino.cc using our school chromebooks, and some others installed the Arduino coding environment on their computers. We were participating in many lectures about using C++ language to program all the parts of Snoopy. Many of the concepts were too advanced for us, but in general, what we've learned was that in order to complete the program, almost each part of Snoopy required a special library that had to be imported at the top of the program using #include command with the name of the library and .h extension enclosed by the less and greater than symbols. Next, we've learned that all the variables had to be defined together with their kind and data type just under the libraries and before the main part of the program, like this:

```

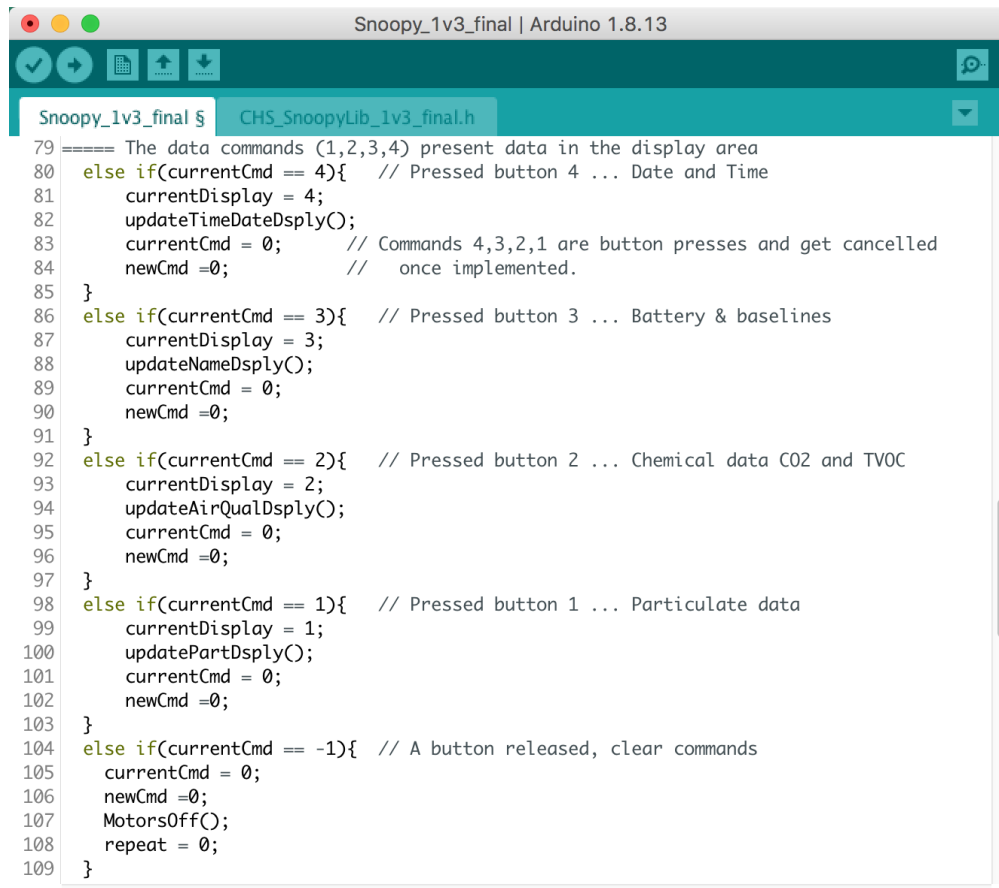
Snoopy_1v3_final | Arduino 1.8.13
Snoopy_1v3_final $ CHS_SnoopyLib_1v3_final.h
1 #include <Adafruit_ATParser.h>
2 #include <Adafruit_BluefruitLE_SPI.h>
3 #include <Adafruit_BLEMIDI.h>
4 #include <Adafruit_BLEBattery.h>
5 #include <Adafruit_BLEGatt.h>
6 #include <Adafruit_BLEEddystone.h>
7 #include <Adafruit_BLE.h>
8 #include <Adafruit_BluefruitLE_UART.h>
9
10 #define mySnoopyName "Snoopy.name_your_Snoopy"
11 #define changeName
12 #define myFileName "DataLog.txt"
13
14 const float batLowThresh =7.0;
15 const int Verbose = 0;
16 const int dontWait4connect = 0;
17 const int EESaveSet = 12;
18 int EESaveCount = 0;
19 int seconds, oldsec,edgesec;
20 int minutes, oldmin;
21 const int DisplayTime = 5;
22 int currentDisplay = 2;
23
24 #include "CHS_SnoopyLib_1v3_final.h" // Library for general Arduino Mega stuff.
25
26 int newCmd=0, currentCmd=0;
27 int LastDirection=0;
--

```

We also learned how to connect Snoopy to the Bluefruit app installed on our cell phones via Bluetooth, and how to control its movement when each of the following arrow buttons was pressed:

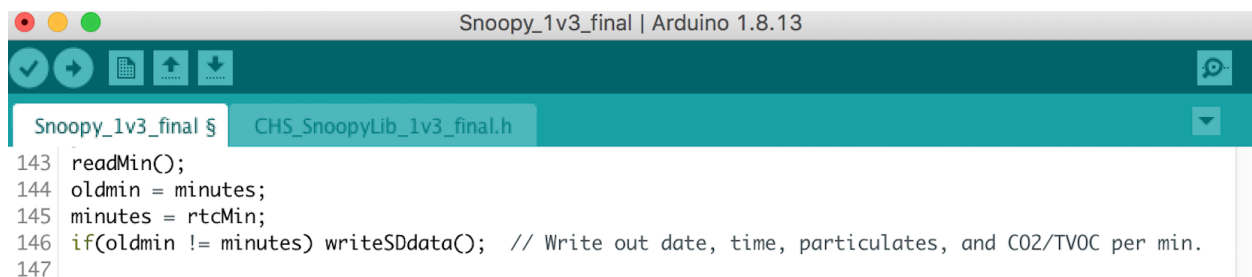


In addition, we've learned how to program each of the four buttons to display data on the screen of the Bluefruit's app, such as button 1, which when pressed, displays the level of pollen and dust in the air; button 2 displays the level of chemicals, such as CO2 and TVOC, button 3 displays the level of battery, and button 4 displays the date and time, as shown on the screenshot below:



```
Snoopy_1v3_final $ CHS_SnoopyLib_1v3_final.h
79 ===== The data commands (1,2,3,4) present data in the display area
80 else if(currentCmd == 4){ // Pressed button 4 ... Date and Time
81     currentDisplay = 4;
82     updateTimeDateDsply();
83     currentCmd = 0; // Commands 4,3,2,1 are button presses and get cancelled
84     newCmd = 0; // once implemented.
85 }
86 else if(currentCmd == 3){ // Pressed button 3 ... Battery & baselines
87     currentDisplay = 3;
88     updateNameDsply();
89     currentCmd = 0;
90     newCmd = 0;
91 }
92 else if(currentCmd == 2){ // Pressed button 2 ... Chemical data CO2 and TVOC
93     currentDisplay = 2;
94     updateAirQualDsply();
95     currentCmd = 0;
96     newCmd = 0;
97 }
98 else if(currentCmd == 1){ // Pressed button 1 ... Particulate data
99     currentDisplay = 1;
100    updatePartDsply();
101    currentCmd = 0;
102    newCmd = 0;
103 }
104 else if(currentCmd == -1){ // A button released, clear commands
105     currentCmd = 0;
106     newCmd = 0;
107     MotorsOff();
108     repeat = 0;
109 }
```

And finally, we've also learned how to write data gathered by Snoopy every minute to the SD card for the future analysis and graphical representation:

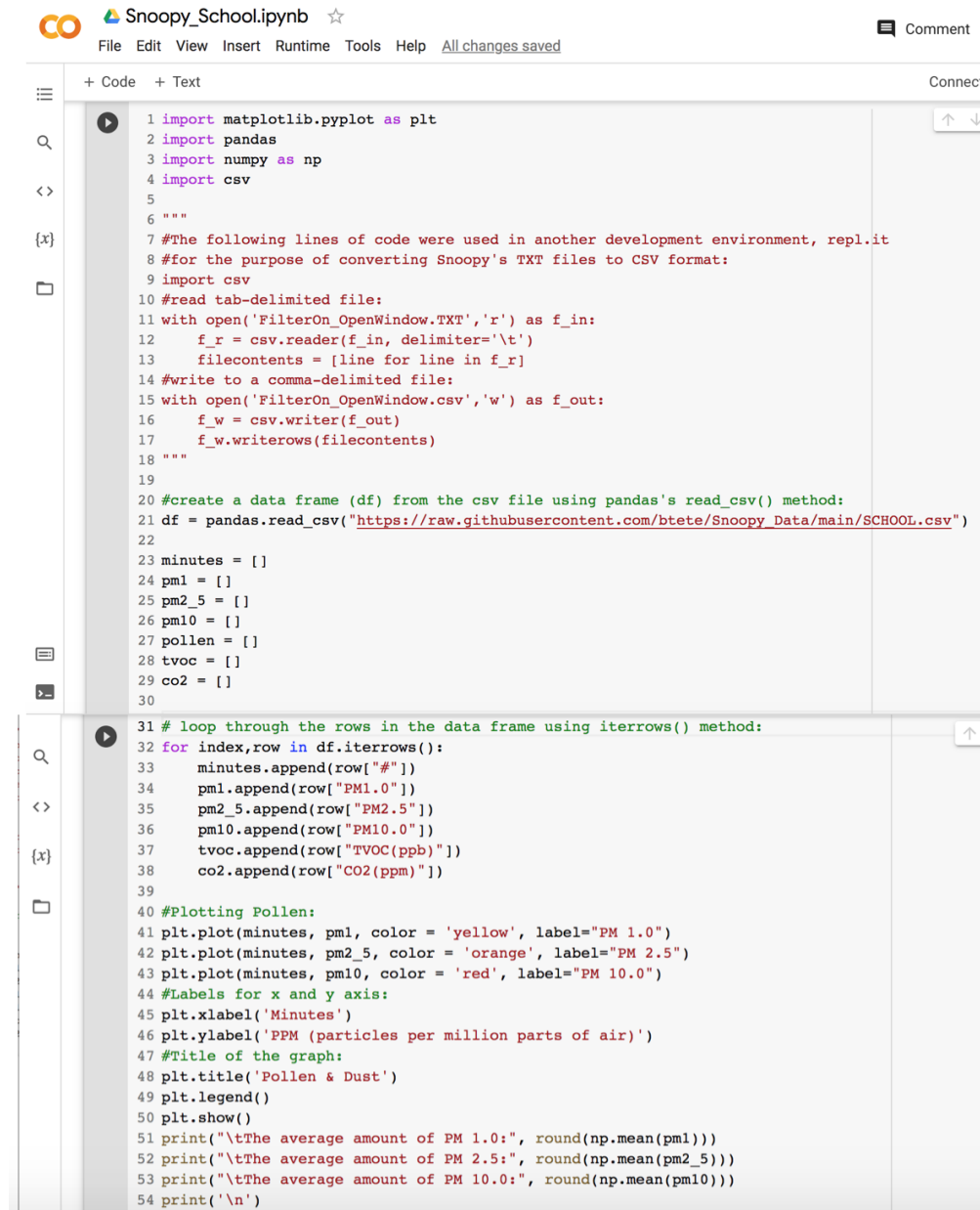


```
Snoopy_1v3_final $ CHS_SnoopyLib_1v3_final.h
143 readMin();
144 oldmin = minutes;
145 minutes = rtcMin;
146 if(oldmin != minutes) writeSDdata(); // Write out date, time, particulates, and CO2/TVOC per min.
147
```

In this part of the program, the time was recorded by rtc (real time clock) and constantly compared to its previous record. When there was a difference in time, data detected by the sensors were then written to the SD card.

Thanks to all these efforts, intense learning, and wonderful support, which we've received from our mentor, all of us took one copy of Snoopy home in order to measure the

quality of their air and to collect data, which we then analyzed and compared with each other as well as presented graphically using another programming language, Python:



```
1 import matplotlib.pyplot as plt
2 import pandas
3 import numpy as np
4 import csv
5
6 """
7 #The following lines of code were used in another development environment, repl.it
8 #for the purpose of converting Snoopy's TXT files to CSV format:
9 import csv
10 #read tab-delimited file:
11 with open('FilterOn_OpenWindow.TXT','r') as f_in:
12     f_r = csv.reader(f_in, delimiter='\t')
13     filecontents = [line for line in f_r]
14 #write to a comma-delimited file:
15 with open('FilterOn_OpenWindow.csv','w') as f_out:
16     f_w = csv.writer(f_out)
17     f_w.writerows(filecontents)
18 """
19
20 #create a data frame (df) from the csv file using pandas's read_csv() method:
21 df = pandas.read_csv("https://raw.githubusercontent.com/btete/Snoopy_Data/main/SCHOOL.csv")
22
23 minutes = []
24 pm1 = []
25 pm2_5 = []
26 pm10 = []
27 pollen = []
28 tvoc = []
29 co2 = []
30
31 # loop through the rows in the data frame using iterrows() method:
32 for index,row in df.iterrows():
33     minutes.append(row["#"])
34     pm1.append(row["PM1.0"])
35     pm2_5.append(row["PM2.5"])
36     pm10.append(row["PM10.0"])
37     tvoc.append(row["TVOC(ppb)"])
38     co2.append(row["CO2(ppm)"])
39
40 #Plotting Pollen:
41 plt.plot(minutes, pm1, color = 'yellow', label="PM 1.0")
42 plt.plot(minutes, pm2_5, color = 'orange', label="PM 2.5")
43 plt.plot(minutes, pm10, color = 'red', label="PM 10.0")
44 #Labels for x and y axis:
45 plt.xlabel('Minutes')
46 plt.ylabel('PPM (particles per million parts of air)')
47 #Title of the graph:
48 plt.title('Pollen & Dust')
49 plt.legend()
50 plt.show()
51 print("\tThe average amount of PM 1.0:", round(np.mean(pm1)))
52 print("\tThe average amount of PM 2.5:", round(np.mean(pm2_5)))
53 print("\tThe average amount of PM 10.0:", round(np.mean(pm10)))
54 print('\n')
```

As a result of this collaborative work, we are able to measure, collect, analyze, present, research and explain the sources of air pollutants, and even propose some solutions to improve the quality of air.

Test model and evaluate

Troubleshooting, Testing & Redesigning

Snoopy let us investigate the situation in both school and home. However, what we've noticed after collecting data was that Snoopy almost always started with indicating the lowest level of CO₂. We then learned that it was because it was the default base level, and that it usually took a while until Snoopy started sniffing real values. We've also noticed sometimes some sudden outliers in data, which could be caused by either sudden air movement or some kind of recalibration happening inside Snoopy.

In addition, once we gathered our data, we found out that it was hard to balance the level of CO₂ and TVOC with the level of pollen and dust in the indoor air. When we were able to lower CO₂ and TVOC by simply opening the window, then the level of pollen went drastically up. When we closed the window, we got the opposite effect: the level of CO₂ and TVOC went up, whereas the level of pollen and dust went down. It forced us to look for some kind of a compromise between the amounts of these two major air pollutants and allergens.

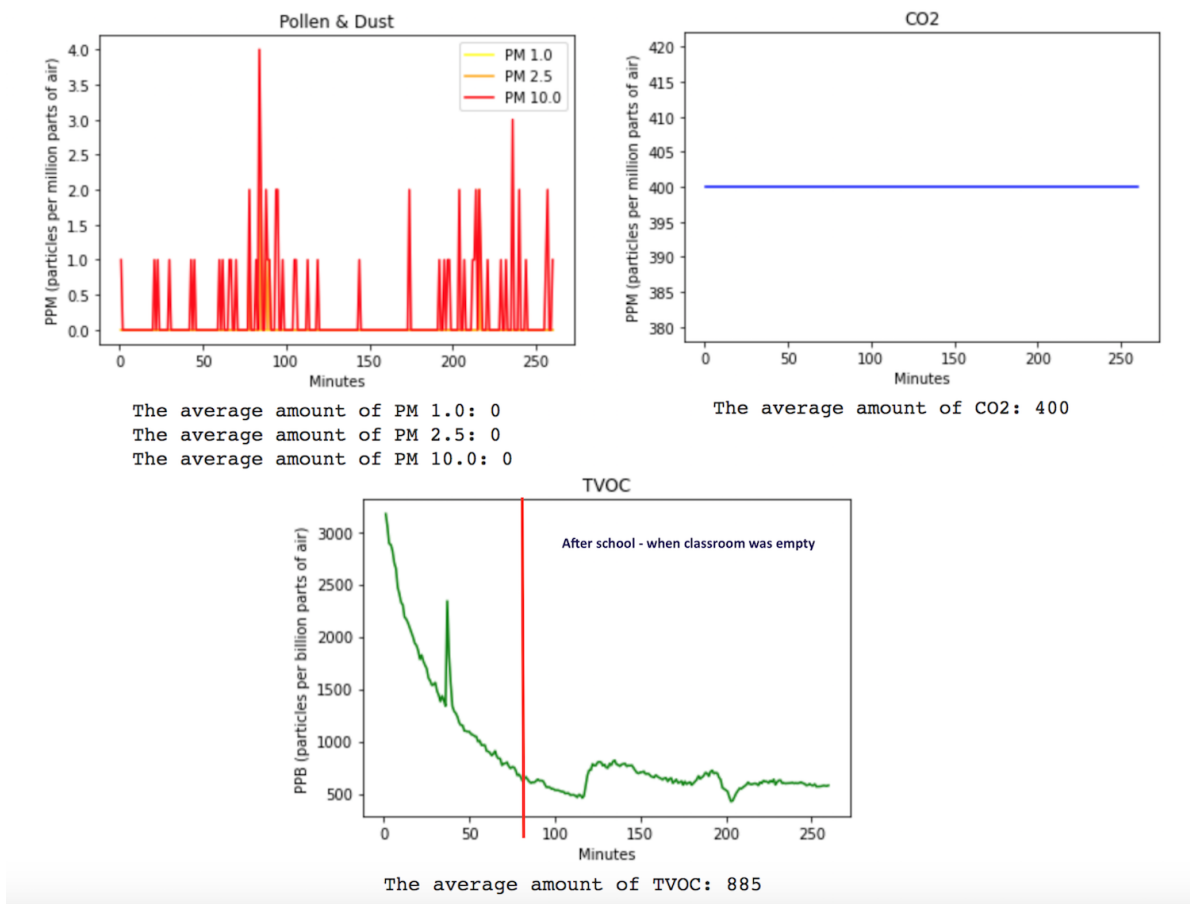
Another difficulty, which we faced while gathering data, was their inconsistency caused by different seasons of the year (e.g. there was more pollen during Fall than during Winter time). We also noticed a discrepancy between daily data collected at school that might depend on whether or not the students wore perfumes and/or deodorants, used laundry detergents and softeners with fragrances, hair spray, etc.

And finally, due to COVID-related guidelines, we were only able to collect data when students were present in class while the air purifier was on and windows were open. Thus, our

data lack information about the influence of people on air quality when the air purifier was off and windows were closed.

Because of the above issues, we decided to start with troubleshooting the performance of Snoopy by simply excluding the outliers and the first default, but not yet accurate, values from our data. According to the following data, which Snoopy gathered for us at school, both the level of pollen and dust as well as the level of CO₂ were very low. However, the level of TVOC was first increased, especially when students were in class, even though the window was open and air purifier was turned on:

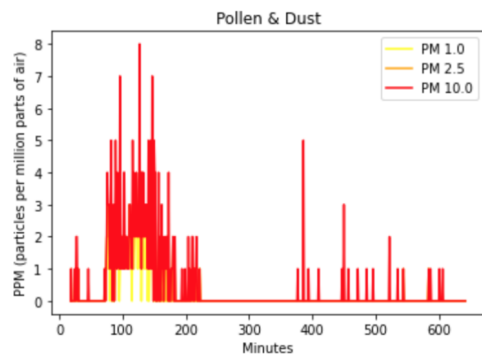
#	MM:DD:YYYY	HH:MM:SS	PM1.0	PM2.5	PM10.0	TVOC(ppb)	CO2(ppm)
1	11/10/2021	2:38: 0 PM	0	0	1	3173	400
2	11/10/2021	2:39: 0 PM	0	0	0	3052	400
3	11/10/2021	2:40: 0 PM	0	0	0	2888	400
4	11/10/2021	2:41: 0 PM	0	0	0	2879	400
5	11/10/2021	2:42: 0 PM	0	0	0	2819	400
6	11/10/2021	2:43: 0 PM	0	0	0	2704	400
7	11/10/2021	2:44: 0 PM	0	0	0	2651	400
8	11/10/2021	2:45: 0 PM	0	0	0	2471	400
9	11/10/2021	2:46: 0 PM	0	0	0	2408	400
10	11/10/2021	2:47: 0 PM	0	0	0	2327	400
11	11/10/2021	2:48: 0 PM	0	0	0	2304	400
12	11/10/2021	2:49: 0 PM	0	0	0	2187	400
13	11/10/2021	2:50: 0 PM	0	0	0	2168	400
14	11/10/2021	2:51: 0 PM	0	0	0	2133	400
15	11/10/2021	2:52: 0 PM	0	0	0	2087	400



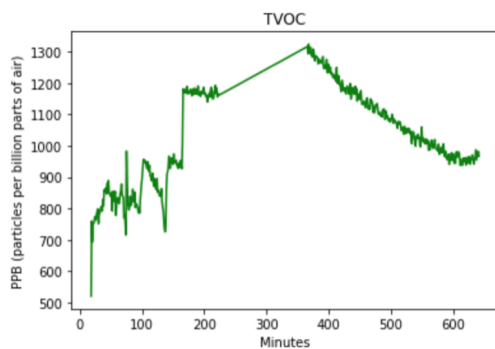
We suspect that this could be caused by the usage of deodorants, perfumes, hair sprays, clothes dried with sheets coated in lubricants and fragrances, etc. As shown on the graphs above, when students left the school and the classroom was empty, the level of both TVOC and CO2 remained at a lower level, even though the window was closed. It proves that our school has a very good ventilation system and was prepared for the pandemic very well.

We then used Snoopy at our houses. As we found out, the situation in all our homes was much worse. We suspect that the reason could be a much smaller area of our rooms than the area of our classrooms, the presence of kitchen, pets, and carpet in our houses, worse or no ventilation system, the lack of air purifiers (except for one of us), and close presence of neighbors and cars, as opposed to our classroom, which is far from the parking lot. The following results represent the level of pollen and dust, as well as CO2 and TVOC when the window was closed:

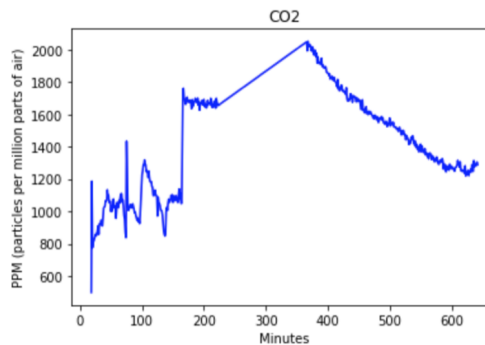
#	MM:DD:YYYY	HH:MM:SS	PM1.0	PM2.5	PM10.0	TVOC(ppb)	CO2(ppm)
18	9/11/2021	9:33: 0 PM	0	0	0	520	498
19	9/11/2021	9:34: 0 PM	0	1	1	760	1186
20	9/11/2021	9:35: 0 PM	0	0	0	693	775
21	9/11/2021	9:36: 0 PM	0	0	0	752	815
22	9/11/2021	9:37: 0 PM	0	0	0	751	815
23	9/11/2021	9:38: 0 PM	0	0	0	757	829
24	9/11/2021	9:39: 0 PM	0	0	0	774	844
25	9/11/2021	9:40: 0 PM	0	0	1	764	857
26	9/11/2021	9:41: 0 PM	0	0	0	764	848
27	9/11/2021	9:42: 0 PM	0	0	2	784	861
28	9/11/2021	9:43: 0 PM	0	0	2	793	881
29	9/11/2021	9:44: 0 PM	0	0	0	798	909
30	9/11/2021	9:45: 0 PM	1	1	1	751	857
31	9/11/2021	9:46: 0 PM	0	0	0	792	922
32	9/11/2021	9:47: 0 PM	0	0	0	785	924
33	9/11/2021	9:48: 0 PM	0	0	0	797	919
34	9/11/2021	9:49: 0 PM	0	0	0	804	945
35	9/11/2021	9:50: 0 PM	0	0	0	806	913
36	9/11/2021	9:51: 0 PM	0	0	0	792	910
37	9/11/2021	9:52: 0 PM	0	0	0	829	969
38	9/11/2021	9:53: 0 PM	0	0	0	809	1020
39	9/11/2021	9:54: 0 PM	0	0	0	859	1027



The average amount of PM 1.0: 0
The average amount of PM 2.5: 0
The average amount of PM 10.0: 1



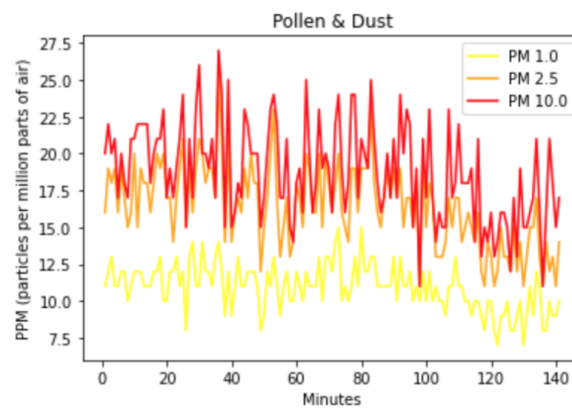
The average amount of TVOC: 1029



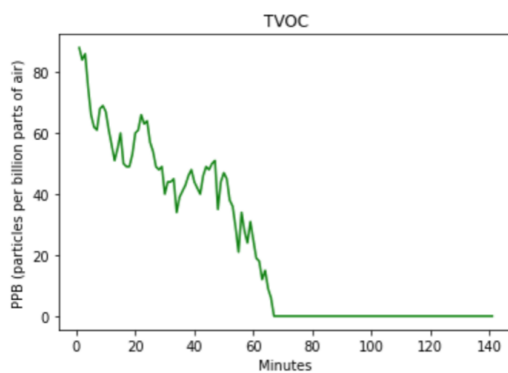
The average amount of CO2: 1407

After opening the window, both TVOC and CO2 decreased significantly, but the level of pollen and dust went drastically up:

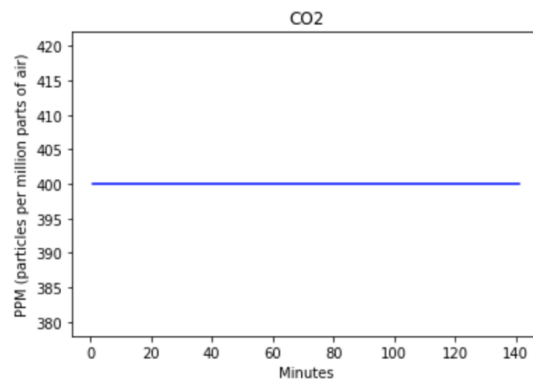
#	MM:DD:YYYY	HH:MM:SS	PM1.0	PM2.5	PM10.0	TVOC(ppb)	CO2(ppm)
1	9/12/2021	12:29: 0 AM	11	16	20	88	400
2	9/12/2021	12:30: 0 AM	12	19	22	84	400
3	9/12/2021	12:31: 0 AM	13	18	20	86	400
4	9/12/2021	12:32: 0 AM	11	19	21	75	400
5	9/12/2021	12:33: 0 AM	11	16	17	66	400
6	9/12/2021	12:34: 0 AM	12	19	20	62	400
7	9/12/2021	12:35: 0 AM	12	17	18	61	400
8	9/12/2021	12:36: 0 AM	10	15	17	68	400
9	9/12/2021	12:37: 0 AM	11	16	21	69	400
10	9/12/2021	12:38: 0 AM	12	20	21	67	400
11	9/12/2021	12:39: 0 AM	12	15	22	61	400
12	9/12/2021	12:40: 0 AM	12	19	22	56	400
13	9/12/2021	12:41: 0 AM	11	18	22	51	400
14	9/12/2021	12:42: 0 AM	11	18	22	55	400



The average amount of PM 1.0: 11
The average amount of PM 2.5: 16
The average amount of PM 10.0: 19



The average amount of TVOC: 22



The average amount of CO2: 400

Using Data To Improve Air Quality

Based on data collected by Snoopy, we realized that even without investing in an expensive ventilation system, we can still improve the quality of air by simply opening or closing windows and doors, and turning air purifiers on. The following data collected by Snoopy when the window was open, show how the level of pollen and TVOC changed when the air purifier with HEPA filter was first turned off and then on (with the open window, the level of CO2 was always at its lowest, regardless of whether the air purifier was turned on or off):

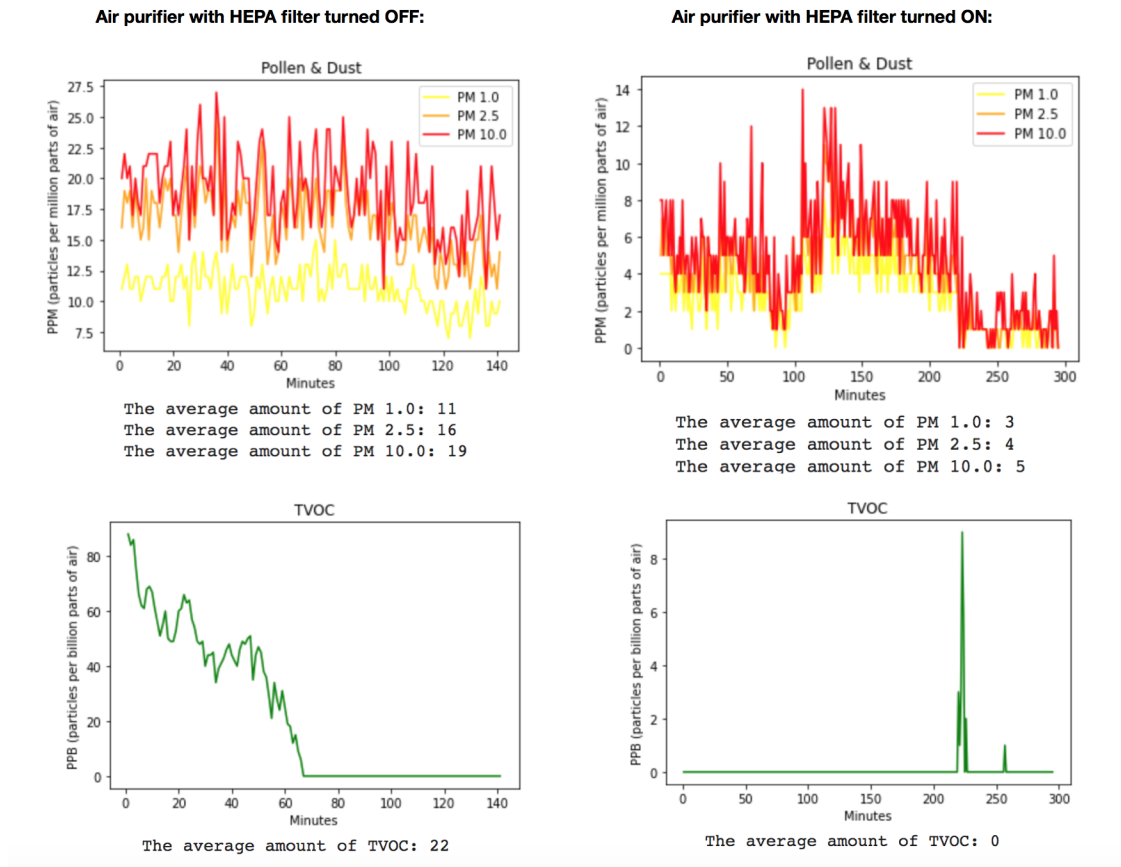
Filter Off:

#	MM:DD:YYYY	HH:MM:SS	PM1.0	PM2.5	PM10.0	TVOC(ppb)	CO2(ppm)
1	9/12/2021	12:29: 0 AM	11	16	20	88	400
2	9/12/2021	12:30: 0 AM	12	19	22	84	400
3	9/12/2021	12:31: 0 AM	13	18	20	86	400
4	9/12/2021	12:32: 0 AM	11	19	21	75	400
5	9/12/2021	12:33: 0 AM	11	16	17	66	400
6	9/12/2021	12:34: 0 AM	12	19	20	62	400
7	9/12/2021	12:35: 0 AM	12	17	18	61	400
8	9/12/2021	12:36: 0 AM	10	15	17	68	400
9	9/12/2021	12:37: 0 AM	11	16	21	69	400
10	9/12/2021	12:38: 0 AM	12	20	21	67	400
11	9/12/2021	12:39: 0 AM	12	15	22	61	400
12	9/12/2021	12:40: 0 AM	12	19	22	56	400
13	9/12/2021	12:41: 0 AM	11	18	22	51	400
14	9/12/2021	12:42: 0 AM	11	18	22	55	400

Filter On:

#	MM:DD:YYYY	HH:MM:SS	PM1.0	PM2.5	PM10.0	TVOC(ppb)	CO2(ppm)
1	9/12/2021	3:37: 0 PM	4	5	8	0	400
2	9/12/2021	3:38: 0 PM	4	7	8	0	400
3	9/12/2021	3:39: 0 PM	4	5	5	0	400
4	9/12/2021	3:40: 0 PM	4	7	7	0	400
5	9/12/2021	3:41: 0 PM	4	5	8	0	400
6	9/12/2021	3:42: 0 PM	4	5	6	0	400
7	9/12/2021	3:43: 0 PM	4	5	5	0	400
8	9/12/2021	3:44: 0 PM	4	5	8	0	400
9	9/12/2021	3:45: 0 PM	2	4	5	0	400
10	9/12/2021	3:46: 0 PM	4	5	8	0	400
11	9/12/2021	3:47: 0 PM	3	4	4	0	400
12	9/12/2021	3:48: 0 PM	2	3	3	0	400
13	9/12/2021	3:49: 0 PM	4	5	5	0	400
14	9/12/2021	3:50: 0 PM	3	4	4	0	400
15	9/12/2021	3:51: 0 PM	2	5	6	0	400

As shown on the charts below, the average level of pollen decreased about four times and the average level of TVOC dropped to 0 after the air purifier was turned on, which proves the effectiveness of HEPA filters.



Computational/Mathematical Model

In order to better understand and analyze the correlation between all the conditions, during which our data were collected, as well as to identify the most influential ones that either significantly decreased or increased the quality of air, we decided to make further statistical analysis on our data in the form of a computational model. After being introduced by our other (coding) mentor to linear regression, we've learned that it measures the relationship between the

input variables (x), also known as independent variables, and the single output variable (y), also known as dependent variable (Brownlee, 2020). In order to better see how each condition affected the quality of air, we decided to use this mathematical concept in our computational model. We learned that a variety of techniques can be used to prepare the linear regression equation from data, and that the most common one is Ordinary Least Squares Linear Regression (Brownlee, 2020). Its purpose is to find the values of coefficients b (y-intercept) and m (slope of the line) used in the linear function together with independent variables x and dependent variable y, for which the sum of the squared distances (also called errors) between the actual data points and the line of such function is the least. And finally, in order to assess how well a regression model fits a dataset, it is necessary to also calculate the root mean square error (RMSE), which indicates the average distance between the predicted values from the model and the actual data points. The lower the RMSE, the better a model fits a dataset (Zach, 2021).

Since two of the students in our team have not taken calculus class yet, we first decided to study the mathematical concepts and equations used in this kind of linear regression model:

Linear function: $y = b + mx$

Where:

$y \rightarrow$ dependent variable

$b \rightarrow$ y-intercept

$m \rightarrow$ slope of the line

$x \rightarrow$ independent variable

For the purpose of our model that uses many data points, we replaced the above formula with the following:

$$q_1 = \beta_0 + \beta_1 p_1 + E_1$$

...

$$q_n = \beta_0 + \beta_1 p_n + E_n$$

Where:

$q_1, q_2, \dots, q_n \rightarrow$ actual data points

$\beta_0, \beta_1 \rightarrow$ constants (coefficients), think about them as b and m

$p_1, p_2, \dots, p_n \rightarrow$ predicted points (points on the line)

$E_1, E_2, \dots, E_n \rightarrow$ errors (distances from the line)

The above can be represented in this simplified form:

$$\vec{q} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \begin{bmatrix} 1 & p_1 \\ \cdot & \cdot \\ \cdot & \cdot \\ 1 & p_n \end{bmatrix} + \vec{E}$$

1) Since we are looking for the least errors, let's solve for E:

$$\vec{E} = \vec{q} - \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \begin{bmatrix} 1 & p_1 \\ \cdot & \cdot \\ \cdot & \cdot \\ 1 & p_n \end{bmatrix}$$

2) Let's calculate the norm ($\|\dots\|$) of E (a norm is a square root of the sum of the squared elements):

$$\|\vec{E}\| = \sqrt{E_1^2 + \dots + E_n^2} = \sqrt{(q_1 - (\beta_0 + \beta_1 p_1))^2 + \dots + (q_n - (\beta_0 + \beta_1 p_n))^2}$$

3) Let's square the norm to get rid of square root:

$$\|\vec{E}\|^2 = \sqrt{E_1^2 + \dots + E_n^2}^2 = \sqrt{(q_1 - \beta_0 - \beta_1 p_1)^2 + \dots + (q_n - \beta_0 - \beta_1 p_n)^2}^2$$

$$\vec{E} = (q_1 - \beta_0 - \beta_1 p_1)^2 + \dots + (q_n - \beta_0 - \beta_1 p_n)^2$$

$$\vec{E} = \sum_i (q_i - \beta_0 - \beta_1 p_i)^2$$

We will now learn how to take partial derivative (∂) in respect to x (∂_y / ∂_x) in the following

example: $y = ax^3 + bx^2 + x + 5$. In order to do so, use x's exponent as a constant value and

multiply it by x, and then decrease this x's exponent by 1:

$\partial_y / \partial_x = 3ax^{3-1} + 2bx^{2-1} + 1x^{1-1} + 0*5x^{0-1} = 3ax^2 + 2bx + 1 + 0$; Now, back to our problem:

4) Take partial derivative in respect to β_0 :

$$y = (q_i - \beta_0 - \beta_1 p_i)^2 = u^2$$

$$u = q_i - \beta_0 - \beta_1 p_i$$

Let's apply the chain rule as shown in this video: https://www.youtube.com/watch?v=Ic_LW7K8eGE

$$\frac{\partial y}{\partial \beta_0} = \frac{\partial u}{\partial \beta_0} * \frac{\partial y}{\partial u}$$

$$\frac{\partial u}{\partial \beta_0} = 0 * q_i \beta_0^{0-1} - 1 * \beta_0^{1-1} - 0 * \beta_1 p_i \beta_0^{0-1} = -1$$

$$\frac{\partial y}{\partial u} =$$

Notice that $y = u^2$, so taking partial derivative in respect to u means placing the value of the exponent (2) on the left side of u and decreasing the actual exponent by 1:

$$\frac{\partial y}{\partial u} = (q_i - \beta_0 - \beta_1 p_i)^2 = 2 (q_i - \beta_0 - \beta_1 p_i)^{2-1} = 2 (q_i - \beta_0 - \beta_1 p_i)$$

Now, let's combine all of this together:

$$\frac{\partial y}{\partial \beta_0} = \frac{\partial u}{\partial \beta_0} * \frac{\partial y}{\partial u}$$

$$\frac{\partial y}{\partial \beta_0} = -1 * 2(q_i - \beta_0 - \beta_1 p_i) = -2(q_i - \beta_0 - \beta_1 p_i)$$

5) Take partial derivative of the error E (from step 3 above) in respect to β_0 in order to find the least sum of the squared errors (which means finding the value of β_0 for which the function equals to 0). Since we already solved it above, we only need to add a Σ symbol and replace y with E :

$$\frac{\partial E}{\partial \beta_0} = -2 \sum_i (q_i - \beta_0 - \beta_1 p_i) = 0$$

Simplify the above, and solve for β_0 :

- Divide both sides by -2:

$$\frac{\partial E}{\partial \beta_0} = \sum_i (q_i - \beta_0 - \beta_1 p_i) = 0$$

- Break up the sum by all its parts by removing parentheses (remember that β_0 repeats n times; why? Because is inside Σ):

$$\frac{\partial E}{\partial \beta_0} = \sum q_i - \beta_0 n - \beta_1 \sum p_i = 0$$

/ why β_1 is not multiplied by n? Look at this: let's say $\beta_1 = 2$, $p_1 = 1$, $p_2 = 2$, $p_3 = 3$, then $\beta_1 * p_1 + \beta_1 * p_2 + \beta_1 * p_3 = 2*1 + 2*2 + 2*3 = 2+4+6 = 12$, which can be also written as $\beta_1 * \Sigma p_i = 2 * (1+2+3) = 12$; if however we multiplied β_1 by n (so by 3 p's), then we would get $2 * 3 = 6$, which when multiplied by the sum of p's: $(1+2+3) = 6 * 6 = 36$. Conclusion: use either Σ or n (not both!) /

- Get rid of n by multiplying each part by 1/n:

$$\frac{\partial E}{\partial \beta_0} = \frac{\sum q_i}{n} - \frac{\beta_0 n}{n} - \beta_1 \frac{\sum p_i}{n} = 0$$

- Notice that the sum of the components when divided by n, equals their average values (the formula for average: $\Sigma p_i / n$, and it's represented using a dash at the top):

$$\frac{\partial E}{\partial \beta_0} = \bar{q} - \beta_0 - \beta_1 \bar{p} = 0$$

- Solve for β_0 by adding β_0 to both sides:

$$\bar{q} - \beta_1 \bar{p} = \beta_0$$

$$\beta_0 = \bar{q} - \beta_1 \bar{p}$$

6) Take partial derivative in respect to β_1 :

Like in step 4 above, let's apply the chain rule:

$$\frac{\partial y}{\partial \beta_1} = \frac{\partial u}{\partial \beta_1} * \frac{\partial y}{\partial u}$$

Where:

$$y = (q_i - \beta_0 - \beta_1 p_i)^2 = u^2$$

$$u = q_i - \beta_0 - \beta_1 p_i$$

$$\frac{\partial u}{\partial \beta_1} = 0 * q_i * \beta_1^{0-1} - 0 * \beta_0 * \beta_1^{0-1} - 1 * \beta_1^{1-1} * p_i$$

$$\frac{\partial u}{\partial \beta_1} = -1 p_i$$

$$\frac{\partial y}{\partial u} = 2 (q_i - \beta_0 - \beta_1 p_i)^{2-1} = 2 (q_i - \beta_0 - \beta_1 p_i)$$

$$\frac{\partial u}{\partial \beta_1} * \frac{\partial y}{\partial u} = -1 p_i * 2 (q_i - \beta_0 - \beta_1 p_i) = -2 (q_i - \beta_0 - \beta_1 p_i) p_i$$

7) Take partial derivative of the error E in respect to β_1 :

$$\frac{\partial E}{\partial \beta_1} = -2 \sum_i (q_i - \beta_0 - \beta_1 p_i) * (p_i) = 0$$

Simplify the above, and solve for β_1 :

- Divide both sides by -2:

$$\frac{\partial E}{\partial \beta_1} = \sum_i (q_i - \beta_0 - \beta_1 p_i) * (p_i) = 0$$

- Break up the sum by all its parts multiplied by p_i (like we did in step 5):

$$\frac{\partial E}{\partial \beta_1} = \sum q_i p_i - \beta_0 \sum p_i - \beta_1 \sum p_i^2 = 0$$

- Replace $\sum p_i$ with the average of $p_i * n$:

$$\frac{\partial E}{\partial \beta_1} = \sum q_i p_i - \beta_0 \bar{p} n - \beta_1 \sum p_i^2 = 0$$

- Replace β_0 with the result from point 5 ($\beta_0 = \bar{q} - \beta_1 \bar{p}$):

$$\frac{\partial E}{\partial \beta_1} = \sum q_i p_i - (\bar{q} - \beta_1 \bar{p}) n \bar{p} - \beta_1 \sum p_i^2 = 0$$

- Multiply the parentheses by $n \bar{p}$:

$$\frac{\partial E}{\partial \beta_1} = \sum q_i p_i - (\bar{q} n \bar{p} - \beta_1 \bar{p} n \bar{p}) - \beta_1 \sum p_i^2 = 0$$

- Get rid of parentheses:

$$\frac{\partial E}{\partial \beta_1} = \sum q_i p_i - \bar{q} n \bar{p} + \beta_1 \bar{p} n \bar{p} - \beta_1 \sum p_i^2 = 0$$

- Simplify:

$$\frac{\partial E}{\partial \beta_1} = \sum q_i p_i - \bar{q} n \bar{p} + \beta_1 \left(n \bar{p}^2 - \sum p_i^2 \right) = 0$$

- Subtract the part with β_1 from both sides:

$$\frac{\partial E}{\partial \beta_1} = \sum q_i p_i - \bar{q} n \bar{p} = -\beta_1 \left(n \bar{p}^2 - \sum p_i^2 \right)$$

- Remove - from β_1 :

$$\frac{\partial E}{\partial \beta_1} = \sum q_i p_i - \bar{q} n \bar{p} = \beta_1 \left(\sum p_i^2 - n \bar{p}^2 \right)$$

- Solve for β_1 (divide both sides by $(\sum p_i^2 - n \bar{p}^2)$):

$$\beta_1 = \frac{\sum q_i p_i - \bar{q} n \bar{p}}{\sum p_i^2 - n \bar{p}^2}$$

The next step was to write a computational model using the Python programming language. However, thanks to one of the scientific libraries called sklearn, we did not have to translate the above mathematical equations into Python, but instead we imported the ready LinearRegression() function from this library, which we named 'regressor'. We then used a variety of conditions as our independent variable x, which data collected by Snoopy (variable y) depended on. And finally, we used this linear regression function with the fit(x,y) method in order to find the function that was the closest to all the data points, as well as regressor.coef_ to get the values of this function's coefficients that represented the effect of each condition (variable x) on our data points (variable y). Our model is shown below:

Import libraries and read data from file:

```
In [120]: import pandas as pd

from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

dataset = pd.read_csv("https://raw.githubusercontent.com/btete/Snoopy_Data/main/rawSnoopyData.csv")
# dataset.drop(dataset.index[dataset['Restroom'] == 1], inplace=True)
# dataset.drop(dataset.index[dataset['Incense'] == 1], inplace=True)
# dataset.drop(dataset.index[dataset['Chemicals'] == 1], inplace=True)
dataset
```

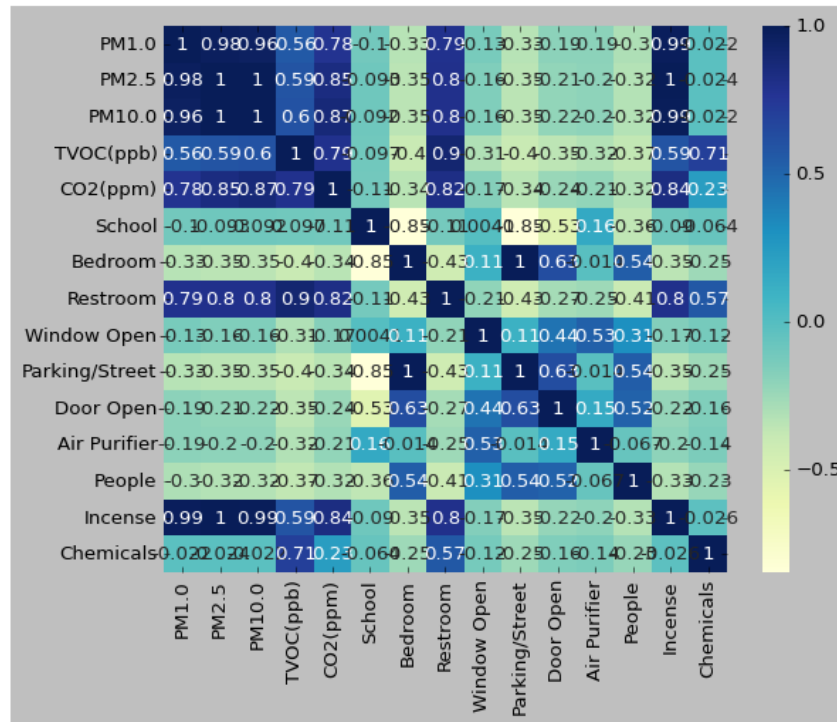
Out[120]:

	MM:DD:YYYY	HH:MM:SS	PM1.0	PM2.5	PM10.0	TVOC(ppb)	CO2(ppm)	School	Bedroom	Restroom	Window Open	Parking/Street	Door Open	Air Purifier	People
0	9/12/2021	3:36:00 PM	4	7	10	0	400	0	1	0	1	1	1	1	1
1	9/12/2021	3:37:00 PM	4	5	8	0	400	0	1	0	1	1	1	1	1
2	9/12/2021	3:38:00 PM	4	7	8	0	400	0	1	0	1	1	1	1	1
3	9/12/2021	3:39:00 PM	4	5	5	0	400	0	1	0	1	1	1	1	1
4	9/12/2021	3:40:00 PM	4	7	7	0	400	0	1	0	1	1	1	1	1
...
2741	1/10/2022	7:51:00 PM	2	3	4	3695	743	0	0	1	0	0	0	0	0
2742	1/10/2022	7:52:00 PM	3	5	8	3580	734	0	0	1	0	0	0	0	0
2743	1/10/2022	7:53:00 PM	3	3	9	3640	733	0	0	1	0	0	0	0	0
2744	1/10/2022	7:54:00 PM	3	5	14	3564	735	0	0	1	0	0	0	0	0
2745	1/10/2022	7:55:00 PM	5	6	12	3364	689	0	0	1	0	0	0	0	0

2746 rows x 17 columns

Looking for relationships in a correlation heatmap:

```
In [30]: sns.heatmap(dataset.corr(), cmap="YlGnBu", annot = True)
plt.show()
```



Thorough analysis of the above graphical representation of the correlation of data and all their conditions indicates that all the factors (or conditions) affected the air quality, incl. burning incense, sprayed chemicals, close location of parking lot and/or street or its lack, presence of people, open/closed window/door, air purifier on/off, but also the actual location of data collection, such as school, bedroom, and restroom. While data gathered at school and bedrooms differed significantly due to being away from parking lot and street, and having better ventilation system and air purifiers at school, as opposed to our bedrooms, the only reason why restroom seems to have such a significant effect on air, is the fact that it is where incense was burnt and chemicals were sprayed. This fact lets us see that using restroom as an independent variable would not be correct, because it's not the location itself that affected the air quality, but the actions performed there. That is why we decided to uncomment the first line with the

dataset.drop() function, and this way exclude the restroom from analyzing the level of pollen. Similarly with people, whose presence seems to have a positive effect on air quality at school. It's because while taking data at school, we had to have air purifiers on and windows open all the time (due to district-wide guidelines caused by the pandemic), which means that our data did not really inform how the presence of people at school affected air quality also when the windows were closed and air purifiers were off during school hours. This made our data lack some additional conditions and circumstances, but due to some limitations, which we couldn't control, our best option was to simply exclude some of these conditions from further analysis.

Select independent variable (features) and dependent variable (outcome):

```
[ ] X = dataset[['Window Open', 'Door Open', 'Air Purifier']] # 'People', 'School', 'Bedroom'
    y = dataset['PM10.0']
    ..
```

As shown on the above screenshot of our code, the independent variables, which we decided to focus on were: Window Open, Door Open and Air Purifier, and as the dependent variable, we used the level of the largest pollen (with a size of 10.0 microns).

Split datasets for validating model:

```
In [123]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
          X_train
Out [123]:
```

As shown above, 80% of data was used for training our model, and 20% was used for testing its accuracy.

Train linear regression model with training data:

```
In [127]: regressor = LinearRegression()  
regressor.fit(X_train, y_train)
```

```
Out[127]: LinearRegression()
```

This is where we used the `LinearRegression()` function imported previously from the `sklearn` library. We first saved it in the `regressor` variable, and then used it with the `fit(x,y)` method in order to train our model, so it could find the best fit linear function for 80% of our data.

Review model intercept and coefficients:

```
[ ] regressor.intercept_  
  
1.2287782035463906
```

```
[ ] coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])  
coeff_df
```

Coefficient	
Window Open	7.581324
Door Open	2.152548
Air Purifier	-2.994513

The above coefficients indicate that opening windows and doors increased the level of pollen, whereas turning air purifier on decreased it.

Test model predictions with testing data:

```
[ ] # PM10 = 1.228 + (WinOpen * 7.5) + (DoorOpen * 2.15) + (AirPurifier * -2.99)

y_pred = regressor.predict(X_test)
```

```
[ ] df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

	Actual	Predicted
182	7	7.968137
224	3	7.968137
1946	0	1.228778
289	2	7.968137
1231	6	3.381326
...
716	1	7.968137
1119	0	-1.765735
2299	0	-1.765735
125	9	7.968137
1150	0	-1.765735

Review model performance:

```
[ ] import numpy as np

# See https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

Mean Absolute Error: 3.72707815803822
Mean Squared Error: 31.004970842460786
Root Mean Squared Error: 5.5682107397673795
```

As explained before, the lower the Root Mean Squared Error (RMSE), the better our model fits actual data. Since its value is only about 5.6, we can assume that it correctly predicts the amount of pollen depending on whether the window and/or door is open or not and/or the air purifier is on or off.

Repeat process above for TVOC, instead of PM10:

```
X = dataset[['Window Open', 'Door Open', 'Air Purifier', 'Incense', 'Chemicals']]
y = dataset['TVOC(ppb)']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
regressor.intercept_
```

```
↳ 471.9253872977935
```

```
2] coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

	Coefficient
Window Open	-157.601225
Door Open	-169.084094
Air Purifier	-126.633370
Incense	3283.929876
Chemicals	5561.337771

Since both incense and chemicals had a significant impact on the level of total volatile organic compounds in the air, we decided to include these factors in the analysis of TVOC. As shown on the above screenshot of our code, simply opening the windows and doors as well as turning the air purifier on helped decrease the level of chemicals in the air, whereas burning incense and spraying chemicals significantly increased it.

```
y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

	Actual	Predicted
2401	222	187.690792
1590	278	302.841294
2381	238	187.690792
2683	2641	3755.855263
1147	322	345.292017
...
1043	186	176.207924
878	12	145.240068
1762	397	471.925387
303	214	18.606698
132	0	18.606698

550 rows x 2 columns

```
[74] print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
      print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
      print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

Mean Absolute Error: 156.42583394082982
Mean Squared Error: 174371.94026412637
Root Mean Squared Error: 417.5786635642755
```

As opposed to RMSE obtained for pollen, this one was much bigger, which means that the model did not predict the amount of TVOC correctly. We were certain that including the outliers, such as incense and chemicals, contributed to this poor performance of our model a lot. Also, incomplete data about the impact of people, which - due to the school district's directives - always required windows to be opened and air purifier to be turned on (which caused the lack of data informing about the effect of people on the air quality when the windows were closed and air purifier was off), might cause the discrepancy between our model and the actual data. After removing these factors, the RMSE, as shown below, improved a lot:

```

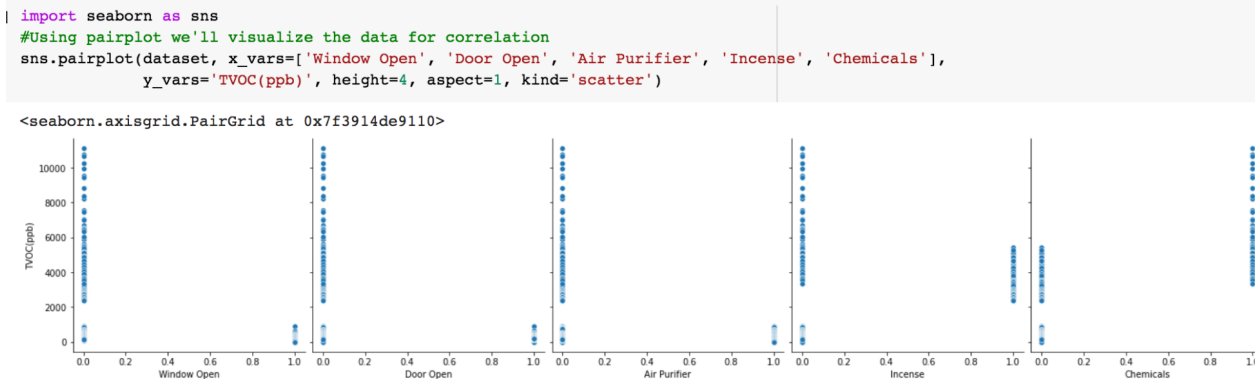
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

Mean Absolute Error: 54.14432043841056
Mean Squared Error: 5086.660344896204
Root Mean Squared Error: 71.32082686632428

```

As shown on the screenshot above, the RMSE significantly improved, but it was still much higher than the one for pollen. We believe that the main reason for it, is the fact that TVOC readings were much larger than PM 10.0 data, and that this higher RMSE might be caused by the scale differences between these two.

Reviewing distribution of data graphically:



As shown on the charts above, closed windows and doors, turned air purifiers off, and sprayed chemicals caused significant accumulation of TVOC in the air. What surprised us here was the higher level of TVOC when the incense was not burnt. This could be caused by the fact that when incense was not burnt in the restroom (that was without a window and with the door closed) we then sprayed a variety of chemicals there. Thus, when the binary value of incense was 0, the binary value of chemicals was 1 (which had a stronger effect on the level of TVOC than incense). We can also clearly see that when chemicals were not sprayed, the amount of TVOC was exactly the same as the amount of TVOC accumulated while burning incense. This finding

lets us clearly see that exploring the presence of the factors without exploring their lack certainly contributed to such a wrong outcome. This could be fixed by measuring the quality of air in the restroom when neither incense nor chemicals were used. Another approach could be separating incense from chemicals by placing them in different locations (e.g. Restroom1 and Restroom2) in our dataset, which would give us better control over them. It would also be more accurate if the same amount of data was gathered for each condition explored. The measurement performed in the restroom took only about two hours, while the measurement of other conditions and locations lasted much longer, spanning even the entire night and day.

Exploring CO2:

```
In [64]: import pandas as pd
import numpy as np

from sklearn import metrics
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

dataset = pd.read_csv("https://raw.githubusercontent.com/btete/Snoopy_Data/main/Snoopy_Data.csv")
dataset
```

```
Out[64]:
```

	School	Bedroom	Restroom	Window Open	Trees	Cars	Door Open	Air Purifier	People	Incense	Chemicals	Avg Pollen/Dust 1.0	Avg Pollen/Dust 2.5	Avg Pollen/Dust 10	Avg CO2	Avg TVOC
0	1	0	0	1	1	0	0	1	1	0	0	8	10	13	411	2142
1	1	0	0	1	1	0	0	1	0	0	0	7	9	12	400	153
2	1	0	0	0	1	0	0	0	0	0	0	5	6	6	400	623
3	0	1	0	0	1	1	0	0	1	0	0	0	0	1	1407	1029
4	0	1	0	1	1	1	1	0	1	0	0	11	16	19	400	22
5	0	1	0	1	1	1	1	1	1	0	0	3	4	5	400	0
6	0	0	1	0	1	1	0	0	0	1	0	295	2139	3273	2868	3782
7	0	0	1	0	1	1	0	0	0	0	1	4	10	26	1427	6036

```
In [65]: X = dataset[['School', 'Window Open', 'Cars', 'Door Open',]]
y = dataset['Avg CO2']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
regressor.intercept_
```

```
Out[65]: 1215.8000000000002
```

```
In [66]: coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

```
Out[66]:
```

	Coefficient
School	-201.2
Window Open	-609.1
Cars	201.2
Door Open	-407.9

```
In [67]: y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df
```

```
Out[67]:
```

	Actual	Predicted
6	2868	1417.0
2	400	1014.6

```
In [48]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

Mean Absolute Error: 1032.8000000000002
Mean Squared Error: 1241567.08
Root Mean Squared Error: 1114.2562908876978
```

Based on the values of the above coefficients, the level of CO2 in school was always at its minimum, as well as opening the window and door always helped with decreasing its amount. However, since RMSE was so high, which made our model even more ineffective than our initial TVOC model, we also decided to exclude the outliers (incense and chemicals) together with incomplete data about people, and this is what we ended up with after modeling linear regression again with our updated dataset:

```

✓ [39] X = dataset[['Window Open', 'Door Open', 'Air Purifier']] # 'People', 'School', 'Bedroom'
0s y = dataset['CO2(ppm)']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
regressor.intercept_

```

400.0

```

✓ [39] coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
0s coeff_df

```

	Coefficient
Window Open	0.0
Door Open	0.0
Air Purifier	0.0

```

✓ [34] y_pred = regressor.predict(X_test)
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df

```

```

✓ [38] print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
0s print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

```

```

Mean Absolute Error: 0.0
Mean Squared Error: 0.0
Root Mean Squared Error: 0.0

```

Both the coefficients and RMSE are equal to zero meaning that our model accurately represents the actual data. Without the outliers and incomplete data about the effect of people on the quality of air, the level of CO2 was always at 400 particles per million parts of air, which is the exact value of the y-intercept as shown above. Thus, our modified CO2 model can be represented by the following equation:

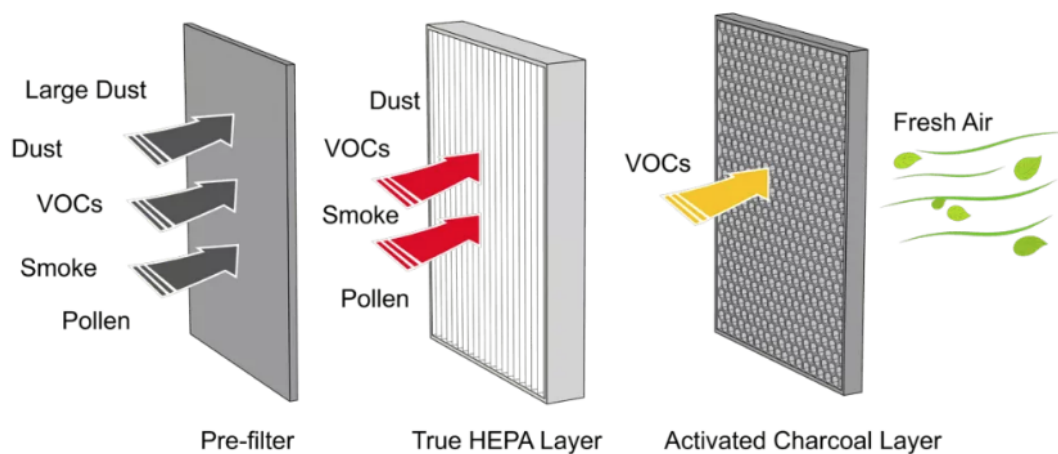
$$y \text{ (CO2 at 400 ppm)} = b \text{ (y-intercept)} + m_1x_1 + m_2x_2 + m_3x_3$$

$$\text{CO2 at 400 ppm} = 400.0 + 0 \cdot \text{Window Open} + 0 \cdot \text{Door Open} + 0 \cdot \text{Air Purifier}$$

$$\text{CO2 at 400 ppm} = 400.0$$

Conclusion

As we've learned, in order to improve the air quality indoors, in addition to helping VOC and CO₂ escape from the room by simply opening the window (if it's OK to bring pollen inside) or at least the door, it would be also very beneficial to replace the carpet with tiles that can be washed frequently, use ecological paints, invest in good ventilation system, and use air purifiers with HEPA filters, like this one:



As officially defined by the U.S. Department of Energy, high efficiency particulate air (HEPA) filters can theoretically remove at least 99.97% of dust, pollen, mold, bacteria, and any airborne particles with a size of 0.3 microns (μm). The diameter specification of 0.3 microns responds to the most penetrating particle size (MPPS) (United States Environmental Protection Agency, 2021).

We also learned that collecting data is a very important process in scientific research. If there is no way that a particular condition (for example the presence of people) can be measured in a variety of circumstances (e.g. open/closed window/door, air purifier on/off), then it's better to exclude such incomplete data from the model. Otherwise, the coefficient representing such a condition will most likely be incorrect, and the entire model will not be an accurate

representation of the reality. Thus, while focusing on a particular condition, it's important to collect an equal amount of data in a variety of circumstances, which will show a more accurate and realistic effect of such a condition on the measured outcomes. In other words, while collecting data when the specific conditions are true, we also should collect data when one of them is true and the other one is false, and vice versa, as well as when all of them are false.

And finally, we strongly believe that collecting the same amount of data for different circumstances should also be taken into account in order to make them more comparable, which would lead to a more accurate model. If gathering the same amount of data for each condition would not be possible, another solution would be a separate model for each explored location.

Collaboration

Roles / Responsibilities

We've shared the responsibilities and roles while working on this project. It all depended on our interests and certain natural skills. Some of us were better in public speaking and oral presentations than others, some others were more interested and skilled in computer programming, whereas others were able to explain the mathematical concepts, which we all learned about, and finally there were also those, who mostly enjoyed working with hardware.

Contributions

We all were meeting regularly after school with our mentors and our teacher-sponsor. We listened to the lectures and followed our mentor while working with hardware. All of us gathered our own data, which we then plotted individually using our collaboratively developed Python

program. We all also explored and studied all the mathematical concepts used in linear regression, including line function, partial derivatives, chain rule, etc., as well as we learned how to write mathematical equations using LaTeX to explain these concepts. We then were regularly meeting with our other mentor, who taught us how to apply such linear regression in a computational model in order to perform statistical analysis and to predict outcomes from actual data. And finally, we were all working together on preparing this report as well as the final presentation.

Works Cited

- Advanced Solutions Nederland B.V. “How TVOC affects indoor air quality: effects on wellbeing and health.” *Advanced Solutions Nederland B.V.*, 3 December 2020, <https://www.advsolned.com/how-tvoc-affects-indoor-air-quality-effects-on-wellbeing-and-health/>. Accessed 9 November 2021.
- Ang, Carmen, et al. “Zooming In: Visualizing the Relative Size of Particles.” *Visual Capitalist*, 10 October 2020, <https://www.visualcapitalist.com/visualizing-relative-size-of-particles/>. Accessed 8 November 2021.
- Brownlee, Jason. “Linear Regression for Machine Learning”. *Machine Learning Mastery*, 15 August, 2020, <https://machinelearningmastery.com/linear-regression-for-machine-learning>. Accessed 26 March, 2022.
- Fields, Lisa, and Carol DerSarkissian. “Pollen Allergies Overview.” *WebMD*, 19 April 2021, https://www.webmd.com/allergies/pollen_allergies_overview. Accessed 14 November 2021.
- Lindsey, Rebecca. “Climate Change: Atmospheric Carbon Dioxide.” *Understanding Climate*, climate.gov, 14 August 2020, <https://www.climate.gov/news-features/understanding-climate/climate-change-atmospheric-carbon-dioxide>. Accessed 10 November 2021.
- Smith, Dianna. “Everything You Need To Know About Carbon Dioxide (CO2).” *kaiterra*, 25 September 2019, <https://learn.kaiterra.com/en/air-academy/carbon-dioxide-co2>. Accessed 10 November 2019.
- United States Environmental Protection Agency. “What is a HEPA filter?” *EPA*, 3 March 2021, <https://www.epa.gov/indoor-air-quality-iaq/what-hepa-filter-1>. Accessed 8 November 2021.

The World Green Building Council. “Indoor Environmental Quality: A How-To Guide.”

WorldGBC, 2020,

<https://www.worldgbc.org/sites/default/files/bp-resource/BPFP-IEQ-Guidance-Note.pdf>.

Accessed 9 November 2021.

Zach. “How to Interpret Root Mean Square Error (RMSE)”. *Statology*, 10 May 2021,

<https://www.statology.org/how-to-interpret-rmse>. Accessed 26 March 2022.

A Ray of Hope

New Mexico
Supercomputing Challenge
Final Report
April 4, 2022

Team 21
Los Alamos High School

Team Member
Andrew Morgan

Project Mentor
Nathaniel Morgan

Executive Summary:

My project is a study of global warming from the viewpoint of light transport. This will be a two-step process. For the first step, I started off by creating a program that simulates the bouncing of light through a scene in a realistic manner. I used a method of a light simulation called backward path tracing (a version of path tracing that starts at the viewpoint instead of the light-emitting objects). Path tracing works by taking a photon (represented by a ray) and tracing its path through a scene as it bounces off surfaces. Based on the collisions and material properties of the objects the ray hit, a final color is designated to that simulated pixel. Due to the random nature of light, if you just simulate one photon, then the image will turn out very noisy. Because of this, you have to simulate many photons per pixel and take the average of them. This takes a very long time, although it gives highly accurate results. The next step, which will be the center of a future project, will be to simulate the actual heating by applying what I found in the first step. Path tracing can be used to find out the effects of gasses and clouds on global warming by finding what percentage of light gets trapped and for how long. The longer light is trapped, the hotter the earth; the quicker the light bounces out to space the less the earth warms. Some more simple approximations can be used to speed up the code at a slight loss of accuracy (likely not noticeable).

Problem Definition:

This project will explore the impact of various gasses and clouds on global warming. I will write a program that traces rays of light through clouds, the atmosphere, and gasses. This is to see how different gasses and varying amounts of cloud cover will affect the warming of the earth over time. Figure 1 graphically illustrates the physical process of how clouds influence the movement of light, and therefore energy.

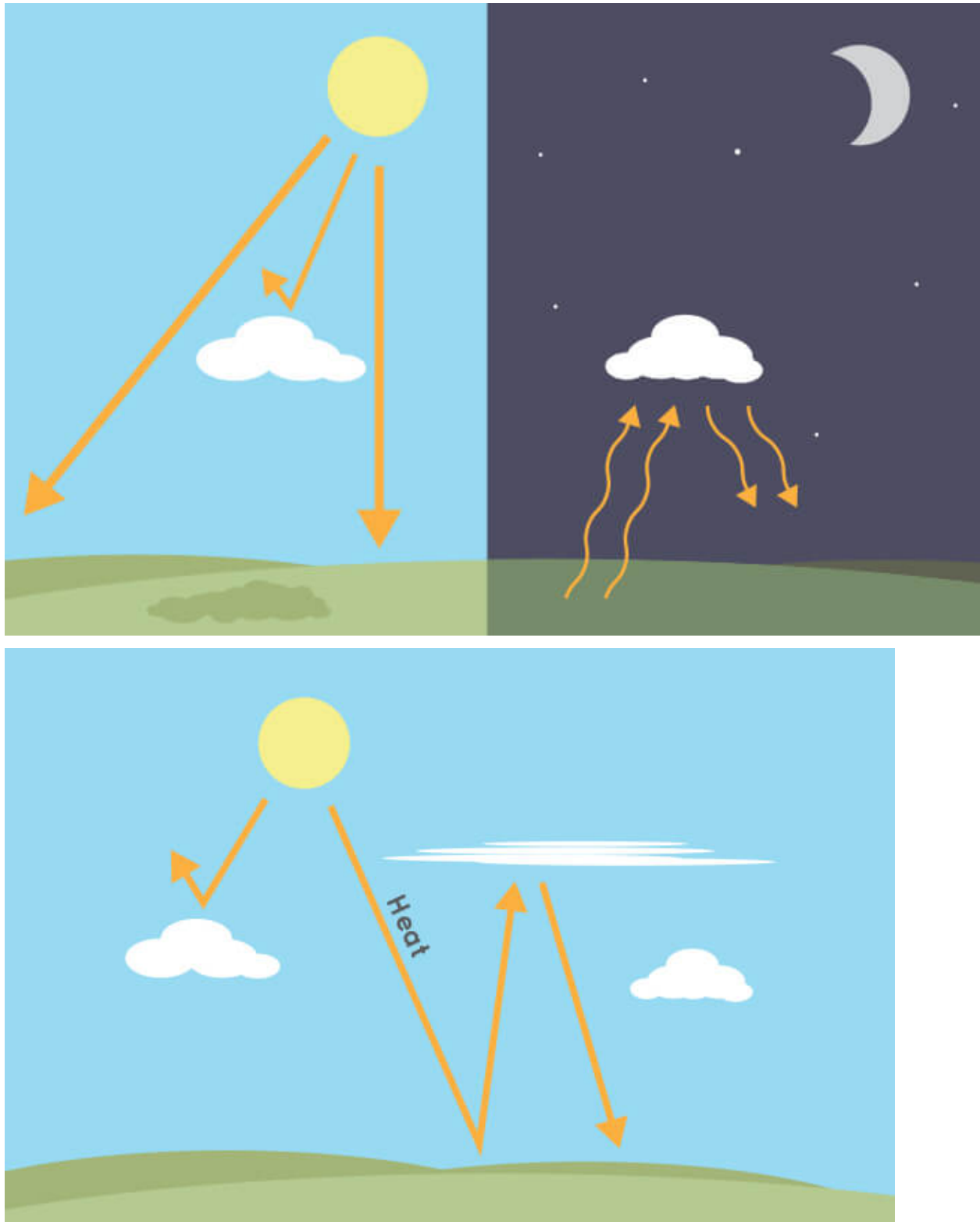


Fig. 1: This figure illustrates how the path of light can change when clouds are accounted for. Images taken from <https://climatekids.nasa.gov/cloud-climate/>

Method Description:

To simulate light bouncing through the earth's atmosphere, I will use multiple methods and combine them. The first method which will speed up the simulation immensely is using Beer's law (e to the power of negative total density represents how much light makes it through a volume of stuff). Instead of directly simulating the bouncing of light through clouds and the atmosphere, with varying levels of different gasses. This does mean that the distribution of scattered light, which changes based on the gas, will not be accounted for. Another thing that I will not be accounting for is how different wavelengths of light scatter through a volume of gas. I will find what percentage of light makes it into the atmosphere, and what percentage stays in the atmosphere, and this will represent the heating effects of varying amounts of gas and clouds.

Verification and Validation:

The methods used in this project were verified and validated by using path tracing and showing the resulting render on screen. The images produced properly matched real world situations similar to the one created in the program. The result of the path tracing can be seen in figure 2.

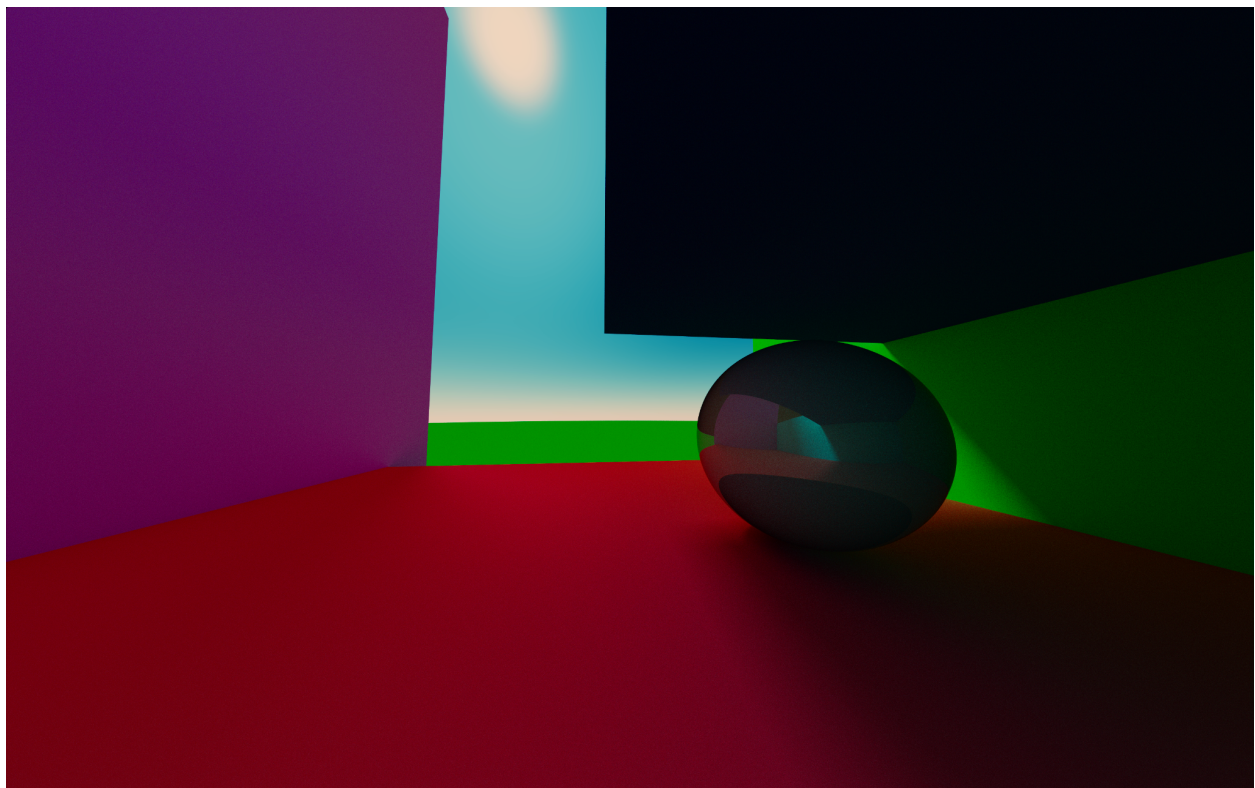
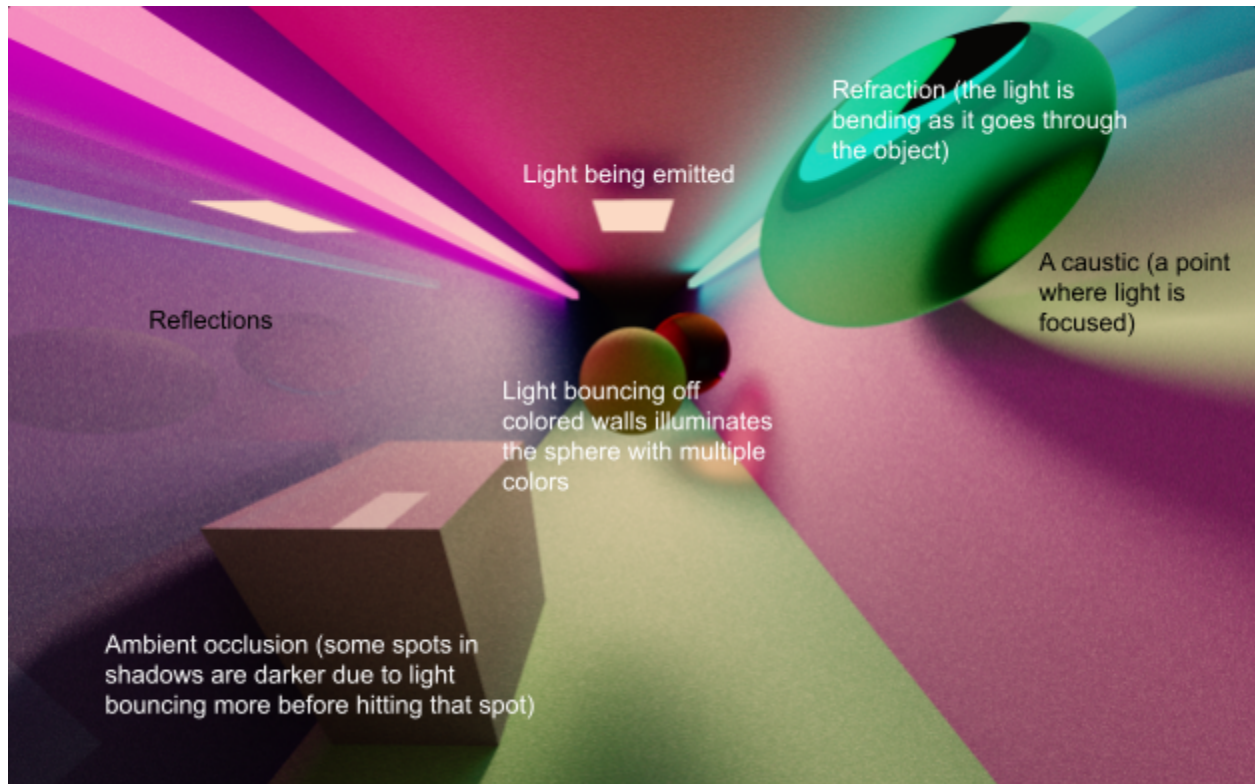


Fig. 2: These images from the code show the same path tracing in an indoor (top) and an outdoor scene (bottom). These scenes include diffuse objects, refractive/translucent sphere, and in the outdoor scene, the atmosphere which uses rayleigh scattering.

Results

When the sunlight strikes the atmosphere, like shown in figure 3, the rays of light scatter depending on the wavelength of the light. This means that at a far distance all of the blue light and a lot of the green light have been scattered off. An example where this happens is at sunset when the sun has to make it farther through the atmosphere before hitting the viewer's eye. During the day the sun doesn't have as far to go through the atmosphere, so very little green and red light are scattered, so the light appears mostly blue. At sea level the sunlight has to go farther compared to at higher altitudes, so in the day the sky is whiter due to more green and red light scatter. Inversely high up in the mountains, the sunlight has less distance to travel to your eye so it's a deeper darker blue.

When sunlight hits a cloud, like shown in figure 4, much of it is bounced back into space. Although if the cloud is thin and facing the direction of the sun, a lot of the sunlight makes it through the cloud, producing super bright highlights on the surface of the cloud. This likely can cause a change in the heating of the earth. The code correctly shows how the clouds can darken light, as well as have the bright highlights in thinner sections that are facing the sun.

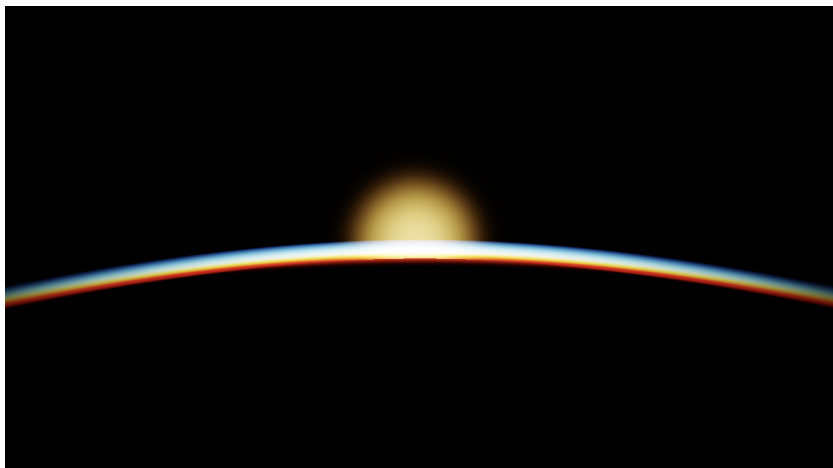
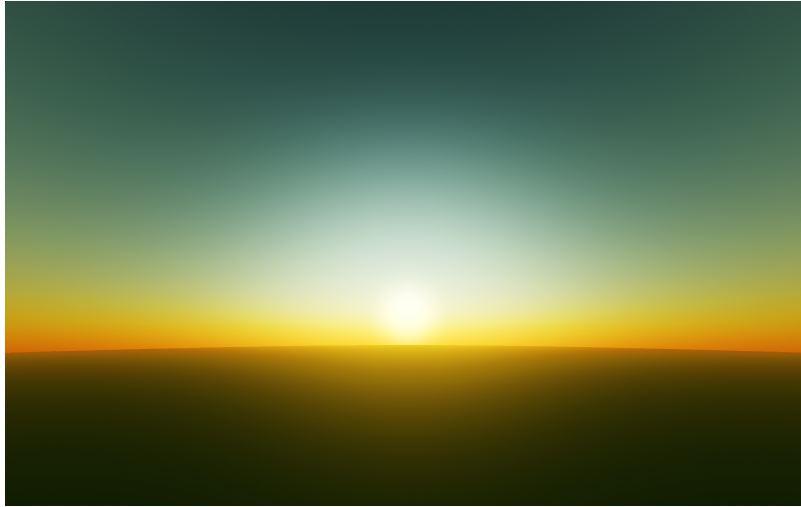


Fig. 3: The images from the code show the atmosphere correctly scattering light based on the angle of the sun in the sky respective to where the camera is viewing it from. The first two images show the sunrise and morning from an altitude of 7,500 feet above sea level and the third image shows the sunset from about the height of the international space station.

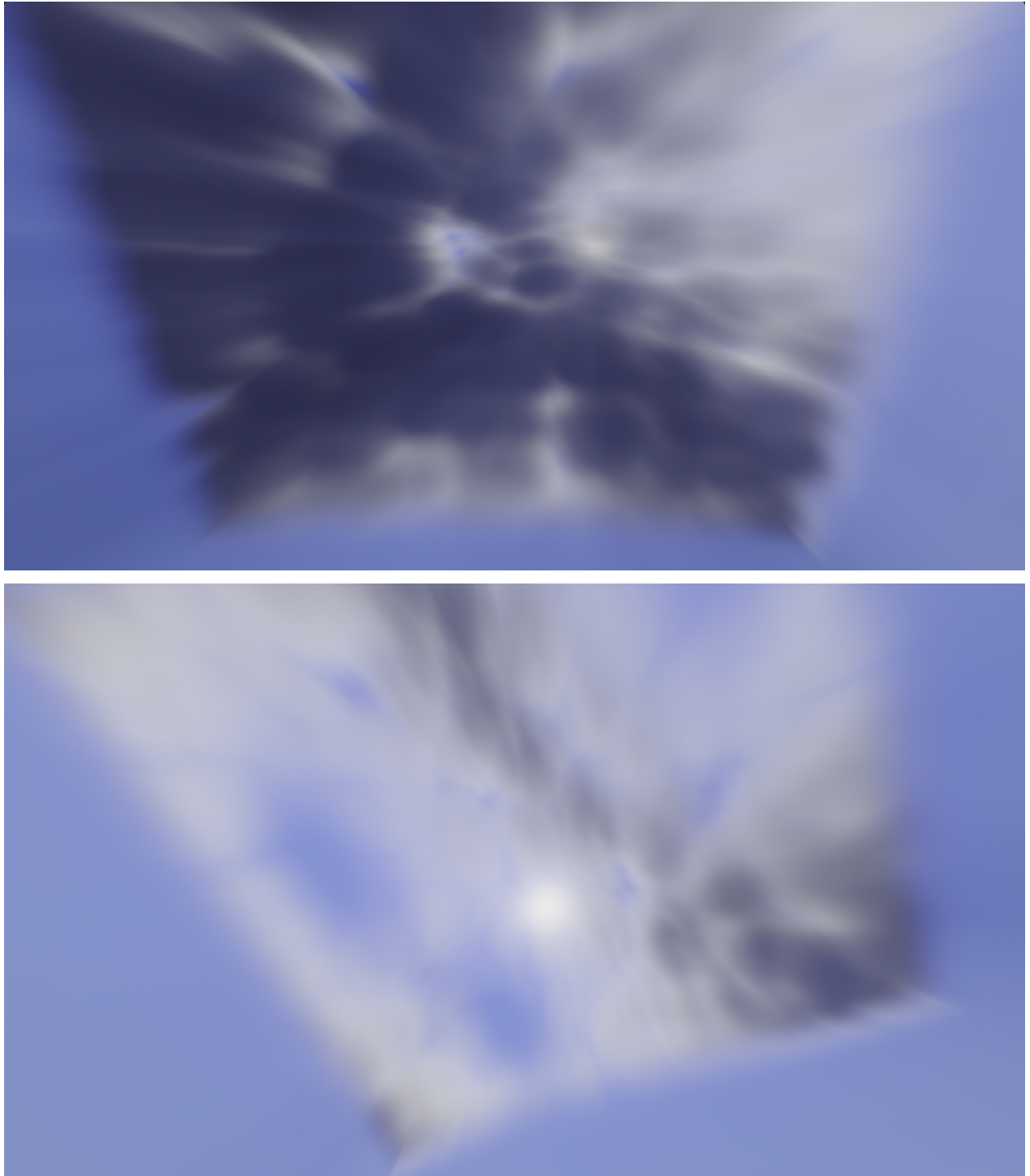


Fig. 4: The images shown from the code show how clouds can darken the light.

Conclusions:

The light simulation code replicates the bouncing of light in an environment using a technique called path tracing. The code was validated by visualizing light bouncing around a room or an outdoor setting, and gathering up the light over multiple samples. I made this to test if my methods would accurately simulate light by generating an image and comparing the output to real-world scenarios.

After the validation was completed, the code was used to simulate light transport in the atmosphere. When the outputted images were compared to images of real-world situations, it matched the images fairly well. This validates the use of path tracing for tracking the effects of gas and clouds on global warming.

I created a light simulation code but I still need to join this together into a program that tracks the amount of trapped light in the atmosphere versus escaped light. Rays that stay for more bounces contribute to more heat while rays that stay for a shorter amount will deposit their energy somewhere else. I can use this to simulate global warming by finding how much and how long rays stay in the atmosphere. The methane, CO₂, and other gasses are expected to increase the amount of light trapped in the atmosphere. There's a good chance that, at some point, the increased heat will cause more clouds to build up. The thickening of the clouds may lead to the earth cooling instead of additional heating. This cooling may lead to fewer clouds, which would lead to higher temperatures. This would mean that more clouds would begin to form again creating a feedback loop that keeps it from reaching even higher temperatures.

Code:

The code is written in GLSL (OpenGL shader language), the language is similar to C++ and automatically has GPU parallelization to speed up the code. The source codes that I developed for the simulations, which in total is over 2,000 lines in length, are at the following web links:

Atmosphere (the default view is from the international space station):

<https://www.shadertoy.com/view/Ns3GzB>

Path Tracing:

<https://www.shadertoy.com/view/7lt3DI>

Clouds (has other non-physically correct stuff, just the clouds are correct):

<https://www.shadertoy.com/view/fs33zB>

References:

<https://climatekids.nasa.gov/cloud-climate/>

<https://www.youtube.com/watch?v=DxfEbulyFcY&t=702s>

<https://www.youtube.com/watch?v=4QOcCGI6xOU&t=369s>

<https://www.youtube.com/playlist?list=PLujxSBD-JXgnGmsn7gEyN28P1DnRZG7qi>

**Effects of Different Combinations of Halogen Additions to *anti*-B₁₈H₂₂ Molecule on
Absorption and Emission Wavelength**

New Mexico

Supercomputing Challenge

Final Report

April 6th, 2022

Team Number: #24

La Cueva High School

Team Members:

Melody Yeh

Teacher:

Mr. Creighton Edington

Project Mentors:

Thomas Peng

William Bricker

Abstract

Downshifting, the process of absorbing high energy photons and emitting them at lower energies, is promising to improving performance and lifespan of solar panels, especially those in space. With modifications to the structure of fluorescent molecules, the downshifting ability of the molecules can be optimized. One such way is through halogenation, which enhances the ability to absorb and emit light at longer wavelengths. By adding combinations of different halogens onto *anti*-B₁₈H₂₂, a naturally fluorescent boron cluster, absorption, emission, and the shift from absorption to emission can be increased. Halogenation of combinations of fluorine or chlorine atoms on the *anti*-B₁₈H₂₂ molecule appears to show a positive trend between the size of the halogens and the length of the emission wavelength. Although the effects of halogen combinations on the absorption to emission shift is uncertain, the use of multiple types of halogens was able to widen the shift by more than 30 nm from the original *anti*-B₁₈H₂₂ molecule.

Rationale

Solar panel cells are a key, renewable source of energy for generating power, especially in space vehicles. The cells in space station solar arrays use the photovoltaic effect to convert light into electricity (Garcia, 2017). Photovoltaic cells mostly utilize visible light with longer wavelengths, such as light in the red or yellow range of the visible spectrum (Pius & Benaiah, 2015). In space, there is a much higher presence of ultraviolet (UV) light than on Earth due to a lack of a shielding atmosphere; however, not only are UV light waves difficult to harvest because of their high energy content, but they also degrade the physical composition of solar panels (National Aeronautics and Space Administration, Science Mission Directorate, 2010; Shamachurn & Betts, 2016). AFRL's various missions largely depend on space vehicles, which has a high reliance on solar panels for power but solar panels are not built to endure the high

intensity of UV light, given that their initial intent was to be used in a much more UV-protected environment, Earth. One solution to collecting UV and protecting solar panels in space is to apply the concept of downshifting molecules, which is absorbing high energy photons and emitting them at a lower energy. By downshifting molecules, UV light can be absorbed and emitted as visible light for the solar panels to harness, yet the extent of its efficiency is still uncertain.

Currently, AFRL, the University of California – Los Angeles (UCLA), and the University of New Mexico (UNM) all have combined efforts working in hopes to understand the effects of halogenation on photoluminescent properties and to optimize the geometry of *anti*-B₁₈H₂₂ molecules. The reason *anti*-B₁₈H₂₂ is chosen is because it has natural fluorescence, high photodegradation resistance, stability to radiation, high quantum yield and can undergo surface modification (Cerdán et al., 2015). In addition, it is important if halogen additions can improve the molecule's downshifting ability, for they are easier to attach to the molecule in lab. By simulating models on computers and synthesizing samples in lab, these three affiliations are working with the goal of using the molecules to develop an application, as simple as a spray, to easily apply onto solar arrays and efficiently utilize the UV light in space. The purpose of this project is to determine how adding different combinations of halogens to the specific molecule, *anti*-B₁₈H₂₂, can increase its absorption and emission wavelengths. AFRL, UCLA, and UNM are mainly focused on examining the effects of the types of halogens, the number of halogens, and symmetry so this work is intended to branch upon their efforts to further understand how halogen types affect photoluminescent properties.

Understanding their correlation can narrow down future research scopes and direct further investigations on more favorable configurations for molecules to downshift or even the

development of UV dependent panels. The potential impact of these efforts is that they can broaden the range of wavelengths that solar cells can utilize and protect the interior of solar panels from intense UV light waves in space, which will in turn increase their efficiency and reduce frequency or costs of solar array maintenance.

Method

I focused on how combinations of halogens will affect the absorption wavelength, emission wavelength, and shift from absorption to emission in derivatives of *anti*-B₁₈H₂₂. To do so, I simulated a total of 19 derivatives, with one *anti*-B₁₈H₂₂ molecule without halogenation, 2 with only fluorine or chlorine at 4 and 4', 4 combinations of fluorine and chlorine at the specific vertices, 1,1', 4, and 4', 4 combinations at vertices 3, 3', 4, and 4', and 8 combinations at the six specific vertices, 1, 1', 3, 3', 4, and 4'.

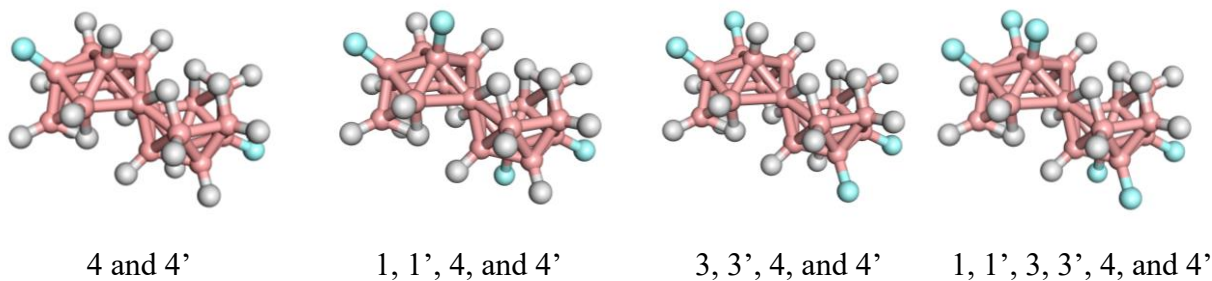


Figure 1. Examples of halogenation at different specified vertices. Halogen additions are shown in blue, borons in pink, and hydrogens in gray

Since it is unclear whether the 1 and 1' vertices or the 3 and 3' vertices are a better choice in trying to improve photoluminescence properties, I wanted to test the combinations for both, at 1, 1', 4, 4' and 3, 3', 4, 4'. This way, not only can the data show the effects of combinations of halogens, but it can also show whether the location of the halogens has an impact on absorption or emission. The reason why I wanted to focus on combinations for my project was to see if adding different halogens would increase the advantages of only using one type of halogen, if

they would cancel out the favorable properties, or if the atoms affect the molecule's photoluminescence properties independently.

The data collection is dependent on using computer simulations to find absorption and emission wavelengths of the halogenated *anti*-B₁₈H₂₂ derivatives. To run the simulations and the data, I primarily use five programs: Discovery Studio Visualizer, PuTTY, Notepad++, WinSCP, and MATLAB. To start off, I simulate the molecular structure of the *anti*-B₁₈H₂₂ derivative I am analyzing in Discovery Studio Visualizer and save the XYZ coordinates of all the atoms. Using the software NWChem and the Dunning basis set, I implement the XYZ coordinates of the molecule and run time-dependent density functional theory calculations to find the energies of the most probable excited states. This provides the information for absorption. When doing so, I have to use Notepad++ to create the files and bash scripts to be run, WinSCP to transfer these files from my local computer to a remote supercomputer, and PuTTY to access and run the jobs on the supercomputer. Then, I optimize the geometry of the molecule at its most probable excited state and when the optimization converges, record the total oscillator strength and energy for the most probable excited state in emission. Finally, I graph the data sets in MATLAB to see the absorption and emission spectrum visually. The files used to call the supercomputer, to optimize the geometry of the molecules for absorption and emission, and to graph the data sets were created with the help of my fellow scholar Mariah and secondary mentor Professor Bricker.

Total oscillator strength is directly related to the probability of reaching a particular excited state, so for both absorption and emission, only the highest total oscillator strength is important. Furthermore, in order to find the wavelengths for absorption and emission, I needed to divide 1284 by the energies recorded.

Data/Results

The absorption and emission data for all 19 derivatives of *anti*-B₁₈H₂₂ are listed below in the table.

#	Vertex 4/4'	Vertex 3/3'	Vertex 1/1'	Absorption (nm)	Emission (nm)	Shift (nm)
1	-	-	-	332.45	425.31	92.85
2	F			349.40	463.05	113.66
3	Cl			363.66	475.12	111.46
4	F	F		341.93	466.57	124.64
5	F	Cl		347.80	473.54	125.74
6	Cl	F		359.42	474.10	114.68
7	Cl	Cl		364.08	478.60	114.52
8	F		F	351.13	455.97	104.84
9	F		Cl	356.85	459.56	102.71
10	Cl		F	365.42	466.11	100.70
11	Cl		Cl	371.48	472.44	100.96
12	F	F	F	347.86	460.58	112.71
13	F	F	Cl	354.40	462.90	108.50
14	F	Cl	F	354.18	468.60	114.42
15	Cl	F	F	364.21	467.91	103.70
16	Cl	Cl	F	368.93	473.52	104.59
17	Cl	F	Cl	370.28	473.87	103.59
18	F	Cl	Cl	359.92	471.42	111.50
19	Cl	Cl	Cl	374.34	478.00	103.65

When collecting the data, I split all the combinations into 4 groups: group A consisted of combinations at 3, 3', 4, and 4'; group B consisted of the combinations at 1,1',4, and 4'; group C consisted of combinations at 1, 1', 3, 3', 4, and 4'; group D consisted of all the other simulations. In each of the groups, both the absorption and emission wavelengths increased as chlorines replaced fluorines. In fact, the longest emission wavelengths occurred where there were only chlorine additions, which was consist for all four groups.

When comparing groups A and B, group A had better results in all three dependent variables and had one of the best shifts out of all 19 simulations, showing that vertices 3 and 3' may be more favorable positions for halogenation. When analyzing the combinations for 6 halides, I also looked more in-depth at the set of having 4 fluorides and 2 chlorides and the set of 2 fluorines and 4 chlorines separately.

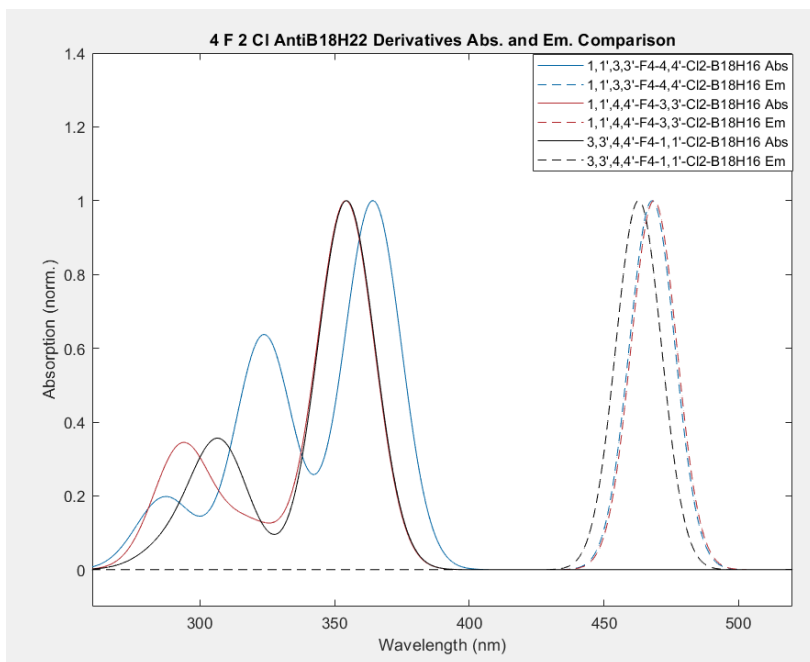


Figure 3. Comparison of the derivatives with 4 fluorines and 2 chlorines. Used Dunning basis

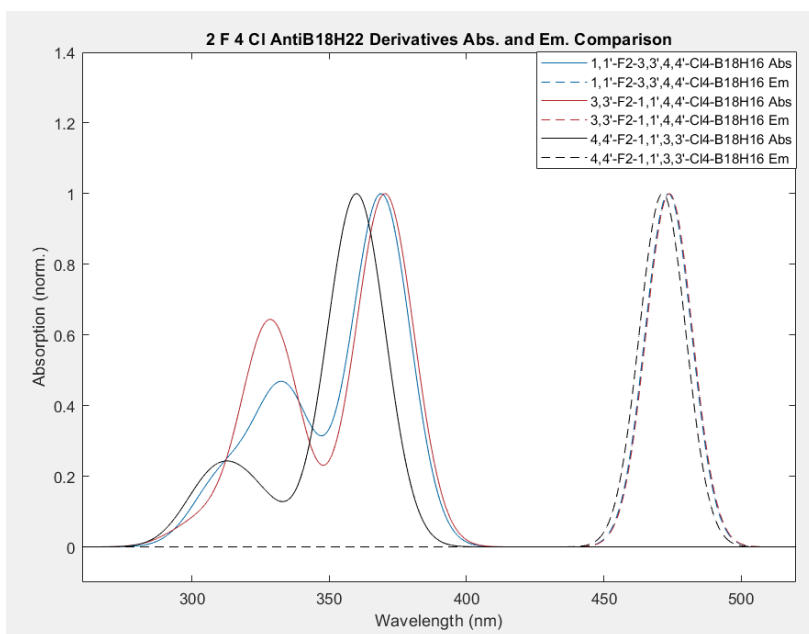


Figure 4. Comparison of the derivatives with 2 fluorines and 4 chlorines.

Used Dunning basis.

As the graphs above show, the blue line in Figure 3., which represents 1, 1', 3, 3' – F₄ – 4, 4' – Cl₂ – anti-B₁₈H₂₂ and the black line in Figure 4., which represents 4, 4' – F₂ – 1, 1', 3, 3' – Cl₄ – anti-B₁₈H₂₂, appear to have a greater shift variance than the others. Although it is unclear why that is the case, these two both have the different halogen at vertices 4 and 4'. More data will need to be collected in order to make further conclusions about why the shift is different from others.

Discussion/Conclusion

The optimal result is for the molecule to absorb light at a short wavelength in the UV light spectrum (less than 380 nm) and emit them at a longer wavelength in the visible light spectrum (preferably in the green region at 550 nm). The data shows four things. First, absorption and emission appear to be correlated with the size of the halides added, likely due to the greater electron density of heavier atoms. As more chlorines than fluorines are added, both

the absorption and emission wavelength became longer. The only outlier was 4,4'-F₂-1,1',3,3'-Cl₄-*anti*-B₁₈H₁₆, which had absorption and emission wavelengths slightly shorter than the other derivatives with 4 chlorines and 2 fluorines and did not follow the relatively linear increase of the data. In addition, the shift from absorption to emission was a lot higher than the other derivatives for 4 chlorines and 2 fluorines. Second, the shift did not appear to be directly related to either the position or the size of the halogen additions. The three greatest shifts were found with 1,1',3,3',4,4'-F₆-*anti*-B₁₈H₁₆, 1,1',4,4'-F₄-3,3'-Cl₂-*anti*-B₁₈H₁₆, and 4,4'-F₂-1,1',3,3'-Cl₄-*anti*-B₁₈H₁₆ but there does not seem to be any clear relationship between these three molecules. Third, halogenation is indeed advantageous to the photoluminescence properties of *anti*-B₁₈H₂₂, as it showed an increase in both emission wavelengths and the shift. With halogenation, the absorption increased by at least 10 nm, emission increased by at least 35 nm, and the shift increased by at least 11 nm. Even though a longer absorption wavelength is not ideal, the benefits with emission and shift outweighs that downside. Last, vertices 3 and 3' appear to be more favorable than 1 and 1', given that group A simulations gave the best shifts and longer emissions than group B.

To get a more conclusive and complete set of data regarding halogenation of a combination of fluorines and chlorines, I will need to make simulations for combinations of different sets of halogens, such as bromine and iodine. Vertices 4 and 4' are where the *anti*-B₁₈H₂₂ molecules have the most electron position, meaning that halogens are also most likely to go to those vertices. For this reason, it would be most important and feasible to still continue work around or at the vertices 4 and 4'.

In this project, most importantly, I have learned not only the real-world implications of chemistry and computer science, but also an introductory understanding of quantum or

computational chemistry. Through the 19 simulations I ran, I learned that the electron density of the different halogens can increase both absorption and emission wavelengths and that halogenation can make *anti*-B₁₈H₂₂ photoluminescence properties more favorable for downshifting.

I believe that there is a lot of room to continue further exploration of this topic. It would be interesting and important to see how combinations of a different set of halogens affects absorption and emission wavelength. The halogen additions analyzed can be of an overall larger electron density, two different types of halogens with great electron density differences from each other, or even completely different types of functional groups. Seeing how the groups of halogen additions on different vertices of the *anti*-B₁₈H₂₂ molecule can also show effects of position on the photoluminescent properties.

Acknowledgements

I would like to thank Dr. Thomas Peng for his guidance and mentorship this summer and Professor William Bricker for all his help in resolving issues with simulations and for explaining concepts for the background knowledge of this project. I would like to thank all the AML scholars, especially Mariah Quigley for her assistance, support, and patience throughout the internship to help me start my project, and Kierstyn Anderson for her continued efforts in synthesizing experimental data for comparison to the simulations and helping me solidify my understanding of this project. I would also like to thank Mr. Creighton Edington for all his advice and suggestions relating to my project. Lastly, I would like to thank the UNM Center for Advanced Research Computing, supported in part by the National Science Foundation, for providing the high-performance computing resources used in this work and MATLAB for its visualizations.

References

- Cerdán, L., Braborec, J., Garcia-Moreno, I., Costela, A., & Londesborough, M. G. (2015). A borane laser. *Nature Communications*, 6(1). <https://doi.org/10.1038/ncomms6958>
- Garcia, M. (2017, July 31). *About the Space Station Solar Arrays*. NASA. https://www.nasa.gov/mission_pages/station/structure/elements/solar_arrays-about.html.
- Help center*. Documentation Home. (n.d.). https://www.mathworks.com/help/index.html?s_tid=CRUX_lftnav.
- National Aeronautics and Space Administration, Science Mission Directorate. (2010). Ultraviolet Waves. Retrieved *June 23, 2021* from NASA Science website: http://science.nasa.gov/ems/10_ultravioletwaves
- Pius, O. E., & Benaiah, B. (2015). Investigating the Wavelength of Light and Its Effects on the Performance of a Solar Photovoltaic Module. *International Journal of Innovative Research in Computer Science & Technology (IJIRCST)*, 3(4). <https://www.ijircst.org/DOC/111e277ced-3ebf-4c57-870d-bc4ec2c9151f.pdf>.
- Shamachurn, H., & Betts, T. (2016). Experimental Study of the Degradation of Silicon Photovoltaic Devices under Ultraviolet Radiation Exposure. *Journal of Solar Energy*, 2016, 1–9. <https://doi.org/10.1155/2016/2473245>

Designing Proteins with Quantum and Neuromorphic Computing

Robert Strauss

April 6, 2022

New Mexico Supercomputing Challenge

Final Report

Team 27

Los Alamos High School

Team Members

- Robert Strauss

Teachers

- Dr. Vikram Mulligan

- Dr. Garrett Kenyon

Project Mentor

- Kyle Henke

Contents

1	Executive Summary	3
2	Introduction	4
2.1	The Protein Design Problem	4
2.2	Quantum Annealing Computing	4
2.3	Neuromorphic Computing	4
2.4	Quadratic Unconstrained Binary Optimization	4
3	Research Question	5
4	Prior Work	5
5	The Challenge: Encoding Protein Design as a QUBO	5
6	Implementation	5
7	Results of Hardware Comparison	6
8	Novel Encoding Approach	6
9	Conclusion	7

1 Executive Summary

I acquired data of residue interaction energies for many protein-packing problems. I wrote python code transforming this data into Q matrices for each problem, involving creating an encoding scheme to represent the residue sequence in binary in such a way tabulated interaction energies can be translated into the Q matrix. I wrote code to upload this Q matrix to a quantum computer (the D-Wave 2000Q) and a neuromorphic computer (the Intel loihi), with instructions to draw several thousand samples by repeatedly annealing or simulating. I received the set of samples from each machine, and compared these solutions to solutions generated by a classical simulated annealing algorithm. I analyzed which hardware performed the best and wrote various tests to determine how my algorithm could be improved. I created a novel encoding scheme which, tested on the isomorphic map-coloring problem, vastly outperformed the algorithm currently used in the literature, producing 3x as many binary sequences that represented valid solutions.

2 Introduction

2.1 The Protein Design Problem

Biological proteins are composed of a chain of amino acids (the residues) which fold the protein into a definite structure due to the forces between the residues. This structure, which is completely defined by the sequences of amino acids and potential energy of each interaction, gives the protein some function, such as catalyzing a reaction. Protein structure prediction has the goal of calculating the final structure of a protein given its amino acid sequence. This was named breakthrough of the year by Science Magazine, going from being considered infeasible to a solved problem in a matter over the past decades. Protein design is the reverse, starting from a protein structure and going backwards to find the amino acid sequence that would fold to such a structure. This is perhaps more significant, since being able to find the sequence that would fold to a desired structure would allow scientists to create that protein in a lab, giving the ability to make designer proteins for any function. This could mean binding a neurotoxin or COVID molecule to stop it, being a highly precise targeted drug, or self-assembling into biological molecule-scale machines or computers. However, protein design is combinatorially complex. There is an exponential amount of possible sequences, making it infeasible to guess and check on a classical computer.

2.2 Quantum Annealing Computing

Quantum computers can theoretically find a global optimum solution to NP-complete problems in $O(1)$ time, making them excellent for solving NP-complete problems. In reality, neither quantum annealing or quantum gate computing has shown this level of performance yet. An active area of research is finding what problems are best matched for these computers, so the performance they can reach currently is useful. Protein design, being an optimization-based, combinatorially complex problem with large utility, is an excellent candidate to try applying quantum computing to. Quantum annealing, rather than gate computing, is best matched to this scenario.

In a quantum annealer such as the D-Wave 2000Q there is a set of qubits which are either 1 or 0 when observed and collapsed, but can be in a quantum superposition of 1 and 0 before collapse. The values of these qubits (once collapsed) represent the answer to the optimization problem posed by the potential applied to them through quantum entanglement.

2.3 Neuromorphic Computing

Neuromorphic computers don't use the traditional Von-Neumann model of separate memory and computation, but instead take inspiration from the biological brain to perform calculations. These systems are composed of an array of leaky integrate and fire neurons which accumulate voltage from spike signals received from other neurons, then fire their own spike signals through synapse transmission lines. This spike-based, memory-integrated model has the advantages of using thousands of times less energy than traditional computers for the same level of calculation [BCHE19] and can solve optimization problems, exploiting random noise as a resource to cross energy barriers [JHM16]. However, it's difficult to exploit these capabilities since these computers can't be programmed with machine code or any programming language. Instead, parameter values have to be loaded onto the device, setting the connectivity of the neurons. This makes it difficult to encode just any problem for this device, but they work especially well with quadratic unconstrained binary optimization.

2.4 Quadratic Unconstrained Binary Optimization

In Quadratic Unconstrained Binary Optimization (QUBO), the values being optimized over are a set of bits. The criterion for optimization is the weighted sum of the products of each pair of bits:

$$E(x) = \sum_i Q_{i,i}x_i + \sum_{i,j} Q_{i,j}x_ix_j \min_x E(X) \quad (1)$$

where x is a binary vector of n bits, Q is the n by n matrix defining the problem, and $E(x)$ will be called the energy. Both the D-wave 2000Q and Intel Loihi are set up with a given Q matrix specific to

the problem, and run a thousand times or more to sample solutions with low energy E (although they may not necessarily find the global minimum).

In the D-Wave, the $Q_{i,j}$ represents the strength of the entanglement made between qubits i and j . If both are 1, this weight contributes to the total energy of the system. Since it starts in the lowest-energy state, the system aims to stay in the global minimum of this function, so when measured the values of the qubits probabilistically collapse, often to a solution minimizing this equation.

Neuromorphic computers also manage to minimize this form of the energy function [JHM16]. If neurons i and j are connected together symmetrically with a synapse of weight proportional to $-Q_{i,j}$, one’s spiking will inhibit the other’s. With the addition of random noise to make spiking more probabilistic, this leads to the system relaxing into a low energy configuration, where whether a neuron is spiking or not represents a bit in the solution.

3 Research Question

The goal of this work was to test the protein design problem on quantum and neuromorphic hardware to see how well these computers were suited for the problem. Simulated annealing, a classical computing algorithm, was applied to the same cases for a comparison. What hardware is better for the problem of protein design: quantum, neuromorphic, or classical computing?

4 Prior Work

Dr. Mulligan *et al.* designed and tested their QPacker algorithm which successfully created a peptide running on the D-Wave 2000Q quantum annealer [MMM⁺19]. This is the only prior example of quantum computing being applied to the protein design problem. Neuromorphic computing has never been applied to the problem before now.

5 The Challenge: Encoding Protein Design as a QUBO

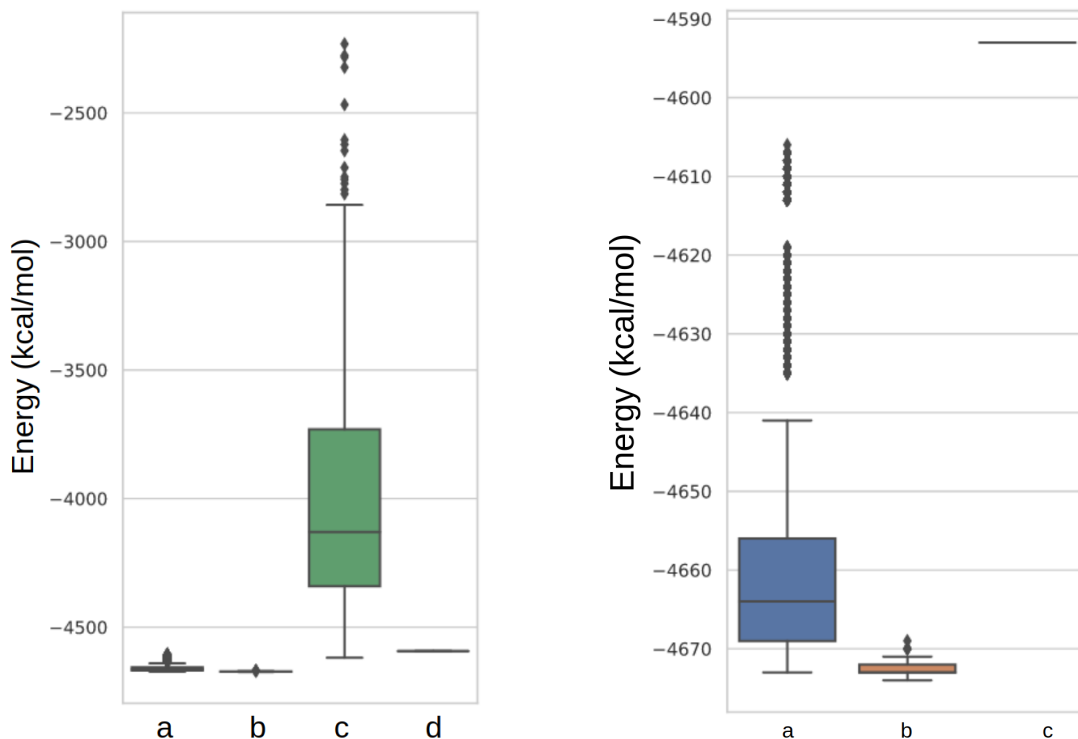
To use quantum or neuromorphic computers for protein design, the protein design problem simply has to be encoded as a Q-matrix according to equation 1, and the optimization parameter x should somehow represent the amino acid sequence in binary. Protein design already has some very convenient parallels with QUBO. Proteins fold to minimize their total potential energy, meaning the sequence of a fold will minimize the sum of one- and two-body interaction energies of contacting residues. In other words, if amino acid A is contacting amino acid B , the interaction energy of A and B is added to the total: $E_{\text{total}} = U(A, B)x_Ax_B$ where x_A is a bit representing whether amino acid A is in the first position, and x_B is a bit representing whether amino acid B is in the other position. So by having a bit for every type of amino acid in every position, the Q-matrix is constructed of copies of a table of the pre-computed interaction energies of every pair of amino acids. This system is called one-hot, where there as many bits per position as there are amino acids, but only one is equal to 1. However, requiring only one bit have the value 1 is a constraint, which QUBO doesn’t allow. This constraint can’t be programmed into the computers, which only take the input of a Q-matrix. Instead, a soft constrain can be used, adding a high energy ”penalty” to solutions with two bits in one sequence position equal to 1. This is a simple addition to the Q-matrix, and should decrease the probability of finding solutions which are invalidly encoded, with more than one non-zero bit in a position.

6 Implementation

I wrote thousands of lines of python code reading in Rosetta amino acid interaction energy data, adding artificial encoding constraint terms, sending the results to quantum and neuromorphic computers, and analyzing solutions.

This included visualization method, file I/O methods, simulated annealing tests, one-hot constraint confirmation, energy calculations, and more.

To prevent confusion of the reader, it’s important to emphasize **quantum and neuromorphic computers don’t run code**. These devices are physics-based systems which evolve to find low-energy



(i) Box-and-whiskers plot showing the distribution of energy (y-axis) of solutions generated by (a) random sequences, (b) simulated annealing, (c) D-Wave 2000Q, and (d) Intel loihi. (ii) Box-and-whiskers plot showing the distribution of energy (y-axis) of solutions generated by (a) random sequences, (b) simulated annealing, and (c) Intel loihi.

solutions to a QUBO problem. They are not programmed, they are set up with a Q-matrix, and run many times to generate a distribution of different samples.

7 Results of Hardware Comparison

On a particular small protein packing problem, I generated 10,000 solution samples from each of simulated annealing, D-Wave 2000Q, and Intel loihi. Due to an unknown technical issue, loihi would appear to fail after only a few samples, producing the same solution repeatedly. I calculated the energy of each solution and plotted the distribution of energies, shown in Figure 1i. Many of the solutions from D-Wave and loihi failed the soft constraint, producing solutions with multiple non-zero bits in a position, an invalid encoding which doesn't represent a sequence. These solutions are included in the energy distribution plot, with the energy penalty term for failing the constraint included. D-wave incurs so much from this penalty that the distribution dwarfs the others. To make the data more clear, I plotted the energy distributions excluding the D-Wave, shown in Figure 1ii. As a control, I generated random amino acid sequences and translated these to the one-hot binary representation of the problem, calculating these energies and plotting them. These random sequences would be very unlikely to fold in a lab, but they at least match the constraint, giving a reference for what kind of energy might be required for a good solution.

8 Novel Encoding Approach

Seeing the clearly worse performance of the D-Wave system, with much higher-energy solutions on average than any other architecture or the randomized control group, I thought about how to improve this given the hardware. I scaled back to the isomorphic map-coloring problem, which has a much simpler Q-matrix and had been done numerous times in the literature [Dah13]. This problem uses the same one-hot encoding to represent the color of countries in a map. The approach in the literature for

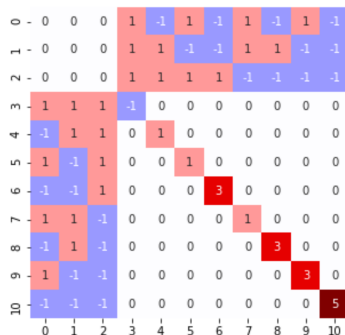


Figure 2: Q matrix of my novel one-hot/binary integer hybrid encoding scheme. The first 3 bits may encode any integer from 0-7 in binary. Of the next 8 (2^3) bits, only the one corresponding to the integer will be 1 to minimize energy. This can be scaled to any number of bits and used in map-coloring and similar QUBO problems.

enforcing one-hot encoding, which I used, implemented it as a soft constraint by assigning a very high energy penalty to solutions with a one-hot encoded with more than one non-zero bit. That is, $Q_{i,j}$ had a very high energy for all i and j representing mutually exclusive options (such as two amino acids in one position). This meant a large portion of values of the Q-matrix were non-zero, which should mean a large number of entanglement connections between each qubit in the D-Wave. However, the D-Wave only supports a finite number of connections between qubits, so when there are too many it creates a chain, connecting one qubit to many proxy qubits with a strong weights to give them all the same value. These chains weaken the efficiency of the annealer, wasting many samples due to chain breaks as qubits in a chain fail to have the same value. To decrease the amount of chains needed, I aimed to create a different way of encoding the sequence or penalty to make the Q matrix more sparse. I came up with a one-hot/binary hybrid encoding system. This had a bit for each option like the one-hot system, but also had a number (equal to the log base 2 of the number of options) of extra bits, which encoded an integer in binary corresponding to which of the bits should be 1. Rather than every bit being connected to all the others, in my novel encoding, each only need be connected to the group of integer-encoding bits. So rather than a dense block of values in the Q-matrix, my novel approach has the Q-matrix shown in Figure 2. This system still pushes towards one-hot solutions with only one amino acid chosen per position, and still has the one-hot representation so coupling the solution to amino acid interaction energies or any other data would be simple. Increasing the percent of validly encoded solutions from 9% to 32%, This innovation **tripled the success rate** of D-Wave, finding 3x as many validly-encoded solutions, **a huge improvement that other quantum researchers could make use of.**

9 Conclusion

Running the same protein design QUBO on classical, quantum, and neuromorphic hardware, I found classical vastly outperformed the others. However, quantum and neuromorphic technology are in their infancy compared to classical computing. As the scale of quantum computers grow and their ability to avoid decoherence improve, they may become more powerful tools for solving problems such as protein design. Neuromorphic and quantum computers alike struggle to adhere to a soft constraint imposed with an energy barrier in a QUBO. I developed a novel encoding scheme which triples the success rate of a quantum computer’s solutions adhering to a soft constraint. I applied this to the case of the map-coloring problem, which can be fit to numerous useful problems with real significance such as the protein design problem.

References

- [BCHE19] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-*

inspired Computational Elements Workshop on - NICE '19, page 1–8. ACM Press, 2019.

- [Dah13] E. D. Dahl. Programming with d-wave: Map coloring problem, Nov 2013.
- [JHM16] Zeno Jonke, Stefan Habenschuss, and Wolfgang Maass. Solving constraint satisfaction problems with networks of spiking neurons. *Frontiers in Neuroscience*, 10, 2016.
- [MMM⁺19] Vikram Khipple Mulligan, Hans Melo, Haley Irene Merritt, Stewart Slocum, Brian D. Weitzner, Andrew M. Watkins, P. Douglas Renfrew, Craig Pelissier, Paramjit S. Arora, and Richard Bonneau. Designing peptides on a quantum computer. Sep 2019.

Classifying Mushroom Edibility

New Mexico

Supercomputing Challenge

Final Report

April 6, 2022

Team 30

La Cueva High School

Team Members

Yana Outkin

Teacher

Yolando Lozano

Classifying Mushroom Edibility

Mushroom hunting is a beloved hobby, especially in Europe. Moreover, there are numerous health benefits due to the antioxidants found in certain mushrooms. But despite the existence of numerous datasets, many apps still misdiagnose common poisonous mushrooms as safe. This problem has negatively impacted the prospects of mushroom-hunting, where hikers are discouraged from picking mushrooms that are unfamiliar to them.

This project uses two datasets, a characteristic dataset from the University of California-Irvine, and an image dataset from WildFoodUK, to computationally predict the edibility of a mushroom given an image. My goal is to provide a framework for utilizing and connecting multiple datasets, in hopes of increasing the accuracy in identifying edibility.

Executive Summary

In hopes of increasing mushroom-hunting prospects, several mobile apps have been designed to identify unknown mushrooms, even going as far as to plot the exact location of where the mushroom was found. But despite the existence of numerous datasets, many apps still misdiagnose common poisonous mushrooms as safe (Shields, 2017). This problem has negatively impacted the prospects of mushroom-hunting, where hikers are further discouraged from picking mushrooms that are unfamiliar to them.

The main reason for the error is that many studies fail to utilize multiple datasets in their research, focusing mainly on a dataset made purely of images or purely of characteristics. For example, with an accuracy of 55%, the following research, Deep Shrooms, utilized a dataset consisting purely of images from Mushroom World (an online database) (Koivisto et al., 2017). Similarly, a study analyzing the speed of the Naive Bayes and Decision Tree algorithms used only the characteristic dataset (Al-Mejibli I., and Abd D. 2017).

My research will focus on optimizing the correlation between multiple datasets to increase the accuracy of computationally identifying the edibility of a mushroom. Furthermore, to make this project more user-friendly, I wanted to have the user input an image, rather than a long list of characteristics, as the input data. This organization will make this project readily available to hikers with little knowledge of specific mushroom characteristics (such as veil, ring type, etc) and to hikers who are currently in the mountains. The ease with which mushroom hunters can upload an image to get the classification will save the time in determining edibility in comparison to using a field guide, be more accurate if the field guides have only a couple of sparse images documenting the mushroom, increase the amount of hikers, and will also help with maintaining forest biodiversity when fewer inedible mushrooms are

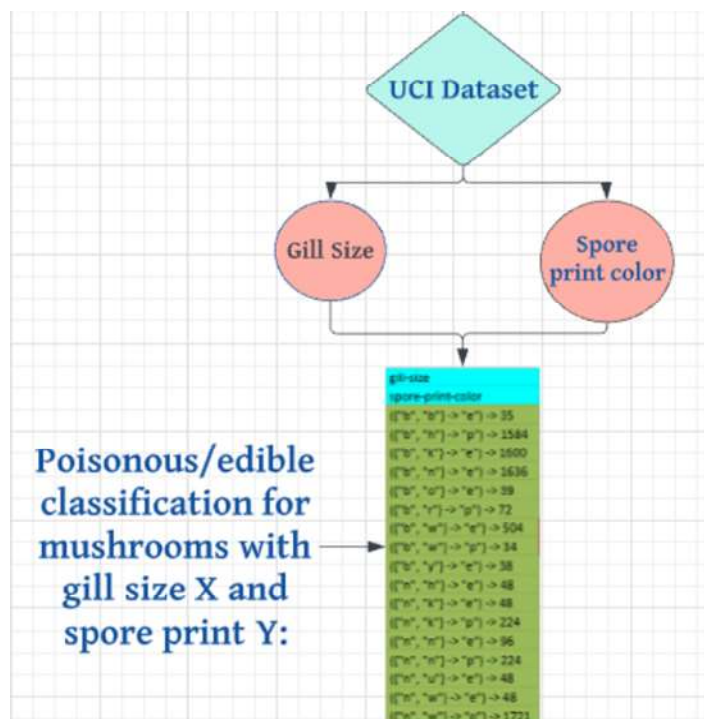
picked.

For my project, I used the Wolfram Language, a flexible language with in-built machine learning functions. I utilized the machine learning functions to extract relevant features from mushroom images, after I created the training datasets. With this programming language, rather than focusing on syntax and specific algorithm structure as I would have needed to do with Java or Python, I was able to focus more on data analysis and finding trends in the datasets.

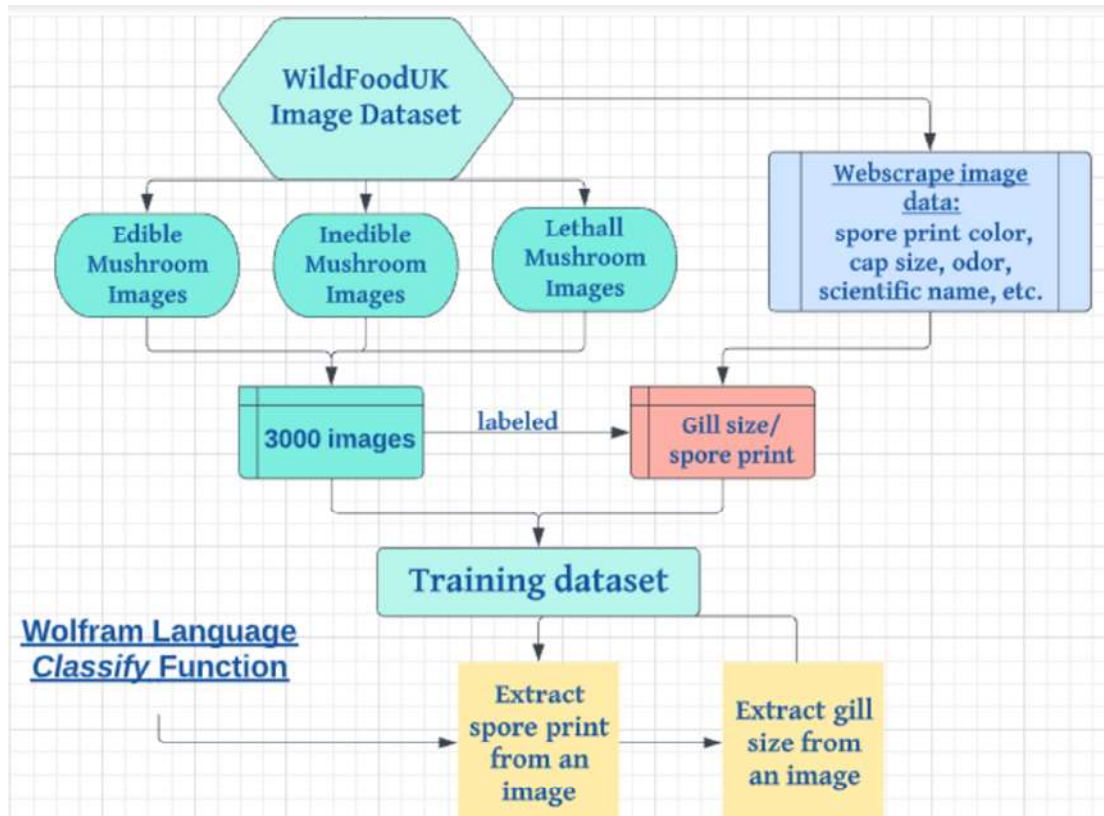
I used two main datasets. The dataset from the University of California Irvine (UCI) documents specific mushroom characteristics and the corresponding edibility of the Agaricus and Lepiota mushroom families. The Agaricaceae mushroom family consists of the most commonly known mushrooms, found mainly across Northern Europe. However, these mushrooms are also found worldwide on a variety of terrains—ranging from dense forests to prairies. They are well-known decomposers, decomposing mulch, leaf litter, and wood. There are also documented online image databases and field guides (Koivisto et al, 2017).

The goal of my project is to take a picture of a mushroom, extract the significant characteristics, and predict the edibility from those characteristics.

My first step is finding the significant characteristics that predict edibility, using the UCI dataset.



Afterwards, I created training images datasets that extract those relevant characteristics from an image, using the Wolfram Language's *Classify* function.



After I created the *Classify* functions to extract the relevant characteristics from an image, I used the UCI dataset to predict edibility.

Enjoy! My paper documents the specific steps I took to link the datasets and then analyze the results. I learned a lot about working with large amounts of data, as well as the limitations of connecting multiple datasets.

Datasets

UCI Dataset

The University of California Irvine dataset is a multivariate, categorical characteristic dataset (Dua and Graph 2019). It contains approximately 8000 mushroom entries, each with 22 characteristics (cap color, gill size, odor, habitat, etc) and an edibility classification (“e” or “p”). Each characteristic contains different attributes, such as cap color “white”, “brown”, “orange”, etc.

<https://archive.ics.uci.edu/ml/datasets/Mushroom>

```
mushroomCharacteristics = Import["cvs file"];
```

(Debug) In[]:= Dataset@mushroomCharacteristics

(Debug) Out[]:=

class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment
p	x	s	n	t	p	f
e	x	s	y	t	a	f
e	b	s	w	t	l	f
p	x	y	w	t	p	f
e	x	s	g	f	n	f
e	x	y	y	t	a	f
e	b	s	w	t	a	f
e	b	y	w	t	l	f
p	x	y	w	t	p	f
e	b	s	y	t	a	f
e	x	y	y	t	l	f
e	x	y	y	t	a	f
e	b	s	y	t	a	f
p	x	y	w	t	p	f
e	x	f	n	f	n	f
e	s	f	g	f	n	f
e	f	f	w	f	n	f
p	x	s	n	t	p	f
p	x	y	w	t	p	f

rows 1-20 of 8125 columns 1-10 of 23

Webscraping: Image Dataset

WildFoodUK contains an extensive database with mushrooms found in the UK, sorted by edible, inedible, and lethally poisonous mushrooms. Each mushroom contains consistent descriptions of aspects such as cap width, cap color, odor, and spore-print-color.

Originally, I was planning on using Mushroom World to create my image dataset because it contains an extensive community-uploaded database with multiple images for each mushroom, along with explanations regarding notable characteristics.

However, I found an even better website. WildFoodUK contains a more extensive database with mushrooms found in the UK, sorted by edible, inedible, and lethally poisonous mushrooms. In addition, in

contrast to Mushroom World, each mushroom contains consistent descriptions of aspects such as cap width, cap color, odor, and spore-print-color. This consistency in the organization of the information makes it great for my computational project. ie, after I extract the images and characteristics for 1 mushroom/website, because of the consistency of the layout, I can easily extract the same information from the rest of the mushroom websites. Webscraping is the process of computationally extracting data from the web.

There are 98 entries for edible mushrooms, 33 inedible entries, and 35 lethally inedible mushrooms. For each entry, I webscraped 10 - 15 images, by going to the individual website documenting that specific mushroom.

An obvious limitation with this dataset, however, is that the mushrooms do not all belong to the *Agaricus* and *Lepiota* family, as is documented by the UCI. Therefore, there may be some discrepancies in classifying edibility for mushrooms not in the *Agaricus* or *Lepiota* family.

Getting the Hyperlinks

Importing the hyperlinks to all the mushrooms listed on the “edible” section of WildFoodUK:

```
(Debug) In[*]:= hyperlinksEdible = Import[  

  "https://www.wildfooduk.com/mushroom-guide/?mushroom_type=edible", "Hyperlinks"];
```

Each mushroom has 3 hyperlinks, removing duplicates:

```
(Debug) In[*]:= goodhyperlinksEdible = First /@ Partition[hyperlinksEdible[[39 ;; -14]], 3];
```

Double checking that there are indeed 98 entries:

```
(Debug) In[*]:= Length@goodhyperlinksEdible
```

```
(Debug) Out[*]:= 98
```

And that each hyperlink is distinct:

```
(Debug) In[*]:= CountDistinct@goodhyperlinksEdible
```

```
(Debug) Out[*]:= 98
```

Inedible hyperlinks:

```
(Debug) In[*]:= hyperlinksPoisonous =  

  Import["https://www.wildfooduk.com/mushroom-guide/?mushroom_type=inedible",  

    "Hyperlinks"];
```

```
(Debug) In[*]:= goodhyperlinksPoisonous = First /@ Partition[(hyperlinksPoisonous[[39 ;; -14]]), 3];
```

Lethally poisonous hyperlinks:

(Debug) In[]:=

```

hyperlinksLethallyPoisonous =
  Import["https://www.wildfooduk.com/mushroom-guide/?mushroom_type=poisonous",
    "Hyperlinks"];

```

(Debug) In[]:=

```

goodhyperlinksLethallyPoisonous =
  First /@ Partition[(hyperlinksLethallyPoisonous[[39 ;; -14]]), 3];

```

Importing images from each of the hyperlinks:

```

edibleMushroomImages =
  Table[(Import[goodhyperlinksEdible[[n]], "Images"])[[10 ;; -10]],
    {n, Length@goodhyperlinksEdible});

```

```

poisonousMushroomImages =
  Table[(Import[goodhyperlinksPoisonous[[n]], "Images"])[[10 ;; -10]],
    {n, Length@goodhyperlinksPoisonous});

```

```

lethallyPoisonousMushroomImages =
  Table[(Import[goodhyperlinksLethallyPoisonous[[n]], "Images"])[[10 ;; -10]],
    {n, Length@goodhyperlinksLethallyPoisonous});

```

Example images from lethally poisonous mushroom #26 - White Domecap:

```
(Debug) In[*]:= lethallyPoisonousMushrooms [ [26] ]
```



In total, there were approximately 4000 training images!

However, as you can see, the training images also contain images other than mushrooms webscraped from the website, such as vouchers, user profile pictures, and mushroom season categories (such as a sun for summer, and a leaf for spring). I was not sure how to filter the images so that there would only be images of mushrooms, instead, I manually deleted the 1000 unneeded images (took about 40 minutes).

The resulting image dataset contains approximately 3000 images.

Random Images - Validating Results

To later test the accuracy, I also compiled a list of random images, 2 for each distinct mushrooms, resulting in 196 random edible mushroom images, and 66 inedible, and 70 lethally poisonous mushroom images.

```
(Debug) In[*]:= randomEdible = Flatten[#[[3 ;; 4]] & /@ edibleMushroomsBetter];
```

```
(Debug) In[*]:= randomInedible = Flatten[#[[3 ;; 4]] & /@ inedibleMushroomsBetter];
```

```
(Debug) In[*]:= randomLethal = Flatten[#[[3 ;; 4]] & /@ lethallyPoisonousMushrooms];
```

```
(Debug) In[ ]:= Length /@ {randomEdible, randomInedible, randomLethal}

{196, 66, 70}
```

Identifying Significant Characteristics

Which characteristics, based of the UCI dataset, are good predictors of edible/poisonous mushrooms?

To determine the relevant characteristics, I analyzed the difference between the amount of poisonous and edible counts. For example, if 80% of entries for cap-color blue are poisonous, then cap-color blue is a relatively high indicator. On the other hand, if 50% of entries for cap-color red are poisonous, then this characteristic gives no valuable information whatsoever, because cap-color red is equally likely to be poisonous as edible.

Single Variable

I first looked at individual characteristics, and the amount of times those mushrooms were edible or poisonous.

Correlation of cap shape v. edibility

```
(Debug) In[ ]:= capShape =
  Table[mushroomCharacteristics[[n, 2]] → mushroomCharacteristics[[n, 1]], {n, 8000}][[
    2 ;; -1]]
```

```
(Debug) Out[ ]:= {x → p, x → e, b → e, x → p, x → e, ... 7989 ... , x → e, k → p, f → e, k → p, k → p}
```

large output show less show more show all set size limit...

Counting the amount of times each attribute of cap shape, “x”, “b”, “s”, “f”, “b”, “k”, and “c”, is poisonous or edible:

```
(Debug) In[ ]:= countsCapShape = Counts[capShape]
```

```
(Debug) Out[ ]:= <| (x → p) → 1703, (x → e) → 1931, (b → e) → 387, (s → e) → 32, (f → e) → 1585,
  (f → p) → 1552, (b → p) → 48, (k → e) → 205, (k → p) → 552, (c → p) → 4 |>
```

By sorting keys, easier to understand datasets and bar graphs

```
(Debug) In[ ]:= KeySort[countsCapShape]
```

```
(Debug) Out[ ]:= <| (b → e) → 387, (b → p) → 48, (c → p) → 4, (f → e) → 1585, (f → p) → 1552,
  (k → e) → 205, (k → p) → 552, (s → e) → 32, (x → e) → 1931, (x → p) → 1703 |>
```

```
(Debug) In[ ]:= Dataset@KeySort [countsCapShape]
```

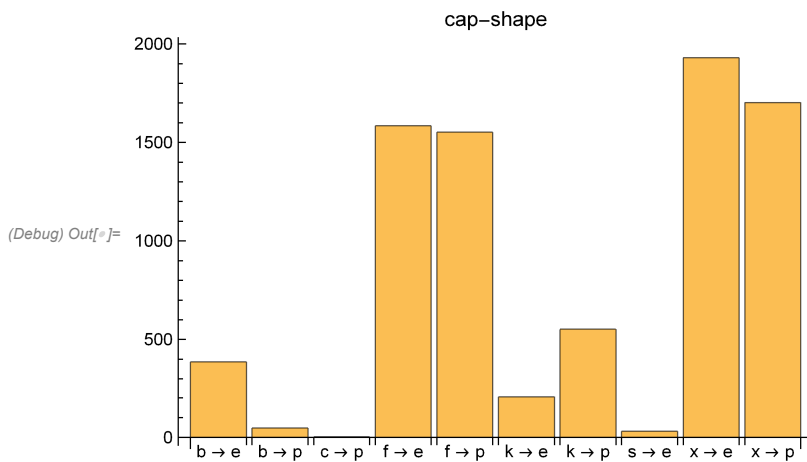
```
(Debug) Out[ ]:=
```

"b" → "e"	387
"b" → "p"	48
"c" → "p"	4
"f" → "e"	1585
"f" → "p"	1552
"k" → "e"	205
"k" → "p"	552
"s" → "e"	32
"x" → "e"	1931
"x" → "p"	1703

Looking at this dataset, one can see that a cap shape of “f”, has an equal amount of poisonous and edible entries associated with it (1585 and 1552 respectively), signaling that cap shape “f” is not a good predictor. This is similar with cap shape “x” (1931 and 1703 respectively for edible and poisonous counts). Overall, cap shape is not a good predictor on its own.

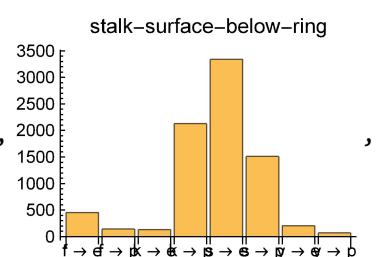
```
(Debug) In[ ]:= BarChart [KeySort [countsCapShape] , ChartLabels → Keys [KeySort [countsCapShape] ] ,
PlotLabel → mushroomCharacteristics [ [1, 2] ]]
```

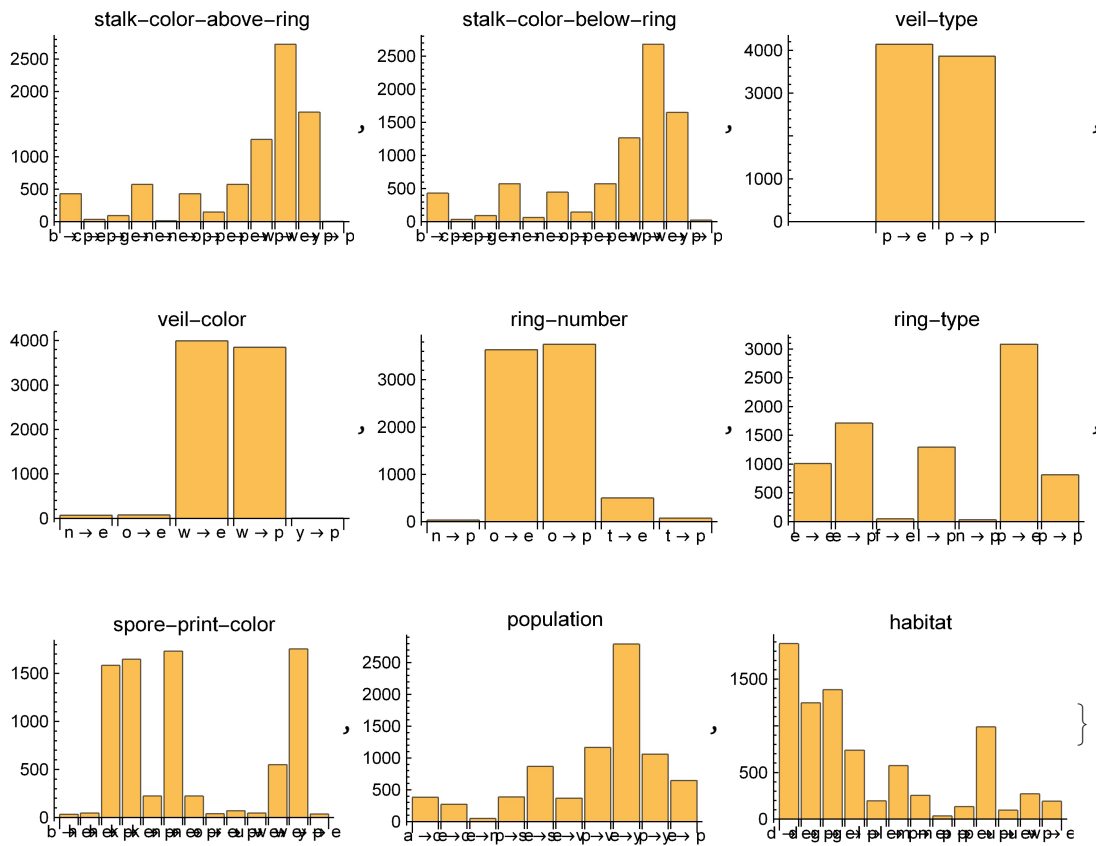
Counts of poisonous/edibility of cap shape in a bar chart:



Bar charts for the rest of the 21 characteristics:

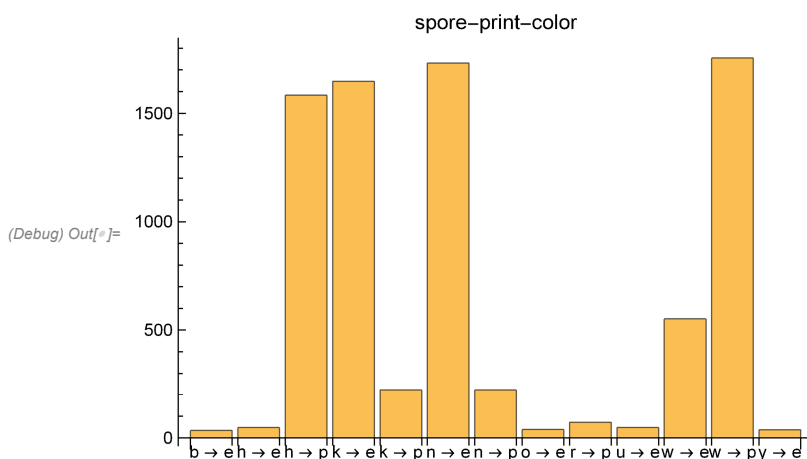
```
(Debug) In[ ]:= allBarChartsLabeled = Table[good = Table[mushroomCharacteristics [[n, characteristic]] →
mushroomCharacteristics [[n, 1]], {n, 8000}][[2 ;; -1]];
countsPE = KeySort [Counts [good]]; BarChart [countsPE, ChartLabels → Keys [countsPE],
PlotLabel → mushroomCharacteristics [[1, characteristic]]], {characteristic, 2, 23}]
```





From first glance, spore print is a good predictor because almost all of the attributes are uniquely poisonous or edible. For example, spore print color “n”, brown, is an indicator for most edible mushrooms.

(Debug) In[]:= **allBarChartsLabeled** [[-3]]



Multiple Variables

The combination of multiple characteristics has a higher probability being more uniquely poisonous or

edible. I repeated the same process for finding good single-variable indicators, but this time with the pairings of multiple variables.

Cap shape (column 2) and cap color (column 3):

```
twoVariables = Table[
  mushroomCharacteristics[[n, {2, 3}]] → mushroomCharacteristics[[n, 1]], {n, 2, 8000}]
```

(Debug) Out[]=

```
{ {n, k} → p, {b, n} → e, {b, n} → e, {n, k} → p, {b, n} → e, {b, k} → e,
  {b, k} → e, {b, n} → e, {n, k} → p, {b, k} → e, {b, n} → e, {b, k} → e,
  {b, n} → e, {n, n} → p, {b, k} → e, {n, n} → e, {b, n} → e, {n, k} → p,
  ... 7964 ..., {n, w} → p, {b, w} → e, {b, w} → e, {b, y} → e, {b, w} → e,
  {n, w} → p, {b, b} → e, {n, w} → p, {n, w} → p, {b, w} → e, {n, w} → p,
  {b, w} → e, {b, o} → e, {n, w} → p, {b, y} → e, {n, w} → p, {n, w} → p }
```

large output

[show less](#)

[show more](#)

[show all](#)

[set size limit...](#)

```
(Debug) In[ ]:= Dataset@KeySort[Counts[twoVariables]]
```

```
(Debug) Out[ ]:=
```

{ "b", "f" } → "e"	40
{ "b", "f" } → "p"	4
{ "b", "g" } → "p"	1
{ "b", "s" } → "e"	217
{ "b", "s" } → "p"	18
{ "b", "y" } → "e"	130
{ "b", "y" } → "p"	25
{ "c", "g" } → "p"	1
{ "c", "y" } → "p"	3
{ "f", "f" } → "e"	688
{ "f", "f" } → "p"	328
{ "f", "g" } → "p"	1
{ "f", "s" } → "e"	295
{ "f", "s" } → "p"	512
{ "f", "y" } → "e"	602
{ "f", "y" } → "p"	711
{ "k", "f" } → "e"	51
{ "k", "f" } → "p"	4
{ "k", "g" } → "p"	1
{ "k", "s" } → "e"	112

rows 1-20 of 30

At least for cap shape and cap color, all the combinations are relatively equal in number (in the 100's), and none of the combinations are obviously poisonous or edible. Therefore this combination is not a good predictor of edibility.

My next step is to export all single and double variable charts to Excel, to better identify the relevant characteristics.

Exporting to Excel

By exporting the charts to Excel, I was able to manually highlight the significant attributes that predict edibility.

Single Variable Set up

Exporting cap shape:

```
(Debug) In[*]:= exportCapShape = Normal[countsCapShape] // TableForm
```

```
(Debug) Out[*]//TableForm=
```

```
(x → p) → 1703
(x → e) → 1931
(b → e) → 387
(s → e) → 32
(f → e) → 1585
(f → p) → 1552
(b → p) → 48
(k → e) → 205
(k → p) → 552
(c → p) → 4
```

```
Export["file location.xls", exportCapShape, "XLS"]
```

Now, export all of the tables.

```
(Debug) In[*]:=
```

```
allCharts2 = Table[good = Table[
  mushroomCharacteristics[[n, characteristic]] → mushroomCharacteristics[[n, 1]],
  {n, 8000}][[2 ;; -1]]; countsPE = KeySort[Counts[good]];
Normal[countsPE], {characteristic, 2, 23}];
```

```
Export["file location.xls", allCharts2 // TableForm, "XLSX", "FieldSeparators" → " "]
```

And then import into Google Spreadsheets, copy and paste transposed, make columns wider, and you're good to go :)

After exporting the data itself, I labeled each of the 22 columns with the characteristic name.

Export characteristic names:

```
(Debug) In[*]:=
```

```
allCharacteristics =
Table[mushroomCharacteristics[[1, characteristic]], {characteristic, 2, 23}];
```

```
Export["file location.xls",
allCharacteristics // TableForm, "XLSX", "FieldSeparators" → " "]
```

Link to google spreadsheet, with significant characteristics highlighted:

https://docs.google.com/spreadsheets/d/15L_AmXXT386yRI4Mpti_uVwpK8akxEkC9585lMeeWJE/edit?usp=sharing

Results: Significant Characteristics

Here is a sample of the spreadsheet with the individual characteristic data.

Red highlighting is insignificant, yellow could be significant, and green is significant. Arbitrarily, for an

attribute to be highlighted as green, it has to be twice as likely to be edible than poisonous, or poisonous than edible.

bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	stalk-surface-above	stalk-surface-below
("f" -> "e") -> 1389	("a" -> "e") -> 400	("a" -> "e") -> 148	("c" -> "e") -> 2962	("b" -> "e") -> 3852	("b" -> "p") -> 1673	("e" -> "e") -> 1548	("?" -> "e") -> 654	("f" -> "e") -> 408	("f" -> "e") -> 456
("f" -> "p") -> 3235	("c" -> "p") -> 192	("a" -> "p") -> 17	("c" -> "p") -> 3747	("b" -> "p") -> 1690	("e" -> "e") -> 96	("e" -> "p") -> 1898	("?" -> "p") -> 1705	("f" -> "p") -> 144	("f" -> "p") -> 144
("t" -> "e") -> 2751	("f" -> "p") -> 2137	("f" -> "e") -> 3992	("w" -> "e") -> 1178	("n" -> "e") -> 288	("g" -> "e") -> 242	("t" -> "e") -> 2592	("b" -> "e") -> 1918	("k" -> "e") -> 132	("k" -> "e") -> 134
("t" -> "p") -> 624	("t" -> "e") -> 400	("f" -> "p") -> 3842	("w" -> "p") -> 112	("n" -> "p") -> 2169	("g" -> "p") -> 504	("t" -> "p") -> 1961	("b" -> "p") -> 1856	("k" -> "p") -> 2198	("k" -> "p") -> 2131
	("m" -> "p") -> 34				("h" -> "e") -> 204		("c" -> "e") -> 512	("s" -> "e") -> 3585	("s" -> "e") -> 3343
	("n" -> "e") -> 3340				("h" -> "p") -> 528		("c" -> "p") -> 42	("s" -> "p") -> 1509	("s" -> "p") -> 1510
	("n" -> "p") -> 120				("k" -> "e") -> 344		("e" -> "e") -> 864	("y" -> "e") -> 15	("y" -> "e") -> 207
	("p" -> "p") -> 256				("k" -> "p") -> 64		("e" -> "p") -> 256	("y" -> "p") -> 8	("y" -> "p") -> 74
	("s" -> "p") -> 563				("n" -> "e") -> 919				
	("y" -> "p") -> 557				("n" -> "p") -> 112				
					("o" -> "e") -> 52				
					("p" -> "e") -> 847				
					("p" -> "p") -> 640				
					("r" -> "p") -> 24				
					("u" -> "e") -> 444				
					("u" -> "p") -> 48				
					("w" -> "e") -> 943				
					("w" -> "p") -> 246				
					("y" -> "e") -> 49				
					("y" -> "p") -> 20				

Relative to other characteristics, bruises, odor, gill size, gill color, stalk surface-above, stalk surface-below, spore-print color, are good predictors of edibility.

The results match well with a similar study, which found that odor, spore-print-color, habitat, gill-size, and cap-color are relevant characteristics (Al-Mejibli I., and Abd D., 2017).

Double Variable Excel Characteristics

The correlation of 2 variables produces a greater variety of unique identifiers for edibility. I repeated the same process for creating the charts as with the single variable correlation. Then, I exported all 484 charts to Excel.

(Debug) In[]:=

```
storeList = {};
allCharts2Variables = Table[
  ((good = Table[mushroomCharacteristics[[n, {characteristic1, characteristic2}]] ->
    mushroomCharacteristics[[n, 1]], {n, 2, 8000}]);
  storeList = AppendTo[storeList, Normal[KeySort[Counts[good]]]];
  , {characteristic1, 2, 23}, {characteristic2, 2, 23}];
```

Export["file name.xls", storeList // TableForm, "XLSX", "FieldSeparators" -> " "]

Now export 484 section labels

(Debug) In[]:=

```
storeListLabels = {};
allCharts2Variables = Table[
  storeList = AppendTo[storeListLabels, {mushroomCharacteristics[[1, characteristic1]],
    mushroomCharacteristics[[1, characteristic2]]}];
  , {characteristic1, 2, 23}, {characteristic2, 2, 23}];
```

Export["file name.xls", storeListLabels // TableForm, "XLSX", "FieldSeparators" -> " "]

I did not analyze each of the 484 combinations, instead, I took a look at the combination of the significant single-variable characteristics found earlier.

Link to full spreadsheet: https://docs.google.com/spreadsheets/d/1k_XzSnoty2YYJPgamyNX-Hta-moemOKxMo5OMPXV5BU4/edit?usp=sharing

gill-size population	gill-size habitat	gill-color cap-shape	gill-color cap-surface	gill-color cap-color	gill-color bruises
{{"b", "a"} -> "e"} -> 384	{{"b", "d"} -> "e"} -> 1/36	{{"b", "f"} -> "p"} -> 5/3	{{"b", "s"} -> "p"} -> 838	{{"b", "e"} -> "p"} -> 835	{{"b", "f"} -> "p"} -> 16/3
{{"b", "c"} -> "e"} -> 272	{{"b", "d"} -> "p"} -> 466	{{"b", "k"} -> "p"} -> 528	{{"b", "y"} -> "p"} -> 835	{{"b", "n"} -> "p"} -> 838	{{"e", "t"} -> "e"} -> 96
{{"b", "e"} -> "p"} -> 34	{{"b", "g"} -> "e"} -> 1386	{{"b", "x"} -> "p"} -> 572	{{"e", "s"} -> "e"} -> 48	{{"e", "b"} -> "e"} -> 24	{{"g", "f"} -> "p"} -> 114
{{"b", "n"} -> "e"} -> 390	{{"b", "g"} -> "p"} -> 612	{{"e", "f"} -> "e"} -> 32	{{"e", "y"} -> "e"} -> 48	{{"e", "e"} -> "e"} -> 24	{{"g", "f"} -> "p"} -> 480
{{"b", "s"} -> "e"} -> 868	{{"b", "l"} -> "e"} -> 148	{{"e", "k"} -> "e"} -> 32	{{"g", "f"} -> "e"} -> 67	{{"e", "n"} -> "e"} -> 24	{{"g", "t"} -> "e"} -> 128
{{"b", "s"} -> "p"} -> 144	{{"b", "m"} -> "e"} -> 256	{{"e", "x"} -> "e"} -> 32	{{"g", "f"} -> "p"} -> 240	{{"e", "p"} -> "e"} -> 24	{{"g", "t"} -> "p"} -> 24
{{"b", "v"} -> "e"} -> 948	{{"b", "m"} -> "p"} -> 36	{{"g", "b"} -> "e"} -> 92	{{"g", "s"} -> "e"} -> 111	{{"g", "b"} -> "p"} -> 8	{{"h", "f"} -> "e"} -> 204
{{"b", "v"} -> "p"} -> 864	{{"b", "p"} -> "e"} -> 134	{{"g", "b"} -> "p"} -> 12	{{"g", "s"} -> "p"} -> 36	{{"g", "g"} -> "e"} -> 56	{{"h", "f"} -> "p"} -> 432
{{"b", "y"} -> "e"} -> 990	{{"b", "p"} -> "p"} -> 432	{{"g", "f"} -> "e"} -> 8	{{"g", "y"} -> "e"} -> 64	{{"g", "g"} -> "p"} -> 232	{{"h", "t"} -> "p"} -> 96
{{"b", "y"} -> "p"} -> 648	{{"b", "u"} -> "p"} -> 144	{{"g", "f"} -> "p"} -> 228	{{"g", "y"} -> "p"} -> 228	{{"g", "n"} -> "e"} -> 12	{{"k", "f"} -> "e"} -> 216
{{"n", "c"} -> "p"} -> 16	{{"b", "w"} -> "e"} -> 192	{{"g", "k"} -> "e"} -> 31	{{"h", "f"} -> "e"} -> 96	{{"g", "p"} -> "p"} -> 24	{{"k", "t"} -> "e"} -> 128
{{"n", "s"} -> "p"} -> 224	{{"n", "d"} -> "e"} -> 144	{{"g", "s"} -> "e"} -> 8	{{"h", "f"} -> "p"} -> 216	{{"g", "w"} -> "e"} -> 110	{{"k", "t"} -> "p"} -> 64
{{"n", "s"} -> "e"} -> 216	{{"n", "d"} -> "p"} -> 781	{{"g", "x"} -> "e"} -> 103	{{"h", "s"} -> "e"} -> 96	{{"g", "w"} -> "p"} -> 24	{{"n", "f"} -> "e"} -> 263
{{"n", "v"} -> "p"} -> 1929	{{"n", "g"} -> "p"} -> 128	{{"g", "x"} -> "p"} -> 264	{{"h", "s"} -> "p"} -> 96	{{"g", "y"} -> "e"} -> 64	{{"n", "f"} -> "p"} -> 48
{{"n", "y"} -> "e"} -> 72	{{"n", "l"} -> "e"} -> 48	{{"h", "f"} -> "e"} -> 102	{{"h", "y"} -> "e"} -> 12	{{"g", "y"} -> "p"} -> 216	{{"n", "t"} -> "e"} -> 656
	{{"n", "l"} -> "p"} -> 575	{{"h", "f"} -> "p"} -> 264	{{"h", "y"} -> "p"} -> 216	{{"h", "b"} -> "p"} -> 32	{{"n", "t"} -> "p"} -> 64
	{{"n", "p"} -> "p"} -> 557	{{"h", "x"} -> "e"} -> 102	{{"k", "f"} -> "e"} -> 120	{{"h", "g"} -> "e"} -> 64	{{"o", "f"} -> "e"} -> 52
	{{"n", "u"} -> "e"} -> 96	{{"h", "x"} -> "p"} -> 264	{{"k", "s"} -> "e"} -> 160	{{"h", "g"} -> "p"} -> 248	{{"p", "f"} -> "e"} -> 319
	{{"n", "u"} -> "p"} -> 128	{{"k", "b"} -> "e"} -> 64	{{"k", "s"} -> "p"} -> 32	{{"h", "n"} -> "e"} -> 64	{{"p", "f"} -> "p"} -> 480
		{{"k", "f"} -> "e"} -> 104	{{"k", "y"} -> "e"} -> 64	{{"h", "r"} -> "e"} -> 4	{{"p", "t"} -> "e"} -> 528
		{{"k", "f"} -> "p"} -> 32	{{"k", "y"} -> "p"} -> 32	{{"h", "u"} -> "e"} -> 4	{{"p", "t"} -> "p"} -> 160

The best combination was either spore-print color or gill size, or spore-print color or odor. Almost all of the combinations are unique, where the combination of two attributes are either all poisonous or all edible.

Combination of spore print color and gill size:

gill-size
spore-print-color
{{"b", "b"} -> "e"} -> 35
{{"b", "h"} -> "p"} -> 1584
{{"b", "k"} -> "e"} -> 1600
{{"b", "n"} -> "e"} -> 1636
{{"b", "o"} -> "e"} -> 39
{{"b", "r"} -> "p"} -> 72
{{"b", "w"} -> "e"} -> 504
{{"b", "w"} -> "p"} -> 34
{{"b", "y"} -> "e"} -> 38
{{"n", "h"} -> "e"} -> 48
{{"n", "k"} -> "e"} -> 48
{{"n", "k"} -> "p"} -> 224
{{"n", "n"} -> "e"} -> 96
{{"n", "n"} -> "p"} -> 224
{{"n", "u"} -> "e"} -> 48
{{"n", "w"} -> "e"} -> 48
{{"n", "w"} -> "p"} -> 1721

I will be using this chart to predict edibility given spore print and gill size. Spore print is the color of the spores found in the gills of a mushroom. A common technique to extract the color of the spores is to leave the cap of the the mushroom on a piece of paper overnight.

For example, if the gill size is "n" narrow, and spore print is "w" white, based on UCI dataset (last line in the chart) there were only 48 entries for that combination that resulted in an edible mushroom, in comparison to an overwhelming poisonous 1721 entries. Therefore, this mushroom with a narrow gill

size and white spore print would be categorized as poisonous. Similarly, mushroom entries with a broad gill size and a brown spore print ("n") were all edible per the UCI dataset.

Analysis

When identifying good predictors of edibility, I also focused on the factors that predict if a mushroom is most likely poisonous.

Because there are only two classes, "edible" and "poisonous", I could have also just focused on identifying attributes that result in edibility. If a mushroom does not contain those attributes then, then it would be classified as poisonous. Using this approach however, I would have had to identify many more attributes that uniquely indicate if a mushroom is edible. For example, only edible mushrooms have an abundant population or a sunken cap shape.

Given my goal to extract these characteristics from an image, this approach of only identifying edible characteristics would not have been time feasible, because I would have had to have created many more training image datasets to extract them. However, this is an outlet for future work: focusing only on the characteristics that identify a mushroom as edible, to see if there is a greater accuracy.







Creating training image datasets for spore print and gill size


From the previous step, I now have a way of predicting edibility given spore print and gill size.

Next, I created functions to extract the spore print and gill size by creating training image datasets.

The overall goal is to use these two *Classify* functions (machine learning functions in the Wolfram Language) to extract spore print and gill size from an image, and then use the UCI chart in the previous step to predict edibility.

Using the *Classify* function, you can label training images with the desired aspect you wish to extract, such as whether the image is a cat or dog. For example:

```
In[1]:=
c = Classify[
  {
     → "cat",
     → "cat",
     → "cat",
     → "dog",
     → "dog",
     → "dog"
  },
  FeatureExtractor → "ImageFeatures"
]

Out[1]=
ClassifierFunction[
  
  Input type: Image
  Classes: cat, dog
]
```

Classify a new image:



Out[2]= dog

Thank you the Wolfram Documentation center for providing this example (Wolfram Language & System).

I will repeat this process, but instead of labelling the images with whether they're a cat or dog, I'll create two training sets: one with mushrooms labeled with spore print, and one labeled with gill size. With the training sets I'll be able to create functions that extract these two characteristics, and then use the UCI dataset to predict edibility.

From an earlier section, my image dataset consists of 98 edible mushrooms, 33 inedible mushrooms, and 35 lethally poisonous mushrooms. Each individual mushroom contains approximately 10 - 15 images, creating an image dataset of approximately 3000 images.

Before I can use the images however, I need to webscrape further data from WildFoodUK regarding spore print and gill size, so that I can label the images with the correct characteristic.

Spore print

For each individual mushroom on WildFoodUK, there is a description of the spore print. For the edible, nonedible, and lethally poisonous mushrooms, I webscraped the first 6 words describing spore print, and then manually converted into the categorical classification as seen in the UCI dataset.

Spore prints for the 98 edible mushrooms:

```
(Debug) In[*]:= listediblesporeprintcolors =
Table[textMaybe = (Import[evenbetterhyperlinks[[n]], "Plaintext"]);
Which[StringContainsQ[textMaybe, "Spore Print"], StringExtract[
StringTake[textMaybe, {Last@Flatten@StringPosition[textMaybe, "Spore Print"],
Last@Flatten@StringPosition[textMaybe, "Spore Print"] + 30}], 2 ;; 8],
StringContainsQ[textMaybe, "Spore"], StringExtract[StringTake[textMaybe,
{Last@Flatten@StringPosition[textMaybe, "Spore"], Last@Flatten@StringPosition[
textMaybe, "Spore"] + 30}], 2 ;; 8]], {n, 1, Length@evenbetterhyperlinks}]

(Debug) Out[*]:= {{ "Dark", "purple/brown.", "Ellipsoi"}, {"Purple/chocolate", "brown.", "Ell"},
{"Brown.", "Subglobose.", ""}, {"Chocolate", "brown.", "Ellipsoid."},
{"Chocolate", "brown.", "Ovoid.", "You"}, {"Brown.", "Ellipsoid.", ""},
{"Purple/brown.", "Ellipsoid.", ""}, {"Brown.", "Ovoid.", ""},
{"Purple/brown.", "Ellipsoid.", ""}, {"White,", "ellipsoid.", "These", "ar"},
{"White,", "subglobose.", ""}, {"White.", "Ellipsoid.", "You", "shoul"},
{"White.", "Spherical.", ""}, {"White.", "Ovoid.", ""}, {"White.", "Spherical.", ""},
{"White", "to", "pale", "cream.", "Ellips"}, {"White.", "Ellipsoid", "and", "smooth"},
```

```

{"White.", "Sausage", "shaped.", ""}, {"Olivaceous/brown.", "Subfusifo"},
{"Cinnamon.", "Subfusiform", "elli"}, {"Pale", "straw", "coloured.", "Ellips"},
{"Olive", "brown.", "Subfusiform.", ""}, {"Olive-brown.", "Subfusiform", "to"},
{"Green/brown.", "Subfusiform.", ""}, {"Olivaceous", "brown.", "Ellipsoid"},
{"Brown.", "Subfusiform.", ""}, {"Brown.", "Subfusiform.", ""},
{"Olive-brown.", "Subfusiform", "to"}, {"Olive-brown.", "Subfusiform", "to"},
{"Olive-brown.", "Subfusiform.", ""}, {"Oche-sienna", "coloured.", "Subfu"},
{"Brown.", "Subfusiform.", "The", "ima"}, {"Olive", "green", "to", "brown.", "Subfu"},
{"Olive/brown.", "Subfusiform.", ""}, {"Green/brown.", "Subfusiform.", ""},
{"Olive", "Brown.", "Ellipsoid.", ""}, {"Olive", "brown.", "Subfusiform.", ""},
{"White.", "Ellipsoid.", ""}, {"Yellow/brown.", "Spherical", "wit"},
{"Ochraceous.", "Ellipsoid.", ""}, {"Off-white.", "Subglobose.", ""},
{"White.", "Ellipsoid.", ""}, {"White.", "Subglobose.", "You", "shou"},
{"Pink.", "Ellipsoid.", ""}, {"Date", "brown.", "Mitriiform.", ""},
{"Blackish", "brown.", "Ellipsoid.", ""}, {"Cream", "to", "salmon", "to", "yellow.", ""},
{"Brown", "Taste", ""}, {"Pink/pale", "ochre.", "Ovate.", ""},
{"Slightly", "off", "white.", "Ellipso"}, {"White.", "Ellipsoid", "cylindric"},
{"White.", "broadly", "ellipsoid.", "Y"}, {"White.", "Broadly", "ellipsoid", "to"},
{"White.", "Ellipsoid.", ""}, {"Off", "white", "to", "cream.", "Ellipso"},
{"White.", "Ellipsoid.", "You", "shoul"}, {"White.", "Ellipsoid.", ""},
{"White.", "Ellipsoid.", ""}, {"White.", "Ellipsoid.", ""},
{"White", "to", "pale", "yellow.", "Ellip"}, {"White.", "ellipsoid.", ""},
{"White.", "Ellipsoid.", ""}, {"White.", "Ellipsoid.", ""},
{"White.", "Globose", "with", "spines."}, {"White.", "cream/yellow.", "Globos"},
{"Cream", "coloured.", "Subglobose."}, {"Pale", "ochre", "to", "pale", "salmon", "c"},
{"Pale", "ochre.", "Ellipsoid", "with"}, {"White.", "Ellipsoid", "to", "broadly"},
{"White-cream.", "Subglobose.", "Yo"}, {"Off-white", "to", "pale", "pink.", "Ell"},
{"Off", "white", "to", "Pale", "pink.", "Ell"}, {"Olive/brown.", ""},
{"Olive/brown.", "Globose", "with", "f"}, {"The", "spores", "start", "inside", "the"},
{"White/off", "white.", "Ellipsoid."}, {"White.", "Ovoid", "dextrinoid.", ""},
{"White.", "Ellipsoid.", ""}, {"White.", "Ovoid", "ellipsoid.", ""},
{"Pa", "le", "cream", "to", "yellow.", "Elli"}, {"Cream.", "Ellipsoid.", ""},
{"Cream", "to", "pale", "yellow.", ""}, {"Pale", "cream", "to", "yellow.", "Ellip"},
{"White.", "Has", "many", "spores", "and", ""}, {"Off", "white.", "Broadly", "ellipsoi"},
{"Pale", "yellow.", "Oblong.", "As", "the"}, {"Lilac.", "Cylindrical.", ""},
{"White.", "Ovoid", "ellipsoid.", "Yo"}, {"White", "to", "off", "white", "to", "pale"},
{"Off", "white.", "Subglobose", "to", "gl"}, {"Pale", "ochre.", "Ellipsoid", "with", ""},
{"White", "to", "pale", "cream.", "Broadl"}, {"White/cream.", "Broadly", "ovoid."},
{"Pale", "cream.", "Subglobose.", ""}, {"White/off", "white.", "Ellipsoid,"},
{"White.", "Ellipsiod", "to", "cylindr"}, {"Colourless", "unless", "in", "large", ""},
{"Blackish", "brown", "to", "brown.", "Ov"}

```

I created a text file to convert these descriptions consistent with the the UCI categorical entries of “white”, “brown”, “chocolate”, “buff”, “orange”, “green” and “purple”.

As you can tell, there are many discrepancies between these two datasets. In what category should “Dark chocolate purple” go in? Because only 40 out of the 8000 entries in the UCI characteristic dataset

were labeled as purple, “Dark chocolate purple” did not go into the “purple” category, but instead in the “chocolate” category (approximately 1000 of those entries), with a chocolate spore print being a much more common classification. Similarly, “Olive brown” went in the “brown” category, instead of the less common “green” category.

New and improved spore prints:

```
ediblePrints = Import["text file", "List"]
```

```
(Debug) Out[*]:= {Chocolate, Chocolate, Brown, Chocolate, Chocolate, Brown, Chocolate, Brown,
  Chocolate, White, White, White, White, White, White, White, White, White, White, Brown,
  Orange, Buff, Brown, Brown, Green, Brown, Brown, Brown, Brown, Brown, Buff,
  Brown, Green, Brown, Green, Brown, Brown, White, Buff, Purple, White, White,
  White, Pink, Brown, Black, Pink, Brown, Purple, White, White, White, White, White,
  White, White, White, White, White, White, Yellow, White, White, White, White, White,
  White, Purple, Purple, White, White, White, White, Brown, Brown, Brown, White,
  White, White, White, Yellow, White, Yellow, Yellow, White, White, Yellow, Purple,
  White, White, White, Purple, White, White, White, White, White, Clear, Black}
```

I repeated this process for inedible and lethally poisonous mushrooms.

Inedible spore prints:

```
(Debug) Out[*]:= {White, White, White, White, Brown, Brown, Pink, White, White, Black, White,
  Orange, White, Black, White, Brown, White, White, White, White, Black, White,
  Clear, Yellow, Brown, Brown, White, Purple, Green, White, Pink, White, White}
```

Lethally poisonous spore prints:

```
(Debug) Out[*]:= {Brown, Chocolate, White, White, White, White, White, White, White, White, Brown, Black, White,
  Brown, Clear, White, White, White, White, Chocolate, Brown, Brown, Brown, White, Yellow,
  White, White, White, White, Brown, White, White, White, Brown, Chocolate, White}
```

After webscraping the spore print, I assigned it to the images it corresponds with. ie edible mushroom #1 images -> edible mushroom #1 spore print.

```
(Debug) In[*]:= edibleSporeTrainingData =
  Table[ (Table[ (edibleMushroomsBetter[ [each]] ) [ [n]] → ediblePrints[ [each]] ,
    {n, 1, Length@edibleMushroomsBetter[ [each]] } ) ,
    {each, 1, Length@edibleMushroomsBetter} ] ] ;
```

Example labelling for edible mushroom #2:

```
(Debug) In[*]:= edibleSporeTrainingData[ [2] ]
```



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



→ "Chocolate",



```
(Debug) In[ ]:= inedibleSporeTrainingData =
  Table[ (Table[ (inedibleMushroomsBetter[[each]])[[n]] → inediblePrints[[each]],
    {n, 1, Length@inedibleMushroomsBetter[[each]]}),
    {each, 1, Length@inedibleMushroomsBetter}];

inedible mushroom #7
```

```
(Debug) In[ ]:= inedibleSporeTrainingData[[7]]
```





I combined all the training images to create a classify function to extract spore print.

```
(Debug) In[ ]:= sporePrintTrainingData = Flatten[Join[edibleSporeTrainingData,
               inedibleSporeTrainingData, lethallyinedibleSporeTrainingData]];
```

Took 2 hours to process:

```
extractSporePrint = Classify[sporePrintTrainingData]
```

```
(Debug) Out[ ]:= ClassifierFunction [  Input type: Image  
Number of classes: 12 ]
```

```
(Debug) In[ ]:= extractSporePrint [  ]
```

```
(Debug) Out[ ]:= Orange
```

```
(Debug) In[ ]:= extractSporePrint [  ]
```

```
(Debug) Out[ ]:= Brown
```

```
(Debug) In[ ]:= extractSporePrint [  ]
```

```
(Debug) Out[ ]:= White
```

So far, the functions accurately extracts spore print!

Gill Size

Per the UCI dataset, the two classifications for gill size are “broad” and “narrow”. I was unsure if this was the horizontal or vertical width of the gills - there wasn’t documentation for it. Because another characteristic was gill spacing, I assumed this to be the vertical width.

For the WildFoodUK mushrooms, there was not specifically a category for gill size, but instead cap size. I assumed that if the cap size is “broad”, than the gill size would be broad as well, and if the cap size was narrow, the gill size would be narrow as well. Of course, this is not always the case.

Cap size was listed numerically in cm for each mushroom entry on WildFoodUK. To categorize it into “broad” and “narrow”, I took the average of the cap sizes for all the mushrooms (12 cm, 5 inches). Then, if the value was less, the gill size was labeled as “narrow”, if it was greater the gill size was labeled as “broad”. Sometimes, the gill size varied drastically, and for those entries I just removed the data.

Extracting all the cap sizes for lethal mushrooms :

```
(Debug) In[ ]:= listlethallycapsizes =
  Table[textMaybe = (Import[goodhyperlinksLethallyPoisonous[[n]], "Plaintext"]);
    StringExtract[StringTake[textMaybe,
      {Last@Flatten@StringPosition[textMaybe, "Average Cap width"],
        Last@Flatten@StringPosition[textMaybe, "Average Cap width"] + 30}], 3],
    {n, 1, Length@goodhyperlinksLethallyPoisonous}]
(Debug) Out[ ]:= {10, 16, 7, 25, 10, 12, 12, 4, 20, 8, 5, 13, 4, 4-14, 3-8,
  6, 7, 8, 7, 6, 4, 6, 10, 12, 5, 4, 5, 8, 20, 10, 10, 10, 12, 8, 6}
```

Removing ones with too large of a range :

```
(Debug) In[ ]:= listlethallycapsizes = ReplacePart[listlethallycapsizes, {14 -> Nothing, 15 -> Nothing}]
(Debug) Out[ ]:= {10, 16, 7, 25, 10, 12, 12, 4, 20, 8, 5, 13, 4, 6,
  7, 8, 7, 6, 4, 6, 10, 12, 5, 4, 5, 8, 20, 10, 10, 10, 12, 8, 6}
```

Then repeat for inedible and edible mushrooms.

Combining all the cap sizes into one large list:

```
(Debug) In[ ]:= allcapsizes = ToExpression@
  Flatten[Join[{listediblecapsizes, listinediblecapsizes, listlethallycapsizes}]]
(Debug) Out[ ]:= {25, 20, 15, 20, 10, 15, 10, 10, 10, 10, 10, 9, 15, 15, 5, 20, 8, 10, 12, 8, 15, 10, 15, 20,
  20, 16, 10, 15, 8, 12, 6, 20, 10, 15, 15, 80, 10, 5, 60, 20, 10, 4, 15, 5, 7, 20, 10, 30,
  18, 4, 10, 4, 3, 5, 4, 4, 3, 8, 8, 6, 6, 9, 10, 8, 45, 10, 12, 12, 4, 5, 14, 12, 30, 5, 80,
  15, 5, 3, 15, 8, 5, 5, 15, 7, 5, 10, 10, 15, 10, 10, 13, 6, 25, 8, 12, 10, 15, 20, 10, 15,
  3, 0.5, 12, 6, 6, 3, 4.5, 4, 5, 10, 20, 4, 25, 15, 4, 1, 4, 7, 6, 10, 10, 16, 7, 25, 10, 12,
  12, 4, 20, 8, 5, 13, 4, 6, 7, 8, 7, 6, 4, 6, 10, 12, 5, 4, 5, 8, 20, 10, 10, 10, 12, 8, 6}
```

Taking average:

```
(Debug) In[ ]:= Mean[allcapsizes]
(Debug) Out[ ]:= 11.8247
```

The average of all cap sizes is approximately 12cm. Therefore, if a cap size is less than 12 cm, it will be classified as “narrow”, and if it is greater than 12 cm, it will be classified as “broad”.

If the mushroom has a variable cap width, such as 4 - 14 cm, then that entry was classified as “clear”, and was later deleted from the training dataset.

Lethal poisonous mushroom gill size classification:

```
(Debug) In[ ]:= lethallyPoisonousGillSize =
  Table[Which[ToExpression[listlethallycapsizes][[n]] > 12, "broad",
    ToExpression[listlethallycapsizes][[n]] > 0, "narrow", True, "clear"],
    {n, Length@ToExpression[listlethallycapsizes]}]
(Debug) Out[ ]:= {narrow, broad, narrow, broad, narrow, narrow, narrow, narrow,
  broad, narrow, narrow, broad, narrow, clear, clear, narrow, narrow,
  narrow, narrow, narrow, narrow, narrow, narrow, narrow, narrow, narrow,
  narrow, narrow, broad, narrow, narrow, narrow, narrow, narrow, narrow}
```

Inedible gill size classification:

```
(Debug) In[*]:= inedibleGillSize = Table[Which[ToExpression[listinediblecapsizes][[n]] > 12, "broad",
      ToExpression[listinediblecapsizes][[n]] > 0, "narrow", True, "clear"],
      {n, Length@ToExpression[listinediblecapsizes]}]

(Debug) Out[*]:= {narrow, narrow, clear, clear, broad, broad, narrow, clear, broad, narrow, narrow,
      narrow, clear, narrow, clear, narrow, narrow, narrow, narrow, narrow, narrow, broad,
      narrow, broad, clear, broad, narrow, narrow, narrow, narrow, narrow, clear, narrow}
```

Edible gill size classification:

```
(Debug) In[*]:= edibleGillSize = Table[Which[ToExpression[listediblecapsizes][[n]] > 12,
      "broad", ToExpression[listediblecapsizes][[n]] > 0, "narrow", True, "clear"],
      {n, Length@ToExpression[listediblecapsizes]}]

(Debug) Out[*]:= {broad, broad, broad, broad, narrow, broad, narrow, narrow, narrow, narrow, narrow, narrow,
      narrow, broad, clear, broad, clear, narrow, broad, narrow, narrow, narrow, narrow,
      broad, narrow, broad, broad, broad, broad, narrow, broad, narrow, narrow, narrow,
      broad, narrow, broad, broad, broad, narrow, narrow, broad, broad, narrow, narrow,
      broad, narrow, narrow, broad, narrow, broad, clear, broad, narrow, narrow, narrow,
      narrow, narrow, narrow, narrow, narrow, narrow, narrow, narrow, narrow, narrow,
      narrow, narrow, broad, narrow, narrow, narrow, narrow, narrow, broad, narrow,
      broad, narrow, broad, broad, narrow, narrow, broad, narrow, narrow, narrow, broad,
      narrow, narrow, narrow, narrow, broad, narrow, narrow, broad, narrow, broad, narrow}
```

Similar to gill size, I now associated the mushroom images to their gill size. ie lethal mushroom #1 -> lethal mushroom #1 gill size.

Creating the training datasets to classify gill size:

Lethal mushrooms:

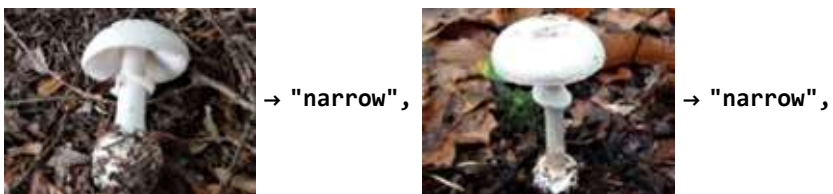
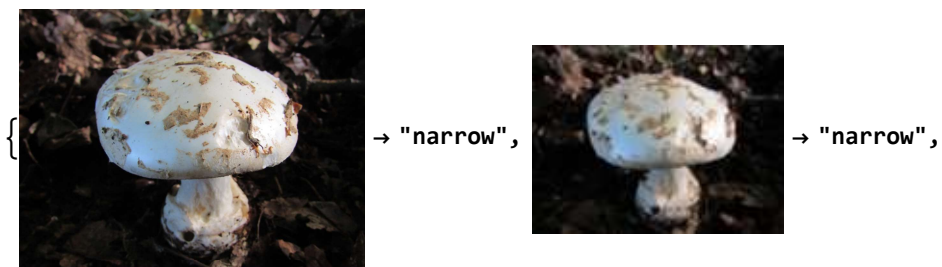
```
(Debug) In[*]:= lethallyinedibleGillsTrainingData =
      Table[(Table[(lethallyPoisonousMushrooms[[each]])[[n]] → lethallyPoisonousGillSize[[
      each]], {n, 1, Length@lethallyPoisonousMushrooms[[each]]}),
      {each, 1, Length@lethallyPoisonousMushrooms}];
```

Inedible mushrooms:

```
(Debug) In[*]:= inedibleGillsTrainingData =
      Table[(Table[(inedibleMushroomsBetter[[each]])[[n]] → inedibleGillSize[[each]],
      {n, 1, Length@inedibleMushroomsBetter[[each]]}),
      {each, 1, Length@inedibleMushroomsBetter}];
```

Example gill classification for inedible mushroom #1:

```
(Debug) In[*]:= inedibleGillsTrainingData[[1]]
```



And last and not least gill size classification for edible mushrooms:

```
(Debug) In[ ]:= edibleGillsTrainingData =
  Table[ (Table[ (edibleMushroomsBetter[ [each]] ) [ [n]] → edibleGillSize[ [each]] ,
    {n, 1, Length@edibleMushroomsBetter[ [each]] } ) ,
    {each, 1, Length@edibleMushroomsBetter} ] ;
```

Sanity check, to makes sure there is a proper amount of training data:

```
(Debug) In[ ]:= Length@inedibleGillsTrainingData
```

```
(Debug) Out[ ]:= 33
```

```
(Debug) In[ ]:= Length@lethallyinedibleGillsTrainingData
```

```
(Debug) Out[ ]:= 35
```

```
(Debug) In[ ]:= Length@edibleGillsTrainingData
```

```
(Debug) Out[ ]:= 98
```


There is indeed training data for all the 98, 33, and 35 edible, inedible, and lethally poisonous mushrooms, respectively.

Combining the edible, inedible, and lethally poisonous gill size training data, to create a function to classify gill size:

```
(Debug) In[ ]:= allGillsTrainingData = Flatten[Join[edibleGillsTrainingData,
  inedibleGillsTrainingData, lethallyinedibleGillsTrainingData]] ;
```


This classify function only took 30 minutes to run in comparison to the 2 hours for spore print, probably because there are only 2 categorizations (broad and narrow).

```
extractGillSize = Classify[allGillsTrainingData]
```

```
(Debug) Out[ ]:= ClassifierFunction[  Input type: Image  
Classes: broad, narrow ]
```

```
(Debug) In[ ]:= extractGillSize [  ]
```

```
(Debug) Out[ ]:= narrow
```

Putting it all together

I have the counts from the UCI dataset to identify edibility of a mushroom given the spore print and gill size. And I have two Classify functions to extract spore print and gill size.

Refresher for the UCI classification chart:

gill-size	
spore-print-color	
{{"b", "b"} -> "e"} -> 35	
{{"b", "h"} -> "p"} -> 1584	
{{"b", "k"} -> "e"} -> 1600	
{{"b", "n"} -> "e"} -> 1636	
{{"b", "o"} -> "e"} -> 39	
{{"b", "r"} -> "p"} -> 72	
{{"b", "w"} -> "e"} -> 504	
{{"b", "w"} -> "p"} -> 34	
{{"b", "y"} -> "e"} -> 38	
{{"n", "h"} -> "e"} -> 48	
{{"n", "k"} -> "e"} -> 48	
{{"n", "k"} -> "p"} -> 224	
{{"n", "n"} -> "e"} -> 96	
{{"n", "n"} -> "p"} -> 224	
{{"n", "u"} -> "e"} -> 48	
{{"n", "w"} -> "e"} -> 48	
{{"n", "w"} -> "p"} -> 1721	

Now I need to be able to automatically, given gill size and spore print, predict edibility using that chart.

Importing raw data again:

```
mushroomCharacteristics = Import["csv file"];
```

```
(Debug) In[ ]:= twoVariables = Table[ mushroomCharacteristics[[n, {9, 21}]] ->  
mushroomCharacteristics[[n, 1]], {n, 2, 8000}];
```

```
(Debug) In[ ]:= KeySort[Counts[twoVariables]]
```

```
(Debug) Out[ ]:= <| ({b, b} → e) → 35, ({b, h} → p) → 1584, ({b, k} → e) → 1600,
  ({b, n} → e) → 1636, ({b, o} → e) → 39, ({b, r} → p) → 72,
  ({b, w} → e) → 504, ({b, w} → p) → 34, ({b, y} → e) → 38, ({n, h} → e) → 48,
  ({n, k} → e) → 48, ({n, k} → p) → 224, ({n, n} → e) → 96, ({n, n} → p) → 224,
  ({n, u} → e) → 48, ({n, w} → e) → 48, ({n, w} → p) → 1721 |>
```

```
(Debug) In[ ]:= sporePrintGillSize = Association@Keys@KeySort[Counts[twoVariables]]
```

Some of the classifications were incorrectly labeled, just because it took the first value of the key. For example, {"b", "w"} was first labeled as poisonous instead of the much more common edible.

```
(Debug) In[ ]:= sporePrintGillSize = <| {"b", "b"} → "e", {"b", "h"} → "p", {"b", "k"} → "e",
  {"b", "n"} → "e", {"b", "o"} → "e", {"b", "r"} → "p", {"b", "w"} → "e", {"b", "y"} → "e",
  {"n", "h"} → "e", {"n", "k"} → "p", {"n", "n"} → "p", {"n", "u"} → "e", {"n", "w"} → "p" |>
```

```
(Debug) Out[ ]:= <| {b, b} → e, {b, h} → p, {b, k} → e, {b, n} → e, {b, o} → e, {b, r} → p,
  {b, w} → e, {b, y} → e, {n, h} → e, {n, k} → p, {n, n} → p, {n, u} → e, {n, w} → p |>
```

Classify edibility:


```
(Debug) In[ ]:= predict[gillsize_, sporeprint_] := First@
  Values@KeyTake[sporePrintGillSize, {{ToString[gillsize], ToString[sporeprint]}}]
```

Example, using only the UCI dataset, predict edibility of a mushroom with a broad gill size ("b") and chocolate spore print ("h"):

```
(Debug) In[ ]:= predict["b", "h"]
{"p"}
```

Now moving onto the image section.

Classifier functions from previous section:

```
(Debug) In[ ]:= extractGillSize = ClassifierFunction[ Input type: Image
Classes: broad, narrow
Method: LogisticRegression
Number of training examples: 2994];
```

```
(Debug) In[ ]:= extractSporePrint = ClassifierFunction[ Input type: Image
Number of classes: 12
Method: LogisticRegression
Number of training examples: 3198];
```

When creating the training datasets, it was easier, organizationally, to label the spore print as "white" or "chocolate", and gill size "broad" or "narrow", instead of the UCI labels of "w" or "h", or "b" and "n". Converting between the labels:

```
(Debug) In[ ]:= replaceProperSporePrint[sporeprint_] :=  
Which[sporeprint == "Chocolate", "h", sporeprint == "White", "w", sporeprint == "Brown",  
"n", sporeprint == "Buff", "b", sporeprint == "Green", "r", sporeprint == "Pink",  
"o", sporeprint == "Purple", "u", sporeprint == "Yellow", "y"]
```

```
(Debug) In[ ]:= replaceProperGillSize[gillsize_] :=  
Which[gillsize == "narrow", "n", gillsize == "broad", "b"]
```

Here is an example identifying the edibility of the edible king boletas mushroom :



```
(Debug) In[ ]:= extractSporePrint[
```

```
(Debug) Out[ ]:= Brown
```

```
(Debug) In[ ]:= extractGillSize[
```

```
(Debug) Out[ ]:= broad
```

```
replaceProperSporePrint["Brown"]  
replaceProperGillSize["broad"]
```

```
(Debug) In[ ]:= predict["b", "n"]
```

```
{e}
```

Correctly predicts that the king boletas mushroom is edible, with a broad gill size and brown spore print!

Accuracy

To predict accuracy, I took 2 random images from each edible, inedible, and lethally poisonous mushroom. As a result, I had 196 edible mushrooms images, 66 inedible images, and 70 lethally poisonous mushroom images.

Random Training Images

```
(Debug) In[ ]:= randomEdible = Flatten[#[[3 ;; 4]] & /@ edibleMushroomsBetter];
```

```
(Debug) In[ ]:= randomInedible = Flatten[#[[3 ;; 4]] & /@ inedibleMushroomsBetter];
```

```
(Debug) In[ ]:= randomLethal = Flatten[#[[3 ;; 4]] & /@ lethallyPoisonousMushrooms];
```

Running predictions on random images using the two classifier functions that extract spore print and gill size from an image, and then using the UCI dataset chart to predict edibility.

```
(Debug) In[ ]:= ediblePredictionsBruh =
  Table[gillSize1 = replaceProperGillSize[extractGillSize[randomEdible[[n]]]];
        sporePrint1 = replaceProperSporePrint[extractSporePrint[randomEdible[[n]]]];
        predict[gillSize1, sporePrint1], {n, Length@randomEdible}]

(Debug) Out[ ]:= {e, e, e, e, p, e, p, e, e, e, e, e, p, p, p, e, e, p, p, p, p, p, p, p, e, e, p, p, p, p, e,
  e, p, p, p, e, e, First[{}], e, p, p, p, p, p, First[{}], First[{}], p, e, e, e, e, e,
  e, e, e, e, p, p, First[{}], First[{}], p, p, e, First[{}], p, e, p, e, p, p, p, e, p,
  e, e, e, e, p, p, p, e, e, e, e, First[{}], p, p, p, First[{}], First[{}], First[{}],
  First[{}], p, p, e, e, p, p, e, e, e, p, e, e, p, p, p, p, p, p, p, p, p, p, p, p,
  First[{}], p, p, p, p, p, p, p, p, p, p, p, p, e, First[{}], e, e, p, e, p, p, p, p, p,
  p, p, p, p, p, e, e, e, p, e, e, p, e, e, p, First[{}], First[{}], p, p, First[{}],
  First[{}], e, e, p, p, p, p, First[{}], e, First[{}], First[{}], p, p, p, p, p, p,
  p, p, p, e, p, p, p, p, p, e, p, p, First[{}], First[{}], First[{}], First[{}]}
```

Some of the combinations, such as a broad gill size and green spore print, were not in the UCI dataset, and that is why there is an error of “First[{}]” for some mushrooms.

```
(Debug) In[ ]:= Count[ediblePredictionsBruh, "e"]
```

```
(Debug) Out[ ]:= 64
```

```
(Debug) In[ ]:= Count[ediblePredictionsBruh, "p"]
```

```
(Debug) Out[ ]:= 108
```

There is a 37% accuracy in using gill size and spore print to predict edible mushrooms.

```
(Debug) In[ ]:= (64) / (64 + 108) // N
```

```
(Debug) Out[ ]:= 0.372093
```

This is quite surprisingly low! One reason for this is that it is hard to distinguish between a chocolate and brown spore print. Personally, I believe spore print should be measure in RGB. Most of the edible mushrooms have a correctly classified broad gill size. However, a chocolate spore print paired with a broad gill size, in contrast to a brown spore print, results in a poisonous prediction. A possibility is that I incorrectly interpreted some of the WildFoodUK spore print descriptions as “chocolate” instead of “brown”, resulting in a higher percentage of images being incorrectly classified with the poisonous indicator.

Another possible explanation is that the image dataset that I’m using, WildFoodUK, does not contain that many entities for mushrooms from the Agarics family. Instead, it documents the mushrooms found in the UK. And the UCI dataset entries are all from the Agaricus family. This means that edible mushrooms in the UK do not highly resemble Agaricus mushrooms.

```
(Debug) In[*]:= inediblePredictionsBruh =
  Table[gillSize1 = replaceProperGillSize[extractGillSize[randomInedible[[n]]]];
    sporePrint1 = replaceProperSporePrint[extractSporePrint[randomInedible[[n]]]];
    predict[gillSize1, sporePrint1], {n, Length@randomInedible}]

(Debug) Out[*]:= {p, p, p, e, e, e, p, p, e, e, e, e, First[{}], p, p, e, p, p, First[{}], p, p, First[{}],
  First[{}], e, e, p, p, e, e, p, p, p, p, p, p, p, p, First[{}], p, e, e, p, First[{}],
  e, e, p, p, p, e, p, p, e, e, First[{}], First[{}], p, p, First[{}], First[{}], e, e, p, p}
```

```
(Debug) In[*]:= Count[inediblePredictionsBruh, "e"]
```

```
(Debug) Out[*]:= 22
```

```
(Debug) In[*]:= Count[inediblePredictionsBruh, "p"]
```

```
(Debug) Out[*]:= 34
```

70% accuracy for inedible mushrooms. Decent.

```
(Debug) In[*]:= 39 / (17 + 39) // N
```

```
(Debug) Out[*]:= 0.696429
```

```
(Debug) In[*]:= lethalPredictionsBruh =
  Table[gillSize1 = replaceProperGillSize[extractGillSize[randomLethal[[n]]]];
    sporePrint1 = replaceProperSporePrint[extractSporePrint[randomLethal[[n]]]];
    predict[gillSize1, sporePrint1], {n, Length@randomLethal}]
```

```
(Debug) Out[*]:= {p, p, e, e, p, p, p, p, p, p, p, p, e, p, p, p, p, p, e, First[{}], p,
  e, e, p, p, First[{}], First[{}], p, p, p, p, p, p, p, p, p, p, p, p, p, p,
  p, e, e, p, p, p, p, p, p, p, p, p, p, p, e, p, p, p, p, p, p, e, p, p, p, p, p}
```

```
(Debug) In[*]:= Count[lethalPredictionsBruh, "e"]
```

```
Count[lethalPredictionsBruh, "p"]
```

```
(Debug) Out[*]:= 10
```

```
(Debug) Out[*]:= 57
```

And a stunning 87% accuracy for predicting lethally poisonous mushrooms!

```
(Debug) In[*]:= 57 / (57 + 10) // N
```

```
(Debug) Out[*]:= 0.850746
```

After creating the training datasets, I noticed that a large amount of lethally poisonous mushrooms have a narrow gill size and white spore print color. This high accuracy is the result then, of the two classifier functions correctly and with a high accuracy identifying the white spore print and narrow gill size! Another explanation for this accuracy is that a much higher proportion of lethally poisonous mushrooms documented by WildFooUK belong to the Agaricus family, which the UCI dataset documents.

Some More Analysis

In the training datasets, inedible and lethally poisonous mushrooms were simply grouped together in the “poisonous” category. A 70% accuracy for inedible mushrooms signifies that 70% do have the spore print and gill size of poisonous mushrooms, while 30% have the spore print and gill size of edible mushrooms. Perhaps a different combination of two characteristics, such as spore print and odor, would result in a greater accuracy for inedible mushrooms - an outlet for future work.

Again, a possibility for the low accuracy of 37% for classifying may be a result of me incorrectly misinterpreting the the spore print listed on the WildFoodUK documentation as “chocolate” instead of “brown” when creating the training datasets. It’s difficult to tell the difference between a brown (edible) and chocolate (poisonous) spore print from a description.

And the solid 85% accuracy for lethally poisonous mushrooms is probably due to the fact that a white spore print and narrow gill size are easier to identify from an image.

But hey! It’s better to have a greater false negative than a false positive (it’s better to not eat an edible mushroom, than to eat a lethally poisonous mushroom).

Limitations

The image dataset from WildFoodUK does not primarily contain mushrooms from the *Agaricus* family, which the UCI dataset is based of. These regional differences may have lead to some discrepancies classifying mushrooms because mushrooms from northern Europe may have other characteristics that better determine edibility. However, WildFoodUK was the only dataset I found that allowed me to webscrape info from hundreds of websites. The layout of it was pertinent for collecting data for spore print and gill size, and as a result for the creation of the training datasets to extract features from images.

Another limitation was that gill size was not directly a part of the descriptions on WildFoodUK. To determine gill size, I used cap size, assuming that if the cap is narrow, than the gills would be narrow as well, and vice versa.

It was difficult to group some spore print descriptions from WildFoodUK to match with the categorical entries of the UCI dataset. Some example descriptions were “Deep chocolate purple”, and “Olive brown”, both of which fit into multiple categories. A possible solution would be to measure spore print in RGB intervals.

Conclusion and Future Work

I learned a lot about using the Wolfram Language to process data, and structuring my code neatly to elegantly analyze the datasets.

Using spore print and gill size, there is a 37% for predicting edible mushrooms, 70% for predicting inedible mushrooms, and 85% accuracy for lethally poisonous mushrooms. That means there is total stunning accuracy of 64%. There is indeed a higher overall accuracy of using a combination of multiple datasets to predict edibility, rather than using one dataset (compared to the project Deep Shrooms which utilized a pure image dataset with a resulting 55% accuracy (Koivisto et al.)).

The main cause for the 37% accuracy for edible mushrooms is most likely incorrectly labeling some spore prints as “chocolate” instead of “brown” when creating the training datasets. On the other hand, the 85% accuracy for lethally poisonous mushrooms indicates that they are easier to spot from an image, with the most common lethally poisonous mushroom having a narrow gill size and a white spore print.

Future work includes on determining whether the combination of spore print and odor, or gill size or odor, are better predictors of edibility. WildFookUK does include descriptions for odor! However, many of the odors are described to be to as “mushroom” smell, which is not a category in the UCI dataset. If a mushroom smells like a mushroom, however, it would probably go into the UCI dataset category of “none”.

Moreover, user input could include the season to filter similar-looking mushrooms that are found in different seasons.

Furthermore, it'd be great if both the image and characteristics dataset contained mushrooms purely from the *Agaricus* family, to limit error. A possibility is isolating, from WildFoodUK, all the mushrooms which scientific names start with “*Agaricus*”. Last time I check though, there were only 13 mushrooms from the *Agaricus* family on WildFookUK, and I am not sure if that would be enough training data. But it's worth a try!

The goal for this project was to provide a framework for utilizing and connecting multiple datasets. If in the future, if more datasets documenting mushrooms characteristics and images become available (especially if they document the same family of mushrooms), a similar method to this paper's can be used to optimize the edibility prediction.

Have fun mushroom hunters! Even with the stunning 64% accuracy, it's still interesting to see what type of gill size and spore print the image functions extract :)

References

Al-Mejibli I., and Abd D. (2017). Mushroom Diagnosis Assistance System Based On Machine Learning by Using Mobile Devices. Journal of AL-Quaisiyah for computer science and mathematics. <http://www.qu.edu.iq/journalcm/index.php/journalcm/article/view/319/289>

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [<http://archive.ics.uci.edu/ml>]. Irvine, CA:

University of California, School of Information and Computer Science.

Koivisto, T., et al. (2017). Deep Shrooms: classifying mushroom images. Github. <https://tuomoniemi-nen.github.io/deep-shrooms/>

Shields, T. (2021). The best apps for mushroom identification (and why a book is better). FreshCap Mushrooms. <https://learn.freshcap.com/tips/mushroom-identification-app/>

Wolfram Language & System. FeatureExtractor. Documentation Center. <https://reference.wolfram.com/language/ref/FeatureExtractor.html>

