Data Analysis and Regression

Dr. Thomas Robey

October 22 & 23, 2022

Data Analysis

An **observation** or **data point** is a single number (**scalar**) or a vector of numbers. An example of a vector of numbers is the x, y, and z coordinates in a three dimensional coordinate system. If the data point is a measurement then there is **measurement error** in determining the data point. For example, reading a thermometer might well result in a whole number and be accurate to +/- 0.5 degrees. While measurement error is simple to understand there are many other types of error. Thus, each data point can be understand to have an associated uncertainty.

A **set** of scalar data points is a group of numerical values. There are a few mathematical characteristics of a set of numbers.

The **average** or **mean** is given by

$$\bar{x} = \frac{1}{n}\sum_{i}^{n} x_i$$

The **median** is given by the value in the middle when the set is ordered with an odd quantity. For even it is the average of the middle two numbers.

The **standard deviation** is a measure of the variability of the set of data and is given by

$$\sigma = \frac{1}{n-1}\sum_{i}^{n} (x_i - \bar{x})^2$$

How much data is enough?

Rules of thumb*

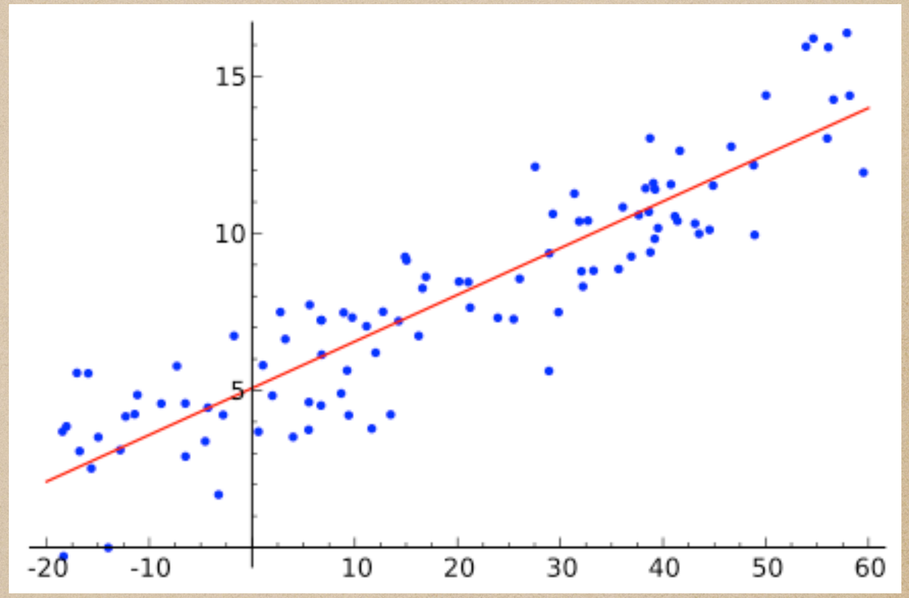
- Expectation, average, mean - 7 or 8 data points
- Standard deviation, linear model, expected range - 13 data points
- 1% (rare) events - hundreds of data points

*For data that is normally distributed. Other distributions will require more.

Regression

- **Observation** - Experimental data or measurement. Expressed as a row vector **d** = [d_1, …, d_n].
- **Model** - A mathematical function.

We talk about **fitting** a model to the observations. One way is to “eyeball” the fit but generally a more automatic and rigorous approach is used.



Models

In a previous slide, the model was a line

$f(x) = mx+b$

but the model can be any function. Sometimes the problem suggests the function (e.g. elliptical orbit). There are approaches for the case when the function is unknown but this requires a lot of data and subjects the results to a lot more questioning. But sometimes the function that is found to provide the best fit can suggest a theoretical underpinning.

Error

To determine how good the fit of the model is to the data we need to be able to measure the fit.

If we add an error term we have f(x) + e where then for each data point we have

$e_i = d_i - f(x_i)$

Then we have e_i, i=1, …, n where there are n data points. This is an error vector **e**. How do we know if this error vector is “small?” First we have to be able to compare the size of vectors.

Norms

A norm $\|\mathbf{x}\|$ is a non-negative number where $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$. A norm must also have $\|k\mathbf{x}\| = |k| \, \|\mathbf{x}\|$ and the triangle inequality $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (the legs of a triangle cannot be shorter than the direct distance or hypotenuse).

Common norms:

$\|\mathbf{x}\|_1 = \sum |x_i|$
$\|\mathbf{x}\|_2 = (\sum x_i^2)^{½}$
$\|\mathbf{x}\|_\infty = \max |x_i|$

Norms are important because they provide a way to get a single number representing a measurement of a vector.

If we consider a two dimensional space where $\mathbf{x} = [3, 4]$ then we have

$\|\mathbf{x}\|_2 = (3^2 + 4^2)^{½} = 5$

For Euclidean geometry this is simply the Euclidean distance.

Least Squares Method

The Least Squares Method is just choosing the parameters for the function such that $\|\mathbf{e}\|_2$ is minimized.

- Linear problem
- Easy to solve
- Usually adequate fit

Fits using the Method of Least Squares can be sensitive to data outliers. The Method of Least Squares is often used where there are better alternatives because it is easy to use and widely accepted.

R Statistical Software

R is a free statistical analysis software package available for Windows, Mac and Linux.

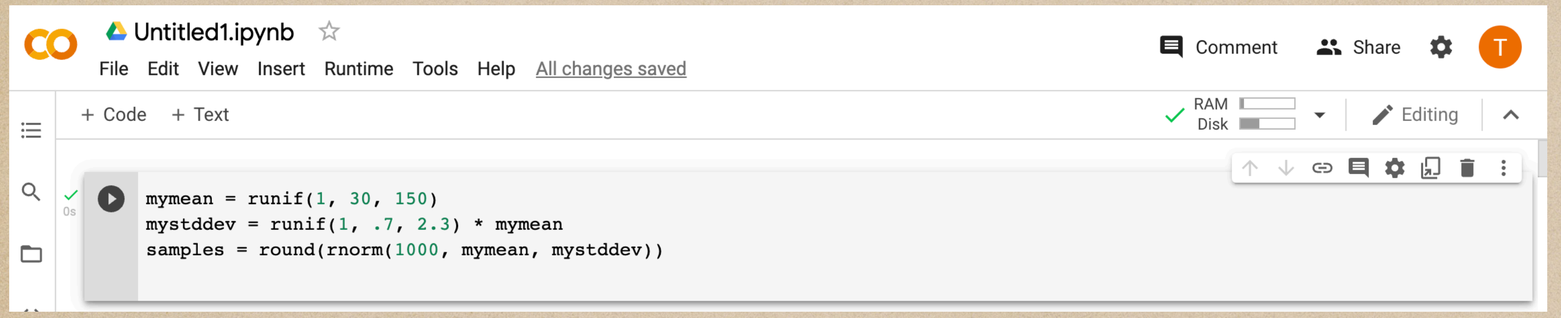
Binary distributions are available for Windows and Mac OS X at https://cran.r-project.org/. Many Linux distributions have R available in the package management systems or check the link above.

Jupyter notebooks (Julia - Python - R) can also be set up with the R kernel. Google Colab which is online Jupyter notebooks also can be used with R. The appendices have instructions for programming in R using Google Colab.

Mexican Wolves Model

For this example we will construct a pseudo-model of the Mexican Wolf population. Our model produces an estimate of the wolf population sometime in the future. To construct our model we randomly select a mean and standard deviation. The mean is somewhere between 30 and 150 wolves. Using R

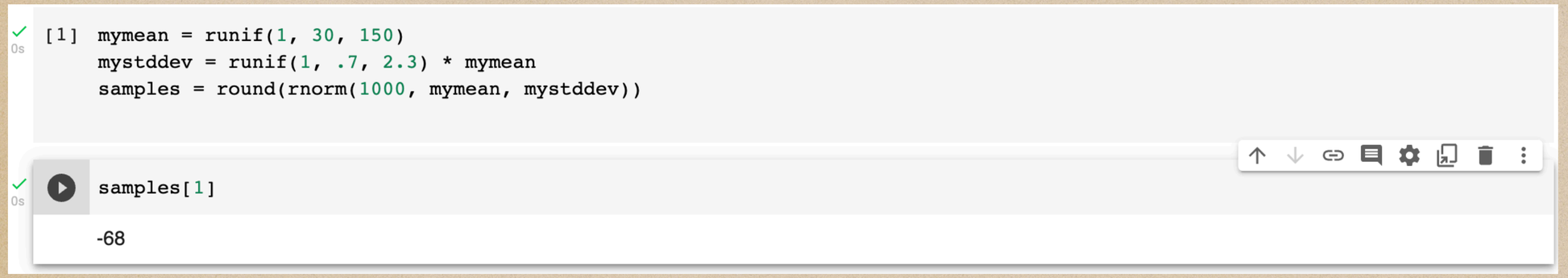
```
> mymean = runif(1, 30, 150)
> mystddev = runif(1, .7, 2.3) * mymean
> samples = round(rnorm(1000, mymean, mystddev))
```



The samples vector contains data generated by the model which is a normal distribution. The model does not capture much about the physical environment of the wolves but we know a lot about what data the model should produce.

Each student will now have their own model of the Mexican wolf population. The student then runs his/her model one time

> samples[1]



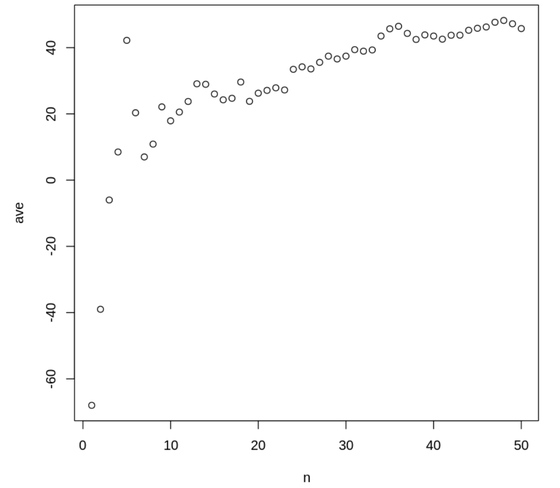
This gives the number of wolves. The student should now write down their analysis of the results of their model. How likely is it that the wolves will survive?

Now run the model multiple times. samples[2] gives the second run. To get the results of ten runs type samples[1:10]. Does the interpretation of the model change as data is added?

Now we are going to compute the average for the first data point and then two data points and so on as data is added.

```
> n = seq(1, 50, 1)
> ave = c()
> for (i in 1:50) {ave[i] = mean(samples[1:i])}
> plot(n, ave)
```

```
n = seq(1, 50, 1)
ave = c()
for (i in 1:50) {ave[i] = mean(samples[1:i])}
plot(n, ave)
```

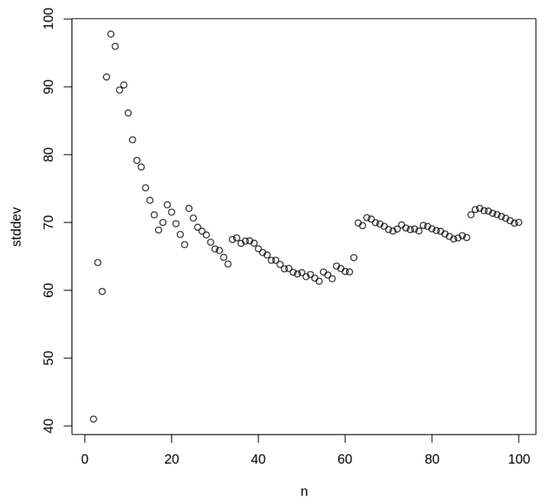


What is your estimate for the average? How many samples does it take to get a good estimate?

Now repeat this using the standard deviation. The standard deviation is a measure of the variability of the output.

```
> n = seq(1, 100, 1)
> stddev = c()
> for (i in 1:100) {stddev[i] = sd(samples[1:i])}
> plot(n, stddev)
```

```
n = seq(1, 100, 1)
stddev = c()
for (i in 1:100) {stddev[i] = sd(samples[1:i])}
plot(n, stddev)
```

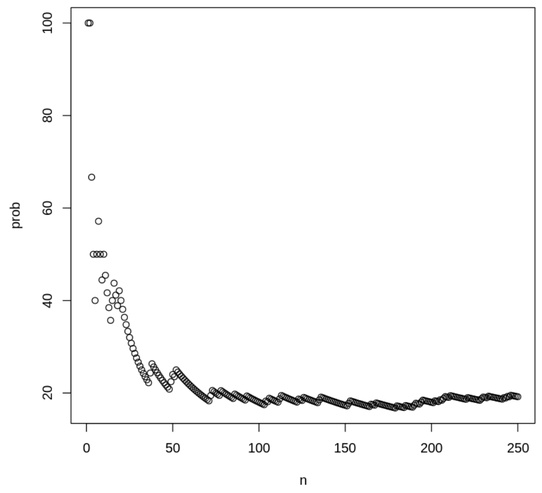


What is your estimate of the standard deviation? How many samples did it take to get a realistic estimate?

Now calculate the probability that the wolf population goes extinct.

```
> n = seq(1, 250, 1)
> prob = c()
> for (i in 1:250) {prob[i] = 100.0 * sum(samples[1:i] <= 0)/i}
> plot(n, prob)
```

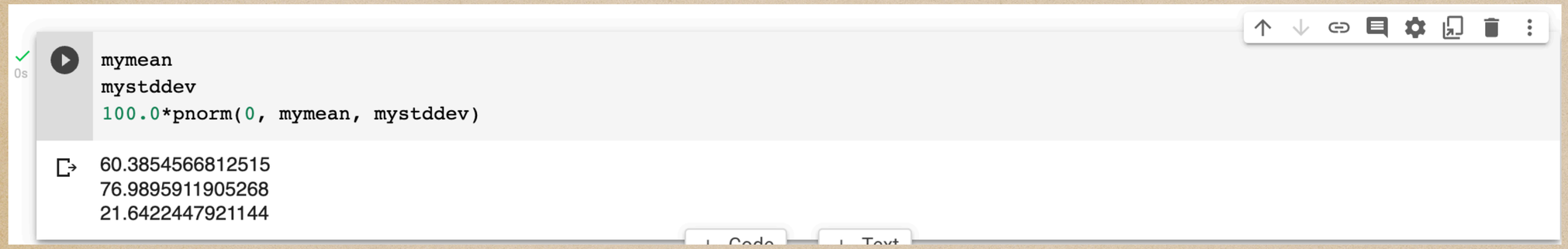
```
n = seq(1, 250, 1)
prob = c()
for (i in 1:250) {prob[i] = 100.0 * sum(samples[1:i] <= 0)/i}
plot(n, prob)
```



What is your estimate of the probability that the wolf population goes extinct? How many samples did it take to get a reliable estimate of the probability? Lower probabilities will generally take more data. Why is this?

Now compare your estimates with mymean, mystddev and for the probability

> mymean
> mystddev
> 100.0*pnorm(0, mymean, mystddev)



Two Dimensional Data Points

Consider the case where we have two data points of two dimensional data. If we fit a linear model to the two data points what is the result? Since two points determine a line, the line exactly describes the data. In general, a polynomial model of degree $m = n - 1$ will exactly describe n data points. Is this a good idea?

The **sum of square errors** (SSE) is $(\|\mathbf{e}\|_2)^2$. How do we choose the degree of the polynomial, m, in the model? One way is to compute

$$(\|\mathbf{e}_m\|_2)^2/(n - m - 1)$$

and continue increasing m as long as the amount decreases significantly. What is the value of this when the degree of the polynomial exactly fits the data points?

Coefficient of Determination

The **coefficient of determination** is a measure of how well the model fits the data. For a linear regression model

$$R^2 = \{ (1 / n) * \Sigma [(x_i - ave(x)) * (y_i - ave(y))] / (\sigma_x * \sigma_y) \}^2$$

- The coefficient of determination ranges from 0 to 1
- $R^2 = 0$ means the model has no predictability for the data
- $R^2 = 1$ means the model perfectly predicts the data

A common error that is made is using a high R^2 to say that the model is the right model. Some data sets are easy to get a high R^2 and others are not.

Simple Linear Regression

The case where the model is y = b + mx has some rather simple formulas.

$$m = (\sum y_i x_i - \sum y_i \sum x_i /n)/(\sum(x_i - ave(x))^2)$$

$$b = ave(y) - m\ ave(x)$$

The SimpleLinearRegression.pdf in the examples is a worksheet for calculating a simple linear regression.

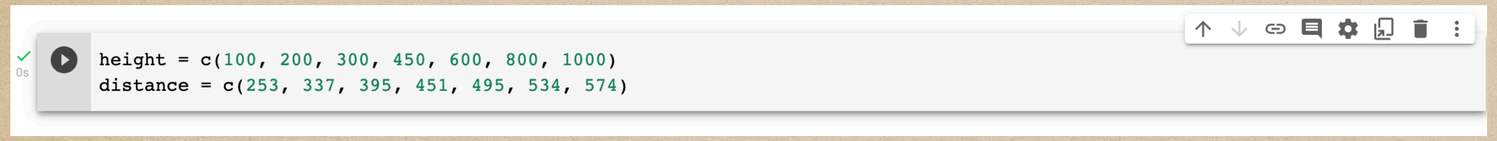
Rocket example

We are given some data pertaining to a rocket.

height = [100, 200, 300, 450, 600, 800, 1000]
distance = [253, 337, 395, 451, 495, 534, 574]

Using R

> height = c(100, 200, 300, 450, 600, 800, 1000)
> distance = c(253, 337, 395, 451, 495, 534, 574)



First, calculate using the formulas for simple linear regression

$$m = (\sum y_i x_i - \sum y_i \sum x_i /n)/(\sum(x_i - ave(x))^2)$$
$$m = (1712350 - 3039 * 3450/7)/642143$$
$$m = (1712350 - 1497793)/642143$$
$$m = 0.3341$$

$$b = ave(y) - m\ ave(x)$$
$$b = 434.143 - 0.3341 * 492.857$$
$$b = 269.48$$

$$f(x) = 269.48 + 0.3341*x$$

Fitting a linear model

> model1 <- lm(distance ~ height); model1

```
model1 <- lm(distance ~ height); model1
```

```
Call:
lm(formula = distance ~ height)

Coefficients:
(Intercept)       height
   269.4661       0.3341
```

How does this compare to what we calculated by hand? Using R we can easily fit a quadratic polynomial to the data.

> model2 <- lm(distance ~ height + I(height^2)); model2

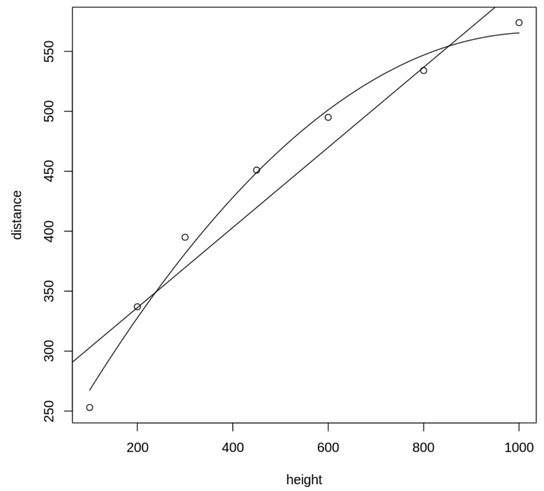
```
model2 <- lm(distance ~ height + I(height^2)); model2
```

```
Call:
lm(formula = distance ~ height + I(height^2))

Coefficients:
(Intercept)       height  I(height^2)
 200.211950     0.706182    -0.000341
```

```
> newh = seq(100, 1000, 10)
> fit2 = 200.211950 + 0.706182*newh - 0.000341*newh^2
> plot(height, distance)
> abline(model1)
> lines(newh, fit2, lty=1)
```

```
newh = seq(100, 1000, 10)
fit2 = 200.211950 + 0.706182*newh - 0.000341*newh^2
plot(height, distance)
abline(model1)
lines(newh, fit2, lty=1)
```



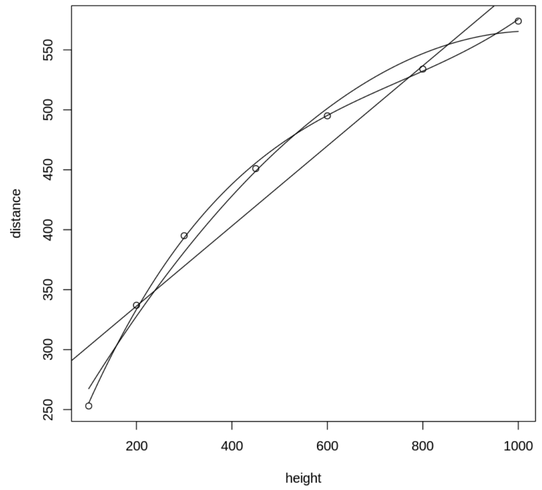
Repeat for a cubic

> model3 <- lm(distance ~ height + I(height^2) + I(height^3)); model3
> fit3 = 1.555e+02 + 1.119*newh - 1.254e-03*newh^2 + 5.55e-07*newh^3
> lines(newh, fit3, lty=1)

```
newh = seq(100, 1000, 10)
fit2 = 200.211950 + 0.706182*newh - 0.000341*newh^2
plot(height, distance)
abline(model1)
lines(newh, fit2, lty=1)
model3 <- lm(distance ~ height + I(height^2) + I(height^3)); model3
fit3 = 1.555e+02 + 1.119*newh - 1.254e-03*newh^2 + 5.55e-07*newh^3
lines(newh, fit3, lty=1)
```

```
Call:
lm(formula = distance ~ height + I(height^2) + I(height^3))

Coefficients:
(Intercept)       height  I(height^2)  I(height^3)
  1.555e+02    1.119e+00   -1.254e-03    5.550e-07
```



Would you choose the linear, quadratic or cubic model for this data? Does the information that this data comes from a rocket help in your choice? Would it make a difference if you were told it was a one stage or two stage rocket?

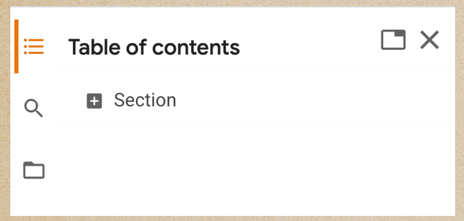
Can you calculate the following for each of these three curves? Does this agree with your decision of the best model?

$$(\|\mathbf{e}_m\|_2)^2/(n - m - 1)$$

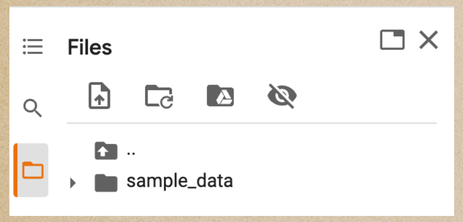
Netlogo example
(coffee mug cooling)

Start Netlogo and read in LinearRegression.nlogo. This model reads in the coffeMugCooling.csv data file. Press setup and then go. This shows the data and a least squares fit of a line to the data. The slope and intercept of the line are displayed. Do you think that a line is the right model for this data? If not, what would be a good model?

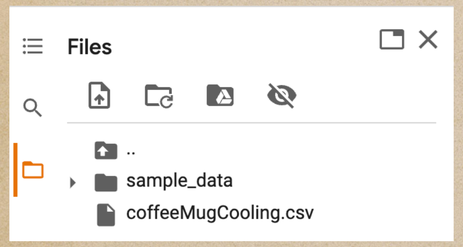
Using R for the coffee mug cooling example. First you will need to upload the data file coffeMugCooling.csv. Then in the upper left corner click on the folder icon.



Then in the upper left corner click on the folder icon.



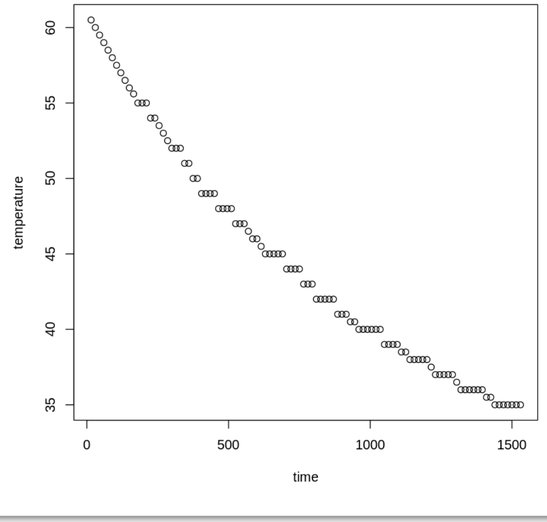
Now click on the page with up arrow icon to upload your data file.



The data file coffeMugCooling.csv should now appear.

```
> mug <- read.csv("coffeeMugCooling.csv", sep=",");mug
> time = mug[,1]
> temperature = mug[,2]
> plot(time, temperature)
```

```
mug <- read.csv("coffeeMugCooling.csv", sep=",");mug
time = mug[,1]
temperature = mug[,2]
plot(time, temperature)
```

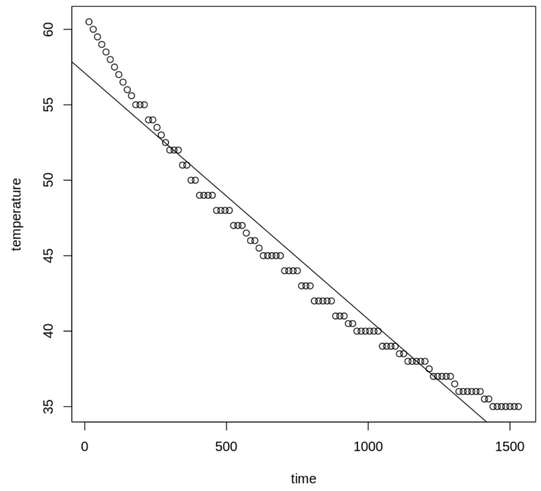


> model1 = lm(temperature~time); model1
> abline(model1)

```
mug <- read.csv("coffeeMugCooling.csv", sep=",");mug
time = mug[,1]
temperature = mug[,2]
plot(time, temperature)
model1 = lm(temperature~time); model1
abline(model1)
```

```
Call:
lm(formula = temperature ~ time)

Coefficients:
(Intercept)         time
   57.10177     -0.01631
```



How do these compare to the Netlogo model?

Fit a quadratic model to the data.

```
> model2 = lm(temperature~time+I(time^2)); model2
> t = seq(0, 1530, 10)
> temp = 60.21 - 0.02826*t + 7.733e-06*t^2
> lines(t, temp, lty=1)
```

```
mug <- read.csv("coffeeMugCooling.csv", sep=",");mug
time = mug[,1]
temperature = mug[,2]
plot(time, temperature)
model1 = lm(temperature~time); model1
abline(model1)
model2 = lm(temperature~time+I(time^2)); model2
```

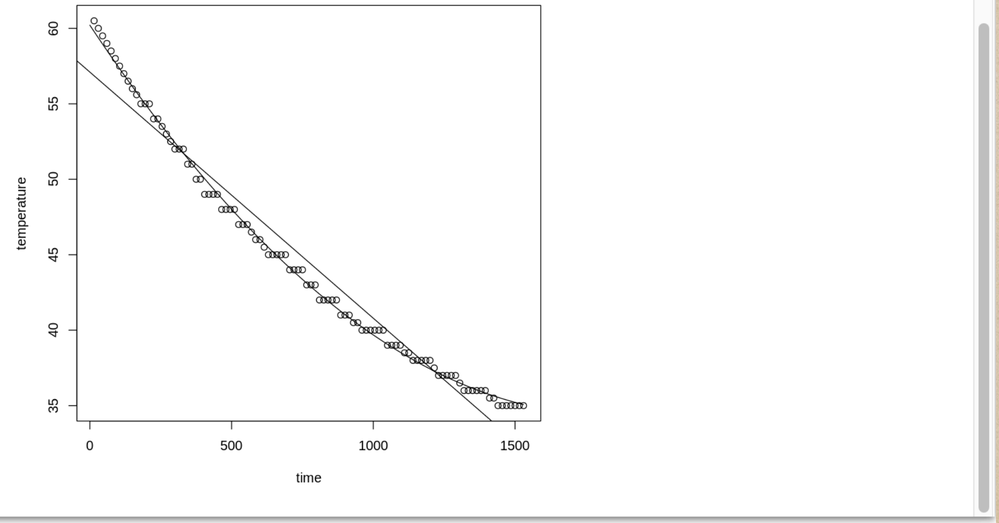
```
Call:
lm(formula = temperature ~ time + I(time^2))

Coefficients:
(Intercept)         time    I(time^2)
  6.021e+01   -2.826e-02    7.733e-06
```

Take the coefficients and plot the line.

> t = seq(0, 1530, 10)
> temp = 60.21 - 0.02826*t + 7.733e-06*t^2
> lines(t, temp, lty=1)

```
mug <- read.csv("coffeeMugCooling.csv", sep=",");mug
time = mug[,1]
temperature = mug[,2]
plot(time, temperature)
model1 = lm(temperature~time); model1
abline(model1)
model2 = lm(temperature~time+I(time^2)); model2
t = seq(0, 1530, 10)
temp = 60.21 - 0.02826*t + 7.733e-06*t^2
lines(t, temp, lty=1)
```

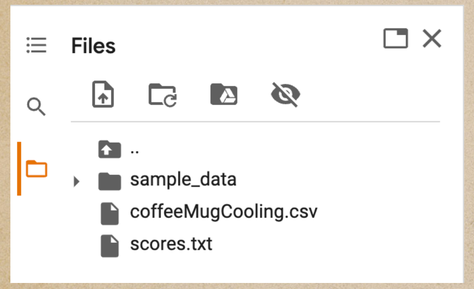


Is the linear or quadratic fit a better model for the data? Why? Are there are other models that might be a better choice for this problem?

Advanced example

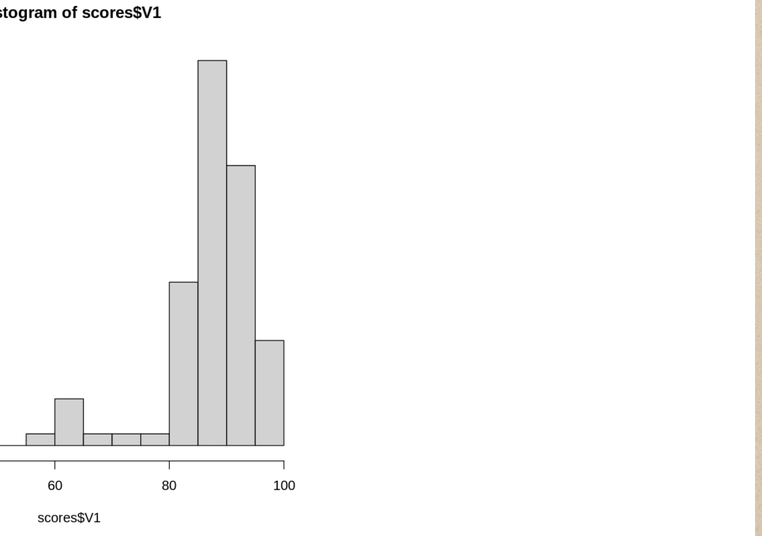
A **probability density function** (pdf) describes the likelihood of a value. The first step is to plot the histogram and compare the shape to types of pdfs. If all values are equally likely then it is a **uniform distribution**. Test scores usually form a normal distribution (bell curve). Often looking at the domain gives a clue. If the domain is from [0, inf] then the log normal distribution may be a better choice than the normal distribution which has a range of [-inf, inf]. The data for this example is a set of test scores.

Upload the scores.txt file as described in the coffee mug example.



```
> scores <- read.table("scores.txt")
> hist(scores$V1, 20)
```

```
scores <- read.table("scores.txt")
hist(scores$V1, 20)
```

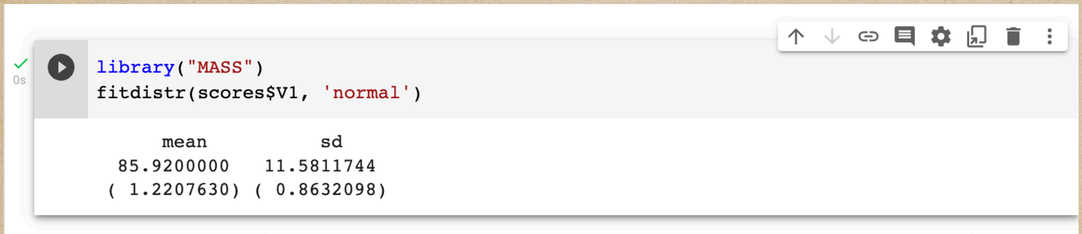


Least squares is not always the best measure of fit for fitting probability distribution functions to data. If you have a f_i data point from the tail of the data then the fitted pdf has $p_i = 0$; this says there is no possibility of that value ever happening but it happened. $e_i = p_i - f_i$ is a small part of the overall error vector and the 2-norm so it is very possible the fitted pdf using least squares will have this happen. It may not be important if the problem does not care about the tails of the pdfs. But there are measures instead of using norms that can give better results. The MASS module for R uses a maximum likelihood estimation to fit a probability density function to a set of data.

The normal distribution is

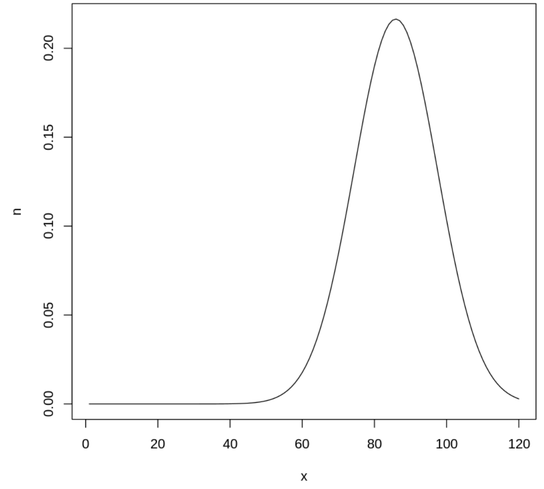
$$y=\frac{1}{\sigma sqrt(2\pi)}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

```
> library("MASS")
> fitdistr(scores$V1, 'normal')
```



```
> x = seq(1, 120, 1)
> n = (1/11.5811744*sqrt(2*3.1415))*exp(-((x-85.92)^2)/(2*11.5811744^2))
> plot(x, n, "l")
```

```
x = seq(1, 120, 1)
n = (1/11.5811744*sqrt(2*3.1415))*exp(-((x-85.92)^2)/(2*11.5811744^2))
plot(x, n, "l")
```



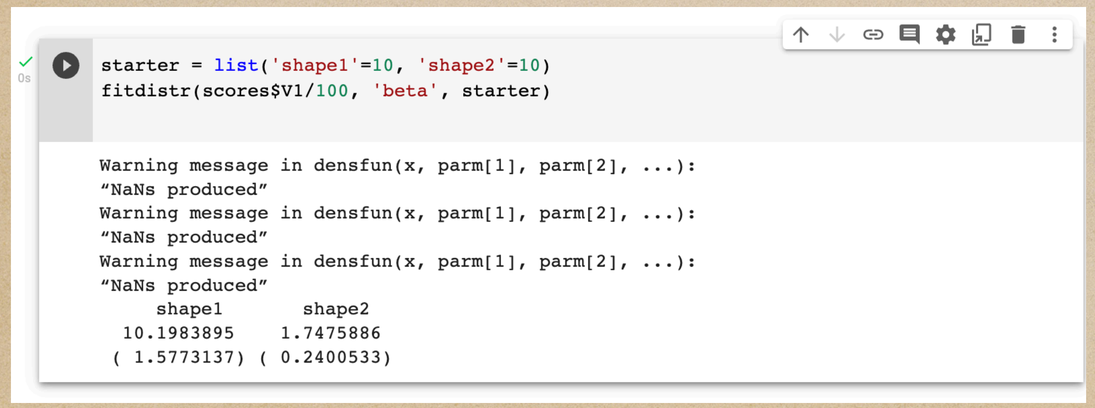
For many cases this may be sufficient and typically is an acceptable approach because other methods are much more difficult. But R does have tools that can make these other methods accessible. Next is an example.

The domain of the normal distribution goes from [-inf, inf]. Is this realistic? Note that the plot of the normal distribution shows values greater than 100 are common. The beta distribution has the domain [0, 1] and can mimic the normal distribution and many other distributions.

$$y = Cx^{\alpha-1}(1-x)^{\beta-1}$$

To fit this we need to supply starting guesses for alpha (shape1) and beta (shape2). We also need to divide our test scores by 100 to get them into the domain [0, 1].

```
> starter = list('shape1'=10, 'shape2'=10)
> fitdistr(scores$V1/100, 'beta', starter)
```

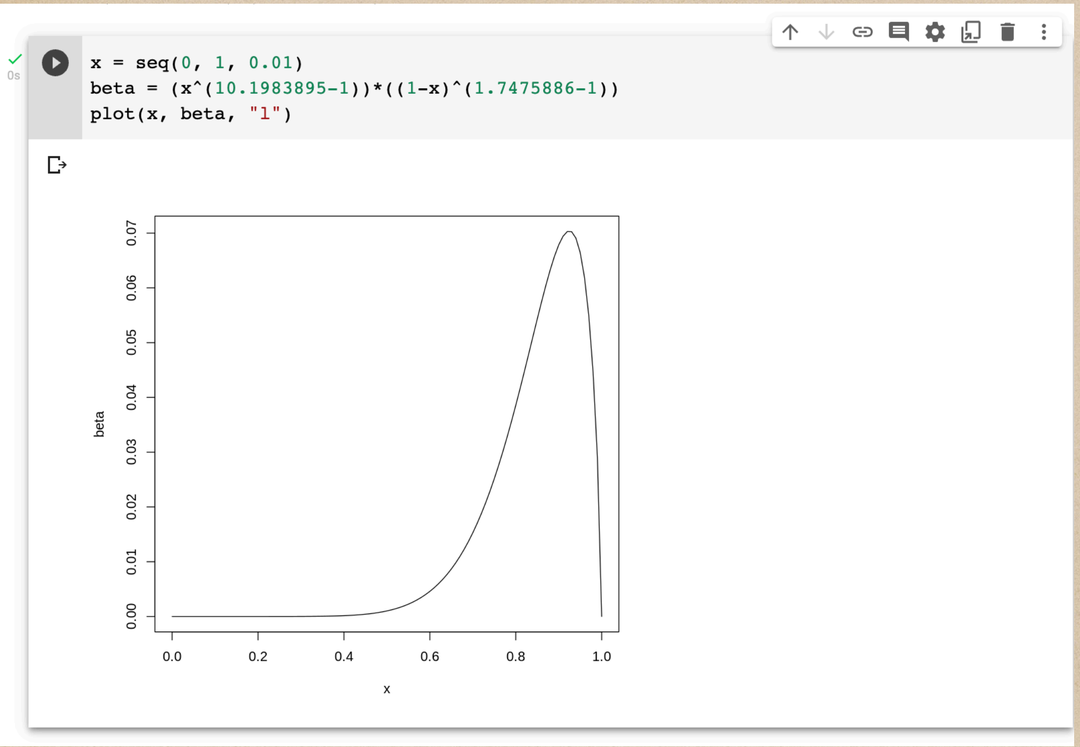


```
starter = list('shape1'=10, 'shape2'=10)
fitdistr(scores$V1/100, 'beta', starter)
```

```
Warning message in densfun(x, parm[1], parm[2], ...):
“NaNs produced”
Warning message in densfun(x, parm[1], parm[2], ...):
“NaNs produced”
Warning message in densfun(x, parm[1], parm[2], ...):
“NaNs produced”
     shape1        shape2
  10.1983895     1.7475886
 ( 1.5773137) ( 0.2400533)
```

Take these results and plot them.

```
>  x = seq(0, 1, 0.01)
>  beta = (x^(10.1983895-1))*((1-x)^(1.7475886-1))
>  plot(x, beta, "l")
```

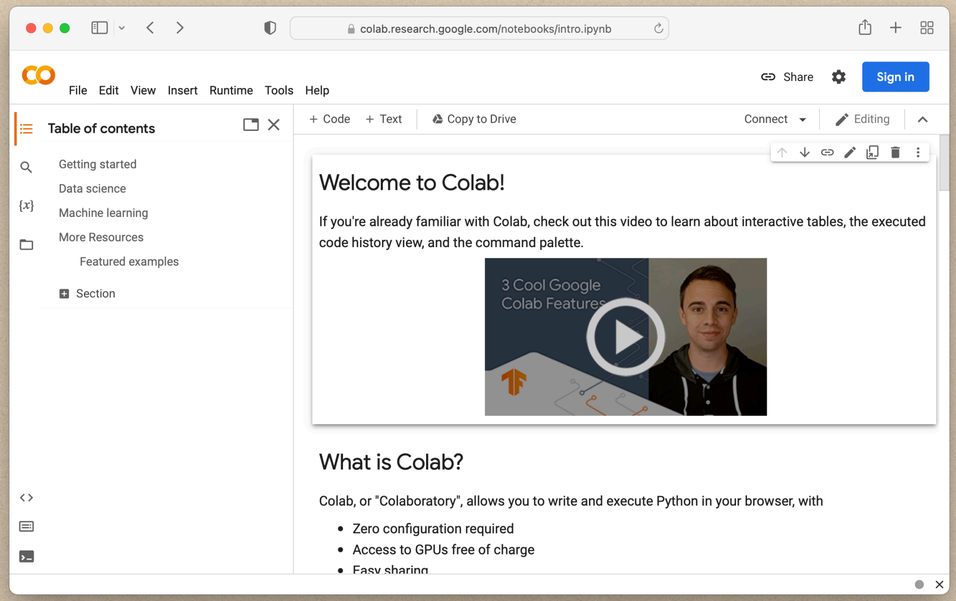


Do you think that the normal distribution or the beta distribution is a better fit for the test scores? Why?

Appendix I: Google Colab

Instructions for using Google Colab. Google Colab is an online version of Jupyter Notebooks. Go to

https://colab.research.google.com/notebooks/intro.ipynb



Click on the blue “sign in”button in the upper left corner and login to your Google account.

In the upper left corner click on File and New notebook.

Alternate approach

Go to https://accounts.google.com and log into your Google account. In the upper right corner click on the icon with 3 x 3 dots and click on Drive. Then click on + New, select More and then Google Colaboratory

Appendix II: R-notebook in Google Drive

To create a new R-notebook use the link:

https://colab.research.google.com/notebook#create=true&language=r

or the shorthand version

https://colab.to/r