HOW TO USE:

```
// This is Python code. Color is from my IDE theme.
```

```
# This is R code, with no styling. Sometimes, this is just pure English.
```

```
/*
This is SAS Code
*/
```

Read CSV

```
import numpy as np
data = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/midterm_marks")
bats = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/bats.csv", skiprows=[1,2]) # Skip first 2 rows, keeping header
bats = pd.read_fwf("/Users/gan/Desktop/Mods/ST2137/data/ex_1_fixed.txt", widths=[2, 1, 3, 3, 1], header=None) # Fixed width

data.columns = ["1st", "2nd"] # Must input all col names, no need assign back
data = df.rename(columns={'2nd' : "New name"}) # Rename specific column by comma separated, must assign back
crab = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/crab.txt", header=0, delim_whitespace=True) # Treat any number of white space as 1
```

```
data = read.csv("/Users/gan/Desktop/Mods/ST2137/data/midterm_marks")
```

```
/* To import data: https://video.sas.com/detail/video/4664358166001/using-the-import-data-utility-in-sas-studio
Click on Left side Task -> Utility -> Import data. Or, right click the file -> Import data.
To specifiy delimiter for Tab, insert with quotes '09'x and uncheck quote delimiter
To access file path: Right click file -> Properties
*/

/*
To import fixed width data: Cannot use the import button. Need write code.
*/
data fixed_width_data;
        infile "/folders/myshortcuts/Working_Folder/ex_1_fixed.txt";
        input var1 1-2 var2 $ 3 var3 5-6 var4 8-9 var5 $ 10;
run;

/* To rename columns: */
data height_weight;
set height_weight(rename=(gender=NewGender height=NewHeight));
run;
```

Read Excel

```
pinot = pd.read_excel("/Users/gan/Desktop/Mods/ST2137/Midterm/pinot_noir.xls")
# The rest like rename, and accessing, all same as above, read csv
```

```
install.packages("readxl")
library(readxl)
pinot = read_excel("/Users/gan/Desktop/zquiz1/pinot_noir.xls")
```

Just import normally

After importing csv, access the column called 'X'

```
df['x'] # df = pd.read_csv(''), or use df.x
c2 = df[['height', 'weight', "gender"]] # Read multiple col, note the inner bracket
print(df.iloc[:, 0:2]) # Read first 2 col of df
```

```
data['x'][, 1]  // data = read.csv("")
OR
data$x
```

Create Dataframe

```
data = {"1stCol": [1,2,3,4], '2ndCol':[5,6,7,8]}
df = pd.DataFrame(data, columns=['1stCol', '2ndCol'])
print(list(df)) # Get title, ['1stCol', '2ndCol']
```

```
matr = matrix( c(1:9), 3, 3)
df = data.frame(matr)
or df = data.frame(colname = matr)
names(df) = c("Col1", "Col2", "Col3")
```

```
data mynewdata;
        input subject gender $ CA1 CA2 HW $;
        datalines;
10 M 80 84 A
7 M 85 89 A
4 F 90 86 B
20 M 82 85 B
;
run;

/* Create fixed format data */
data my_fix_format_data;
        input subject 1-2 gender $ 3 CA1 5-6 CA2 8-9 HW $ 10;
        datalines;
10m 80 84a
7 m 85 89a
14f 88 84c
```

```
;
```

Transformation of data

To transform

```
> tablets
   lab1 lab2 lab3 lab4 lab5 lab6 lab7
1  4.13 3.86 4.00 3.88 4.02 4.02 4.00
2  4.07 3.85 4.02 3.88 3.95 3.86 4.02
3  4.04 4.08 4.01 3.91 4.02 3.96 4.03
4  4.07 4.11 4.01 3.95 3.89 3.97 4.04
5  4.05 4.08 4.04 3.92 3.91 4.00 4.10
6  4.04 4.01 3.99 3.97 4.01 3.82 3.81
7  4.02 4.02 4.03 3.92 3.89 3.98 3.91
8  4.06 4.04 3.97 3.90 3.89 3.99 3.96
9  4.10 3.97 3.98 3.97 3.99 4.02 4.05
10 4.04 3.95 3.98 3.90 4.00 3.93 4.06
```

into

```
> newtablets
   amount lab
1    4.13   1
2    4.07   1
3    4.04   1
4    4.07   1
5    4.05   1
6    4.04   1
7    4.02   1
8    4.06   1
9    4.10   1
10   4.04   1
11   3.86   2
12   3.85   2
13   4.08   2
14   4.11   2
15   4.08   2
16   4.01   2
```

```
amount = c(tablets$lab1, tablets$lab2, tablets$lab3, tablets$lab4, tablets$lab5, tablets$lab6, tablets$lab7)
lab = c(rep(1, length(tablets$lab1)), rep(2, length(tablets$lab2)), rep(3, length(tablets$lab3)), rep(4, length(tablets$lab4)), rep(5, length(tablets$lab5)), rep(6, length(tablets$lab6)), rep(7, length(tablets$lab7)))
newtablets = data.frame(amount = amount, lab = lab)
```

```
tablets = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/tablets1.txt")


tablets.columns = ["lab1", "lab2", "lab3", "lab4", "lab5", "lab6", "lab7"]


amount = np.array([tablets.lab1, tablets.lab2, tablets.lab3, tablets.lab4, tablets.lab5, tablets.lab6, tablets.lab7]).flatten()


x = [1, 2, 3, 4, 5, 6, 7]
lab = np.repeat(x, [len(tablets.lab1), len(tablets.lab2), len(tablets.lab3), len(tablets.lab4), len(tablets.lab5), len(tablets.lab6), len(tablets.lab7)])



df = pd.DataFrame({"Amount": amount, "Lab": lab})
```

```
proc sql;
create table mysql_table like tablets;
insert into mysql_table
select * from tablets;

create table mysql_table2(
        amount numeric,
        lab numeric
);
```

```
insert into mysql_table2(amount, lab)
select Lab1, 1 from mysql_table
union all
select Lab2, 2 from mysql_table
union all
select Lab3, 3 from mysql_table
union all
select Lab4, 4 from mysql_table
union all
select Lab5, 5 from mysql_table
union all
select Lab6, 6 from mysql_table
union all
select Lab7, 7 from mysql_table
;
```

Add, replace, delete Dataframe

```
df.iloc[0:5, ] # First 5 rows
newCol = [9, 10, 11, 12]
df['NewColName'] = newCol # Add new col
df['ExistingName'] = newCol # Replace existing col with new values
del df['ExistingName'] # Delete existing column
df.loc[4] = [-1, -1, -1] # Replace row if index exist, else create the new index and insert the row
df.loc[len(df)] = [-2, -2, -2] # Append new row to the end
df = df.drop(df.index[2]) # Remove Row 3, must assign back
```

Merge

```
data = pd.read_csv()
newData = pd.read_csv()
data = pd.concat([data, newData], axis=1) # Add the 2 dataframe together by putting newData to new column, doesnt natural join
newData2 = pd.merge(df1, df2, on=['id', 'test']) # df1 JOIN df2 ON df1.id = df2.id AND df1.test = df2.test
```

Set value based on another value

```
@@@@ Method 1
df = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/test.csv")
df['Grade'] = 'F' # Initialise all value, optional
df.loc[df.test > 50, "Grade"] = "D" # If df['test'] value > 50, set df['Grade']. Grade can be existing
```

```
col or new col. Cannot call df['test'], must be df.test
df.loc[df.test > 60, "Grade"] = "C" # Set in descending order


@@@@ Method 2


grade2 = list()
for i in range(len(df)):
    if df['test'][i] >= 80:
        grade2.append("A")
    elif df['test'][i] >= 70:
        grade2.append("B")
    else:
        grade2.append("F")
df['Grade2'] = grade2
```

```
df['NewCol'] = c(10, 11, 12)
which(names(df) == 'Col3') # Return the index 3
df = df[, -2] # Remove the 2nd col
df = df[, which(names(df) == 'Col3') * -1] # Remove the col called 'col3'
df = df[-2, ] # Remove the 2nd row
nrow(df) # Return number of rows of df
The rest all same as matrix
```

```
data mydata4;
        merge mydata mydata2;
        by id; /* Must be sorted */
run;
```

Create Matrix

```
matrix = np.array([ [1,2,3,4,5], [6,7,8,9,10] ])
```

```
matr = matrix( c(1,2,3,4,5,6,7,8,9,10), 2, 5)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Accessing 2nd col:

```
matrix[:, 1] # [2 5 8]
```

```
matr[, 2]
```

Accessing 2nd and 3rd row:

```
matrix[1:3,]
```

matr[2:3,]

Accessing (5, 8), i.e. 2nd and 3rd row of the 2nd col

```
matrix[1:3, 1] # [5 8]
```

matr[2:3, 2]

Add new rows to matrix (append to end):

```
matrix = np.vstack((matrix, [10, 11, 12])) # Does not affect original matrix
```

matr = rbind(matr, c(10, 11, 12)) # non-mutating
or matr = c(matr[1:1], c(10, 11, 12), matr[2:length(matr)])

```
/* Add mydata2 below mydata and save in mydata3. Can be use even if different number col
*/
DATA mydata3;
	SET mydata mydata2;
RUN;
```

Add new rows to matrix after 1st row:

```
matrix = np.insert(matrix, 1, newMatrix, axis=0) # newMatrix can be multirow, or 1 row [10, 11,
12]
```

matr = rbind(matr[1:1,], c(10, 11, 12), matr[2:nrow(matr),]) # non-mutating

Add new col to matrix (append to end):

```
matrix = np.column_stack((matrix, [10, 11, 12])) # Non-mutating
```

matr = cbind(matr, c(10, 11, 12)) # non-mutating

```
data mydata6;
	set work.mydata5;
	myNewCol = 3; /* If myNewCol is not defined then add new col. */
	if weight = "." then final_grade="";
	else if weight > 65 then final_grade="C";
	else final_grade="D";
run;
```

Add new col to matrix after 1st col:

```
matrix = np.insert(matrix, 1, [10, 11, 12], axis=1)
```

matr = cbind(matr[, 1:1], c(10, 11, 12), matr[, 2:ncol(matr)]) # non-mutating

Delete the 3rd col:

```
matrix = np.delete(matrix, 2, axis=1) # Non-mutating
```

matr[, -3] # non-mutating

Delete the 3rd row:

```
matrix = np.delete(matrix, 2, axis=0) # Non-mutating
```

matr[-3,] # non-mutating

Filter

```
bats = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/bats.csv")
print(bats) # All the dataframe
print(bats[bats['type'] == 1]) # Dataframe where type == 1
print(bats['energy'][bats['type'] == 1]) # The energy content where type == 1
## To have a and statement in the filter
k = student[(student.workhour > 0) & (student.workhour < 20)]
```

```
k = [1, 2, 3, 4]
print(list(filter(lambda x: x > 2, k))) # Method 1 => [3, 4]
k = np.array(k)
print(np.where(k > 2)[0]) # Method 2 => [2 3] impt for [0] to get array() element out
```

bats = read.csv(“”) # Have column energy, mass, type
attach(bats) # Now we can access energy, mass, type directly
energy # [43, 44, 23, ….., 1.02]
type # [1 1 1 1 2 2 1 1 …. 3]
energy[which(type == 1)] # [43, 44] Give those energy whose type is 1

- Alternatively, don’t attach:
- bats['energy'][which(bats['type'][, 1] == 2) , 1] # Find type 2

EASY WAY:
Give those energy whose type is 1:
bats$energy[bats$type == 1]
Select all the df rows and all col where type == 1
bats[bats$type == 1,]

```
DATA mydata;
        SET height_weight;
        WHERE gender="M" AND weight > 60;
        KEEP id; /* Col name */
        /* DROP id; */
RUN;
```

Number of rows:

```
print(matrix.shape[0]) # 3, also applies to df from Panda, numpy array
```

nrow(matr)

```
To limit number of rows when saving:
DATA ..
 SET ..
 IF _n_ > 5 THEN /* Only save 5 rows. _n_ refers to each of the row name. */
   STOP;
RUN;
```

Number of col:

```
print(matrix.shape[1]) # 3, Also applies to df from Panda, numpy array
```

ncol(matr)

Mean

```
np.mean(data) # 18.25
```

mean(data)

```
proc means data=work.pinot mean var std Q1 Median Q3 min max;
        var Quality;
run;
```

Median

```
np.median(data) # 18.25
```

median(data)

```
proc means data=work.pinot mean var std Q1 Median Q3 min max;
        var Quality;
run;
```

Mode

```
import scipy.stats as sc

print(sc.mode([1,2,3,3])) # ModeResult(mode=array([3]), count=array([2]))
```

```
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}
```

```
proc means data=work.pinot mean var std Q1 Median Q3 min max mode;
        var Quality;
run;
```

Trimmed Mean

Sort data, then take out x% top and x% bottom, then calculate remaining mean.

```
import scipy.stats as sc

print(sc.trim_mean(bats.energy, 0.2)) # 20% trimmed mean
```

mean(bats$energy, trim=0.2) # 20% trimmed

```
proc univariate data=pinot trimmed=0.4;
 var Quality;
 ods select TrimmedMeans;
 run;
```

Winsorized Mean

Sort data, take out x% top and x% bottom and replace them with the min value of the remaining data and the max value of the remaining data.

```python
import scipy.stats as sc
print(np.mean(sc.mstats.winsorize(bats.energy, [0.2, 0.2]))) # 20% Winsorized mean
```

```r
library(psych)
winsor.mean(bats$energy, 0.2)
```

```
proc univariate data=pinot winsorized=0.2;
        var Quality;
        ods select WinsorizedMeans;
run;
```

MAD Median Absolution Deviation

```python
# Either will do
mad = sc.median_abs_deviation(heats.heat, scale=1)
mad2 = np.median(np.absolute(heats.heat - np.median(heats.heat)))
```

```r
mad(heats$heat, scale=1)
```

Percentile

```python
np.percentile([1,2,3,4,5], 25) # 2.0
```

```r
quantile(c(1,2,3,4,5), 0.25)
```

```
proc univariate data=pinot noprint;
  var Quality;
  output out=data1 pctlpre=P_ pctlpts= 30, 50, 75 to 100 by 5;
run;
proc print data=data1;
run;
```

Matrix Transponse

```python
matrix.T # Shape (5, 2)
```

```r
t(matr) // dim(matr) is 2 5
```

Matrix Inverse

```python
## True if matr is invertible, false otherwise
print(matr.shape[0] == matr.shape[1] and np.linalg.matrix_rank(matr) == matr.shape[0])
## Inverse of matr
inverse = np.linalg.inv(matr)
print(inverse)
```

```r
solve(matrixsq)
```

Variance

```
import statistics as st
st.variance(data) # Use n-1
np.var(data) # Use n
```

var(c(1:9))

For matrix variance:

```
store = dim(matr)
dim(matr) = NULL # flatten to 1d
var(matr)
dim(matr) = store
```

```
proc summary data=pinot;
  var Quality;
  output out=data2 mean= var= / autoname;
run;
proc print data=data2;
run;
```

Range of values

c(1:9) # 1 2 3 4 5 6 7 8 9
seq(from=1, to=10, by=2) # 1 3 5 7
np.arange(1, 10) # 1 2 3 4 5 6 7 8 9
np.arange(3, 7, 2) # 3 5

Replicate

rep(c(1, 2, 3), c(4, 5, 6)) # 1 1 1 1 2 2 2 2 2 3 3 3 3 3

```
x = [1, 2, 3]
y = np.repeat(x, [4, 5, 6])
print(y) # [1 1 1 1 2 2 2 2 2 3 3 3 3 3]
```

or

```
lab = [[1]*len(tablets.lab1), [2]*len(tablets.lab2), [3]*len(tablets.lab3), [4]*len(tablets.lab4),
[5]*len(tablets.lab5), [6]*len(tablets.lab6), [7]*len(tablets.lab7)]


lab_flatten = sum(lab, [])
```

Tilde / Tilda ~

Tilde operator is used to define the relationship between dependent variable and independent variables in a statistical model formula. The variable on the left-hand side of **tilde** operator is the dependent variable and the variable(s) on the right-hand side of **tilde** operator is/are called the independent variable(s).

Sort

```
np.sort(ran)
```
sort(ran)

```
proc sort data=mydata;
        by id;
        /* by descending id; */
run;
```

Reverse

```
arr[::-1] # View of original
```
arr[length(arr):1] # non-mutating
OR
rev(arr) # non-mutating

Copy

```
arr2 = np.copy(arr)
```
matr2 = matr # matr2 and matr are 2 separate copies with same values

Unique

```
arr = np.array([1,1,1,2,3])
print(np.unique(arr)) # [1 2 3]
```
unique(c(1,1,1,2,3))

Skew

```
from scipy.stats import skew
skew(marks)
```
install.packages("moments")
library(moments)
skewness(marks) # -0.4140474
OR use psych

Kurtosis

```
from scipy.stats import kurtosis
kurtosis(marks) # -0.6752517
```
install.packages("psych")
library(psych)
describe(marks)$kurtosis

ggplot

# Plot a bar chart
ggplot(iris, aes(x = Sepal.Length)) + geom_bar()
# Add label
ggplot(iris, aes(x = Sepal.Length)) + geom_bar() + labs(title = 'Hey', x = 'Length', y = 'Money')

```
# Add fill
ggplot(iris, aes(x = Sepal.Length, fill = Species)) + geom_bar()
# Plot 3 graphs by Species (since theres 3 species)
ggplot(iris, aes(x = Sepal.Length)) + geom_bar() + facet_wrap(~ Species)
# Box plot by Species
ggplot(iris, aes(x = Species, y = Sepal.Length)) + geom_boxplot()
# scatter plot separated by 2 types
ggplot(testscore) + geom_point(aes(x = A, y = B, color = "Male"), data=subset(testscore,
testscore$gender == "M"), shape=19) + geom_point(aes(x = A, y = B, color="Female"),
data=subset(testscore, testscore$gender == "F"), shape=24) + scale_color_manual(name =
"Gender", values = c("red", "blue")) + guides(color = guide_legend(override.aes =
list(shape=c(24, 19))))

# Convert a continuous variable into categorical:
Bats$species = as.factor(Bats$species)
```

```
newtablets['lab'] = newtablets['lab'].astype(object)
```

Histogram

```
import matplotlib.pyplot as plt
plt.hist(marks)
plt.title("Histogram Title")
plt.xlabel("Bottom axis")
plt.ylabel("Side axis")
plt.show()
# hist(bins = Any, range: (1,10), density: False (draws PDF), cumulative: False, orientation:
'vertical', color: Any, label: Any)
```

```
hist(marks)
hist(marks, freq=F) # Shows the pdf
plot(ecdf(marks)) # CDF
# To add a PDF of normal not using ggplot:
hist(marks, freq=F)
curve(dnorm(x, mean(marks), sd(marks)), 3, 10, add=T) # Here x is nothing, just x. 3 10
means from x value of 3 to 10
```

```
proc univariate data=work.pinot;
        var Aroma; /* Can choose more than 1 var */
        /* Choose one of the following */
        histogram; /* Pure histogram of Aroma */
        histogram / normal; /* Add a normal dist with mu=mean(Quality) sigma=sd(Quality)
*/
        histogram / normal(color=red mu=13 sigma=2.1);
run;
```

```
## Using ggplot:
```

```
data = c()
data_F = data.frame(data)
ggplot(data_F) + geom_histogram(aes(x = VARNAME, y = ..density..), binwidth = 30)
## To add PDF of Normal
x_range = seq(min(data) – 30, max(data) + 30, length.out = length(data))
y_values = dnorm(x_range, mean(data), sd(data))
ggplot()… + geom_line(aes(x = xrange, y = y_values))
```

Boxplot

```
import matplotlib.pyplot as plt
plt.boxplot(marks)
plt.title('Histogram')
plt.xlabel('Marks')
plt.ylabel('Values')
plt.show()
```

or use panda

```
bats = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/bats.csv")
bats.boxplot(column='energy', by='type')
plt.show()
```

Multiple boxplot:

```
fig = plt.figure()
ax1 = fig.add_subplot(121)
bats.boxplot(column="energy", ax=ax1, grid=False)
ax1.title.set_text("Boxplot of all bats")
ax1.set_xlabel("All bats")
ax1.set_ylabel("Energy")

ax2 = fig.add_subplot(122)
bats.boxplot(column="energy", by="type", ax=ax2, grid=False)
ax2.title.set_text("Boxplot separated by Bats type")
ax2.set_xlabel("Bats type")
ax2.set_ylabel("Energy")

fig.suptitle("Overall Title")
plt.show()
```

boxplot(marks, xlab = “mark”)

##
ggplot(iris) + geom_boxplot(aes(x = Species, y = Sepal.Length))

```
proc sgplot data=pinot noautolegend;
  vbox Quality / category=Region; /* Boxplot separated by region */
run;
```

Multiple box plot separate by a type

bats = read.csv(“”) # Bats have the column energy, type, mass
boxplot(bats['energy'][,1] ~bats['type'][, 1]) # Group by type, show the value for energy
or ggplot:
ggplot(bats, aes(x = type, y = energy)) + geom_boxplot()

QQ Plots

```
import pylab
import scipy.stats as scst
scst.probplot(marks, plot=pylab)
pylab.show()
```

qqnorm(marks)
qqline(marks, col=”red”)

```
proc univariate data=work.pinot;
        var Aroma;
        qqplot / normal(mu=est sigma=est);
run;
```

Scatter Plot

```
import matplotlib.pyplot as plt
midterm = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/midterm_marks")
midterm = midterm['x']
final = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/final_marks")
final = final['x']
plt.scatter(midterm, final)
plt.xlabel('Height')
plt.ylabel('Weight')
plt.title('Weight vs Height')
plt.show()
```

plot(midterm, final)
OR
ggplot(..) + geom_point(aes(x = midterm, y = final))
# Scatter plot with line of best fit:

ggplot(data=ex10) + geom_point(aes(x=weight, y=height)) + geom_smooth(aes(x=weight, y=height), method='lm', formula= y~x)

```
proc sgscatter data=ex10;
plot height * weight; /* Y*X */
run;
```

Scatter plot separated by category:

```
/* Scatter plot separated by gender */
proc sgscatter data=ex10;
/* Y*X */
plot height * weight / datalabel=gender group=gender;
run;

/* More colourful version */
ods graphics on / attrpriority=none;
proc sgplot data=crab;
  title 'Scatter plot of Width and Weight classified by Spine condition ';
  styleattrs datasymbols=(circlefilled squarefilled starfilled);
  scatter x=width y=weight / group=spine markerattrs=(size=10px);
run;
ods graphics / reset;
```

Scatter plot with line of best fit:

```
proc sgplot data=ex10 noautolegend; reg x=weight y=height;
run;
```

Superimpose Plot

```
x = np.arange(1, 31) # 1 to 30
y1 = x * 1
y2 = x * 2
y3 = x * 3
# Plot 3 lines in the same graph
plt.plot(x, y1, color="red") # Plot vales of x from 1-30, and value of y as x
plt.plot(x, y2, color="blue") # Plot vales of x from 1-30, and value of y as x*2
plt.plot(x, y3, color="green") # Plot vales of x from 1-30, and value of y as x*3
plt.show()
```

Plot histogram and PDF

```
plt.hist(df['test'], bins=5, density=True)
x = np.arange(0, 110)
y = sc.norm.pdf(x, loc=np.mean(df['test']), scale=np.sqrt(np.var(df['test'])))
plt.plot(x, y)
plt.show()
```

Plot 2 scatter plot separated by type onto same graph, with legend

```
plt.scatter(gasoline['x10'][gasoline['x11'] == 1], gasoline['y'][gasoline['x11'] == 1], color="red",
marker="x", label="Manual")
plt.scatter(gasoline['x10'][gasoline['x11'] == 0], gasoline['y'][gasoline['x11'] == 0], color="blue",
marker="o", label="Auto")
plt.legend(loc="upper right")
plt.show()
```

In R:
eq = function(x) x
curve(eq, 1, 10) # Draw a straight line, from 1 to 10. Can't use anonymous function.
curve(eq, 1, 10, add=T) # Draw a straight line on existing graph
# Using ggplot:
x_range = seq(1, 10, length.out=length(data$data)
y_range = x_range * 2
ggplot() … + geom_line(aes(x = x_range, y = y_range)) # Draw (x, x*2) line

Settings for multiple Plots

```
bats = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/bats.csv")
## Use the method ax1 below, if you want to have xlabel, ylabel, and title
plt.subplot(1, 3, 1)
plt.hist(bats['energy'][bats['type'] == 1], density=True, label="Type 1") # Show Energy probability
for Type 1
plt.subplot(1, 3, 2)
plt.hist(bats['energy'][bats['type'] == 2], density=True, label="Type 2")
plt.subplot(1, 3, 3)
plt.hist(bats['energy'][bats['type'] == 3], density=True, label="Type 3")
plt.show()
```

Subplot with add subplot, and label

```
fig = plt.figure()
ax1 = fig.add_subplot(121)
ax1.title.set_text("Male")
ax1.set_xlabel("X label")
ax1.set_ylabel("Y label")
ax1.scatter(testscore.A[testscore.gender == "M"], testscore.B[testscore.gender == "M"],
marker="x", color="red", label="Male")
ax2 = fig.add_subplot(122)
```

```
ax2.title.set_text("Female")
ax2.set_xlabel("X label")
ax2.set_ylabel("Y label")
ax2.scatter(testscore.A[testscore.gender == "F"], testscore.B[testscore.gender == "F"],
marker="o", color="blue", label="Female")
plt.show()
```

par(mfrow=c(2, 2)) # 2 by 2 graph
# To use ggplot on multiple side by side:
install.packages("gridExtra")
library(gridExtra)
plot1 = ggplot(..) + …
plot2 = ggplot(..) + …
grid.arrange(plot1, plot2, ncol=2) # Put 2 gg plot side by side

Frequency Table (of 1 categorical variable)

A table that shows the total count, e.g.
Table---
Female: 99
Male: 79
Also plot a frequency table. This is slightly different from histogram in the sense histogram may group values together, but frequency table does not group together values.

```
bats = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/bats.csv")
tab = pd.crosstab(index=bats.type, columns="Counts")
print("Frequency table of Bats Type")
print(tab) # Tabular form
plt.bar(tab.index, np.array(tab).flatten())
plt.title("Frequency table of Bats Type")
plt.xlabel("Bats Type")
plt.ylabel("Count")
plt.show()
```

freq_table = table(bats$type)
barplot(freq_table)
OR
gendergp <- ifelse(gender=="F","Female","Male")
# for every observation in gender, if F then label as Female, else label as Male

Correlation Coefficient

```
np.corrcoef(midterm, final)[0, 1] # 0.7778, Must put [0,1] becos output is a 2x2 matrix
k = np.corrcoef(np.array([ex10.height, ex10.weight, ex10.age])) # > 2 variables
```

Plot line of best fit

```
x = bats.energy[bats.type == 2]
y = bats.mass[bats.type == 2]
plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))(np.unique(x)), color='blue')
```

```
C1 = c(1,2,3,4,5)
C2 = c(6,20,8,9,10)
cor(C1, C2) # -0.08689
cor(cbind(ex10$height, ex10$weight, ex10$age)) # If we want > 2 variables
```

```
proc corr data=ex10 nosimple;
title "Correlation Coefficient";
var height weight age;
run;
```

Function declaration

```
def myfunc(x, y):
 return x * y
```

```
myfunc = function(x, y) {
        return(x * y)
}
```

For loop

```
k = [1,2,3,4,5]
for (index, data) in enumerate(k):
 print(str(index) + ": " + str(data))
```

```
for(i in 1:10) {
        data = i
        index = which(data == i) # only if data is unique
}
```

If-else

```
if ave_mark >= 90 and HW = "A" then final_grade = "A";
else if ave_mark >= 85 then final_grade = "B";
else final_grade = "C";
```

Normal Distribution fn

```
import scipy.stats as sc
sc.norm.ppf(0.5) # Equivalent to qnorm(0.5)
```

qnorm(0.5) # 0. Returns the output where P(X < output) = Input. i.e. input is the area, output is the point.

qnorm(x, mean, sd), e.g. qnorm(0.5, 25, 20). No mean and sd, assume to be standard norm.

```
import scipy.stats as sc
sc.norm.cdf(0.2) # Equivalent to pnorm(0.2)
```

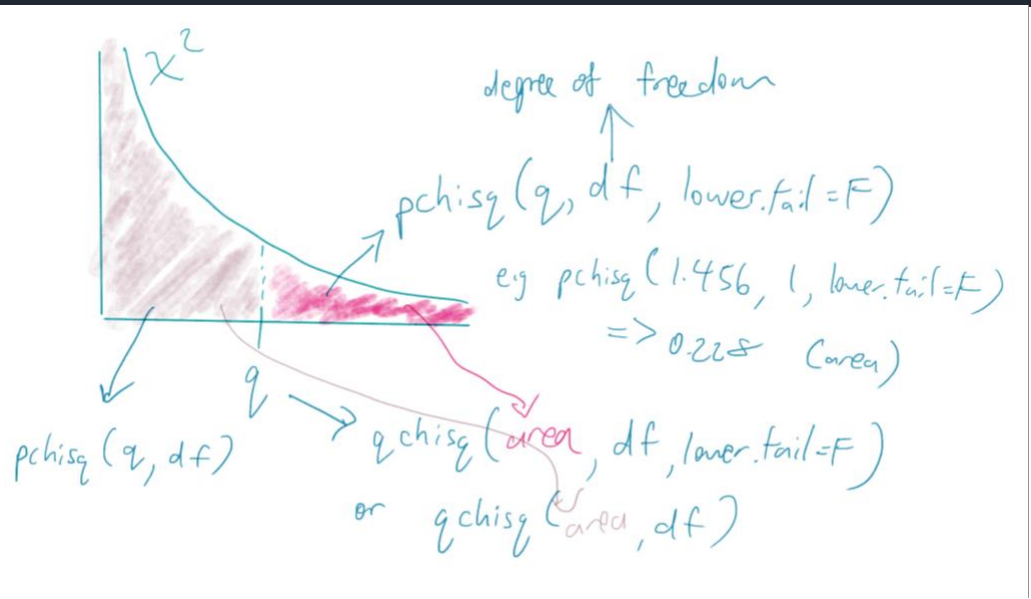
pnorm(q, mean, sd), e.g. pnorm(0) returns 0.5. I.e. P(X < input) = output (area).

```
sc.norm.pdf(x, loc=np.mean(df['test']), scale=np.sqrt(np.var(df['test'])))
```

Equivalent to dnorm
dnorm gives the y-value at the input x-value. E.g. dnorm(0) = 0.3984, i.e. the y-value at the point x = 0.

Chi-square

```
import scipy.stats as sc
sc.chi2.ppf(0.7724, 1) # qchisq(0.7724, 1)
sc.chi2.cdf(1.456, 1) # pchisq(1.456, 1)
```



par(mfrow=c(2, 2)) # 2 by 2 graph
plot(xxx) # Will be at 1,1
plot(xxx) # Will be at 1,2

Student t

```
import scipy.stats as sc
sc.t.ppf(0.3, 2) # qt(0.3, 2)
sc.t.cdf(0.3, 2) # pt(0.3, 2)
```

?dt, pt, qt, rt
Same definition as normal dist above.

F distribution

```
sc.f.cdf(F0, 1, len(x) - 2)  # pf(x, df1, df2)
```

?df, pf, qf, rf

Ask user input

```
while True:
    try:
        userInput = input("Enter input")
        userInput = float(userInput)
        break
    except ValueError:
        pass
print(userInput)
```

```
{
  total_cost = NA
  while (length(total_cost) == 0 || is.na(total_cost) || total_cost <= 0) {
    print("Please enter the cost of your dream home in positive numbers: ")
    total_cost = scan("", what="", nmax=1)
    total_cost = suppressWarnings(as.numeric(total_cost))
  }
  print(total_cost)
}
```

Export

```
df.to_csv("/Users/gan/Desktop/Mods/ST2137/data/MINE.csv")
df.to_csv("/Users/gan/Desktop/Mods/ST2137/data/MINE.txt", sep='\t')
```

write.csv(df, “path”)
write.table(df, “path”, sep=’t’)

```
/* Or just open snippets -> Generate csv file */
proc export data=work.pinot
        outfile="/folders/myshortcuts/Working_Folder/testout2.csv"
        dbms=csv replace;
run;
```

SAS output value to another table

```
ods output WinsorizedMeans=winMeans;
```

Useful Packages

install.packages(“psych”)
library(psych)
describe(df or arr or matr) # Get list of useful things

Print

```
proc print data = subset_heat;
```

```
run;
```

Response vs Explanatory

Response is y-axis. Explanatory is x-axis. Explanatory cause response to change.
Regressor is X, i.e. regressor is explanatory to the target variable (response) Y.

Hypothesis Testing

Rules:

1. State assumptions
2. State hypothesis H0 H1
3. Measure test statistic
4. Calculate p-value. Reject H0 if p-value is small.
5. Conclude

Chi-square Test (Expected count > 5, 2 categorical variables)

1. Assumption: The expected count in all cells are > 5 (<5 use fisher exact test). Applicable to r x c table, not necessarily must be 2x2. Note, does not care about order, hence ordinal variable ordering is ignored. We find the frequency count of the categorical variable, and put them into the table.
2. Hypothesis. H0: The two variables are independent vs. H1: The two variables are dependent.
3. Calculate test statistics.

$$\text{Expected count} = \frac{\text{Row total} \times \text{Column total}}{\text{Total sample size}}$$

The formula χ^2 test statistic (with continuity correction) is:

$$\chi^2 = \sum \frac{(|\text{observed count} - \text{expected count}| - 0.5)^2}{\text{expected count}}.$$

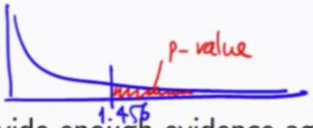
Note that there are variations on this formula in other books/documents.

degree of freedom: (r-1)(c-1)

	Chest Pain	No Chest Pain	Total
Male	40.2 46	479.8 474	520
Female	42.8 37	510.2 516	553
Total	83	990	1073

All the expected counts are larger than 5.

- Test statistic: 1.456 $= \frac{(46-40.2)^2}{40.2} + \frac{(474-479.8)^2}{479.8} + \cdots + \chi^2_1$
- p- value: 0.228



- Conclusion: Data do not provide enough evidence against H_0; At $\alpha = 0.05$,

```
import scipy.stats as sc
data = np.array([[46, 474], [37, 516]]) # Original data, not expected count
```

```
test_statistic, p_value, dof, expected_count = sc.chi2_contingency(data, correction=True) # Also
applicable to table larger than 2x2
print("Test Statistic:", test_statistic)
print("P-value:", p_value)
print("Degree of freedom:", dof)
print("Expected Count:", expected_count) # Make sure all values are > 5
print("All values of expected count are > 5:", np.all(expected_count > 5))
```

```
## To get p-value and test-statistic:
chestpain = matrix(c(46, 37, 474, 516), 2, 2) # Can also be larger than 2x2 table
chisq.test(chestpain) #chestpain is the original 2x2 matrix, not expected count matrix
## To find residual:
chisq.test(politician)$stdres # See page 43 of Topic6 to find out meaning of residual.
## To find expected count of each cell:
chisq.test(politician)$expected
```

4. See p-value
5. Conclude. If p-value < alpha (0.05), reject H0. Else, do not reject H0.

```
/* Reformat data in this way */
data cp3;
input gender $ chest_pain $ Count;
datalines;
M yes 46
M no 474
F yes 37
F no 516
;

/* Look for chisq or continuity adjusted chisq row */
proc freq data=cp3 order=data;
        weight count;
        tables gender*chest_pain / chisq;
run;
```

Fisher Exact Test (Expected count < 5, categorical variables)

1. Assumption. No need expected count in cell to be > 5. ONLY for 2x2 table.
2. Same hypothesis as chi-square test above. H0: The two variables are independent vs. H1: The two variables are dependent.
3. Calculate p-value.

```
import scipy.stats as sc
# Data:
# 4  184
```

```
# 2  260
claritin = np.array([[4, 184], [2, 260]]) # Original data, not expected count
result = sc.fisher_exact(claritin, alternative="two-sided")
p_value = result[1]
print("P-value:", p_value)
```

```
# Data:
# 4  184
# 2  260
claritin = matrix(c(4, 2, 184, 260), 2, 2) # Data is original, not the expected count data
fisher.test(claritin) # Get p-value
```

```
/* Reformat data in this way */
data cp3;
input gender $ chest_pain $ Count;
datalines;
M yes 46
M no 474
F yes 37
F no 516
;

/* Look for fisher's exact test p_value*/
proc freq data=cp3 order=data;
        weight count;
        tables gender*chest_pain / chisq;
run;
```

Linear by Linear Association Test (2 ordinal categorical variables)

1. Assumption. We use this for 2 ordinal variables (where order matters). No need sort, but make sure corresponding row scores are correct.
2. Hypothesis H0: Two variables are independent vs H1: Two variables are dependent. Can also consider H1: Association between two variables is positive/negative.

```
def linear_by_linear_test(data, row_scores, col_scores, alternative="two.sided"):
    data = np.array(data)
    col_scores = np.array(col_scores)
    row_scores = np.array(row_scores)
    nrow = data.shape[0]
    ncol = data.shape[1]
    rsum = np.sum(data, axis=1) # row sums
    csum = np.sum(data, axis=0) # column sum
```

```
    n = np.sum(data) # total cell counts
    rowp = rsum / n  # margin prob for rows
    colp = csum / n  # margin prob for columns
    ubar = np.sum(row_scores * rowp)  # weighted average scores for rows
    vbar = np.sum(col_scores * colp)  # weighted average scores for columns
    numerator = 0
    for v in range(ncol):
        for u in range(nrow):
            numerator = numerator + (row_scores[u] - ubar) * (col_scores[v] - vbar) * (data[u, v] / n)
    V1 = np.sum((row_scores - ubar)**2 * rsum / n)  # weighted variance for rows' scores
    V2 = np.sum((col_scores - vbar)**2 * csum / n)  # weighted variance for columns' scores
    r = numerator / np.sqrt(V1 * V2)  # weighted correlation
    M = np.sqrt(n - 1) * r  # Normalized test statistic
    M2 = M**2
    if alternative == "two.sided":
        # 2 sided
        p_value = 1 - sc.chi2.cdf(M2, 1)
    elif alternative == "less":
        # 1 sided left
        p_value = sc.norm.cdf(np.abs(M))
    else:
        # 1 sided right
        p_value = 1 - sc.norm.cdf(np.abs(M))

    return M2, p_value

data = np.array([[17066, 48], [14464, 38], [788, 5], [126, 1], [37, 1]])
print(data)
print("LINEAR:")
print(linear_by_linear_test(data, [0, 0.5, 1.5, 4, 7], [0, 1]))
```

Using package COIN:
Input =(
"MI Absent Present
Alcohol

```
Zero       17066       48
Below.1     14464       38
1-2          788        5
3-5          126        1
>6           37         1
")

set = as.table(read.ftable(textConnection(Input)))
set
library(coin)
test = lbl_test(set,scores = list(MI = c(0,1), Alcohol = c(0,0.5,1.5,4,7)))
test
```

One sample t-test (Parametric, Quantitative)

1. Assumption: Sample taken randomly. Sample follows or approx. Normal dist, or sample size > 30.
2. Hypothesis: Population mean $\mu = 3.3$ vs μ != 3.3
3. Test statistic:

Sample mean - pop mean

$$T = \frac{\bar{X} - \mu_0}{\text{s.e}(\bar{X})} = \frac{3.208 - 3.3}{\sqrt{0.2564/47}} = -1.2456,$$ = -1.248786

s.e(xbar) = s/sqrt(n) or sqrt(s^2 / n)

where μ_0 is the value of parameter μ under H_0.

3. Under H0, test statistic T follows t-dist with t(n-1) degree of freedom

```
from scipy import stats
test_statistic, p_value = sc.stats.ttest_1samp(baby.weight, popmean=3.3)  # 2-sided p-value
df = len(baby.weight) - 1 # Degree of freedom
one_sided_left_p_value = sc.t.cdf(test_statistic, df)
one_sided_right_p_value = 1 - sc.t.cdf(test_statistic, df)
print("Test statistic: ", test_statistic)
print("2-sided test P-value:", p_value)
print("1-sided-left test P-value:", one_sided_left_p_value)
print("1-sided-right test P-value:", one_sided_right_p_value)
conf_interval_95 = sc.t.interval(0.95, df=df, loc=np.mean(baby.weight),
scale=sc.sem(baby.weight))
print("95% confidence intervale:", conf_interval_95)



# Draw it out
# x = np.linspace(test_statistic - 5, test_statistic + 5)
```

```
x = np.linspace(test_statistic - 5, test_statistic + 5, 100)
y = sc.t.pdf(x, df)
plt.plot(x, y)
# 2 sided
plt.fill_between(x, y, where=(x < -1 * np.abs(test_statistic)), color="orange")
plt.fill_between(x, y, where=(x > np.abs(test_statistic)), color="orange")
plt.title("2 sided t-Test with " + str(df) + " degrees of freedom" + "\nTest-statstic: " +
str(test_statistic) + "\np-value: " + str(p_value))
# 1-sided left
# plt.fill_between(x, y, where=(x < -1 * np.abs(test_statistic)), color="orange")
# plt.title("Left-tailed t-Test with " + str(df) + " degrees of freedom" + "\nTest-statstic: " +
str(test_statistic) + "\np-value: " + str(one_sided_left_p_value))
# 1-sided right
# plt.fill_between(x, y, where=(x > test_statistic), color="orange")
# plt.title("Right-tailed t-Test with " + str(df) + " degrees of freedom" + "\nTest-statstic: " +
str(test_statistic) + "\np-value: " + str(one_sided_right_p_value))
plt.xlabel("x-values")
plt.ylabel("Density")
plt.show()
```

```
t.test(pinot$Quality, mu=13) # 2-Sided
t.test(pinot$Quality, mu=13, alternative="less") # Left-sided
t.test(pinot$Quality, mu=13, alternative="greater") # Right-sided
t.test(pinot$Quality, mu=12, conf.level=0.9) # Same p-value, just show different conf.level
Calculate manually:
test_statistic = (mean(pinot$Quality) - 13) / (sqrt(var(pinot$Quality) / length(pinot$Quality)))
p-value = pt(test_statistic, length(pinot$Quality) - 1)  # Left-sided
```

```
proc univariate data=work.pinot mu0=13;
        var Quality;
        /* Then read the table Test for location -> Student's t. Note it is default 2-sided */
run;
```

```
# To get 1-sided left or right:
proc univariate data=work.pinot mu0=13;
        var Quality;
        ods output TestsForLocation=TestsForLocation;
        /* Then read the table Test for location -> Student's t. Note it is default 2-sided */
run;
```

```
data result;
        set TestsForLocation(obs=1);
        one_sided_left_pvalue=cdf('T', Stat, 37); /* DF is n-1 */
        one_sided_right_pvalue=1-cdf('T', Stat, 37); /* DF is n-1 */
        put one_sided_left_pvalue=one_sided_right_pvalue=;
run;

/* Run this independently to get the p-value */
proc print data=result;
        var one_sided_left_pvalue;
        var one_sided_right_pvalue;
run;
```

4. See p-value as above.
5. Reject H0 if p-value is small.

One Sample Sign Test (Non-parametric, quantitative)

1. No need assume sample from normal. No need to have large sample size.
2. Hypothesis: H_0: Population Median = 3.3 vs H_1: Population Median < 3.3
   - A sample of size n is randomly collected from population: $x_1, x_2, ..., x_n$ that is skewed. We want to test

     $$H_0 : \text{population median} = m_0 \quad \text{vs} \quad H_1 : \text{population median} \neq m_0$$

   - We'll assign the sign (+ or -) to each data point: if $x_i > m_0$ then x_i has + sign; if $x_i < m_0$ then x_i has - sign; if $x_i = m_0$ then no sign is given and the sample size will be reduced. if datapoint == m0 then exclude it
   - The total number of positive sign is counted, $V+$; and the total number of negative sign is $V-$.
   - We then take the test statistic $V = \min(V+, V-)$.
   - This test statistic follow Binomial distribution $Bin(n^*, 0.5)$. $n^* = n$ - number of data points that are equal to m_0.
   - p-value is then calculated (2 tails probability for the 2 sided test).
3. Compute test statistics

H0 = Median is 3.3 vs H1: Median < 3.3

```
import scipy.stats as sc
success_weight = baby.weight[baby.weight < 3.3]
failure_weight = baby.weight[baby.weight > 3.3]
p_value = sc.binom_test(x=[len(success_weight), len(failure_weight)], p=0.5, alternative="less") #
alternative="less" "greater" "two-sided"
print(p_value)
```

```
success.weight = baby$weight[baby$weight < 3.3]
failure.weight = baby$weight[baby$weight > 3.3]
binom.test(x = c(length(success.weight), length(failure.weight)), alternative="less")
```

```
proc univariate data=work.pinot mu0=13;
        var Quality;
        /* Then read the table Test for location -> Sign. Note it is default 2-sided. If it is 1.0
then too bad =( */
run;
```

4. Calculate p-value as above.
5. Conclude.

Wilcoxon Signed Rank Test One Sample (Non-parametric, quantitative)

1. No need assume sample from normal. No need have large sample.
2. Hypothesis: H_0: Population Median = 3.3 vs H_1: Population Median < 3.3
3. Compute test statistic
   - Hypotheses:

     $$H_0 : \text{population median} = m_0 \quad \text{vs} \quad H_1 : \text{population median} \neq m_0$$

   - The difference of data point and m_0 is calculated $x_i - m_0$ and give the sign accordingly, they are $d_i(+)$ or $d_i(-)$.
   - $V+$ is the sum of all the positive differences. $V-$ is the sum of all the negative difference.
   - The idea behind the test is that if $V+ \approx V-$ then we have evidence supporting H_0.
   - If $V+$ is much greater (lesser) than $V-$, then we have evidence that the median is greater (lesser) than the hypothesised value.
   - This test is stronger than the sign test.

rank and sign

rank

rank

Test statistic is V+

Example:
x1, x2, ... x10 = (1,2,3,4...10)
Then, x1 will be assigned -9 becos x1-m0 = -9
x2 assigned -8
...
x8 assigned 0, ignored
x9 assigned +1
x10 assigned +2
Then, we take absolute value of them to find the rank. Rank is smallest to largest. So e.g. x9=1 is rank 1, x10=2 is rank 2, .. x2=|8| = rank 8, x1=|9| = rank 9. Assuming no tie in ranking. Then, we assign sign back to them again, so x10=rank +2, x9= rank+1, x1=rank -9. Then, V+ is sum of all positive rank to be the test statistic.

```
import scipy.stats as sc
m0 = 3.3
# 2-sided:
test_stastic, p_value = sc.wilcoxon(x=baby.weight-m0, y=None, zero_method='wilcox',
correction=False)
# Left-tail:
# test_stastic, p_value = sc.wilcoxon(x=baby.weight-m0, y=None, zero_method='wilcox',
correction=False, alternative="less")
# Right-tail"
# test_stastic, p_value = sc.wilcoxon(x=baby.weight-m0, y=None, zero_method='wilcox',
correction=False, alternative="greater")
```

```
print(test_stastic)
print(p_value)
```

```
weight.non.0 = baby$weight[baby$weight != 0]
wilcox.test(weight.non.0, mu=3.3, alternative="less")
```

```
proc univariate data=work.pinot mu0=13; /* Change the param */
        var Quality;
        /* Then read the table Test for location -> Signed Rank. Note it is default 2-sided. If it is 1.0 then too bad =( */
run;
```

Two Sample t-test (Parametric, independent)

1. Assuming two independent random samples. Two sample variances are the same(else use equal_variance = False). Population distribution of each group is approx. normal or n is large.
2. Hypothesis H0: u1 = u2, or u1 – u2 = 0. Where u1 and u2 is the population mean of group 1 and group 2. H1: u1 != u2, or u1 – u2 !=(<, >) 0.
3. Check if variances are equal first, then compute test statistic.
   - Let us represent the sample from group 1 as $X_1, X_2, \ldots, X_{n_1}$ and the sample from group 2 as $Y_1, Y_2, \ldots, Y_{n_2}$.
   - We shall denote the sample mean from group 1 as $\bar{X}$ and the sample mean from group 2 as $\bar{Y}$. Note that n_1 and n_2 need not be equal.
   - The point estimate of the difference between the population means is

     $$\bar{X} - \bar{Y}$$

   - Let us denote the sample variance from group 1 as s_1^2 and the sample variance from group 2 as s_2^2.
   - Recall that this test, we have assumed that the population variances of the two groups are equal. If we denote this common value by σ^2, then we can use the data from both samples to estimate it. We shall call this the pooled estimate of the common variance:

     $$s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$$

   - The test statistic is the distance between the point estimate and the null hypothesis value of the difference between population means, which is 0.
   - This distance is measured in terms of standard error:

     $$T = \frac{(\bar{X} - \bar{Y}) - 0}{se}$$

     0 is value under H0

     where the standard error is computed as

     $$se = s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

     se assumes the 2 samples have similar variance for this test.

   - If H_0 is true, then T follows a t-distribution with $n_1 + n_2 - 2$ degrees of freedom.
4. Compute test statistic. If H0 is true, T~t(n1 + n2 – 2)

```
import scipy.stats as sc
protein = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/protein_and_weight_gain.csv")
x = protein.weight_gain[protein.level == "high"]
y = protein.weight_gain[protein.level == "low"]
# Test to see variance are equal or not
barlett_statistic, barlett_pvalue = sc.bartlett(x, y) # P-value high indicates equal variance. Then we
can proceed.
print(barlett_statistic)
print(barlett_pvalue)

# Conduct t-test
# 2-sided test
test_statistic, p_value = sc.ttest_ind(x, y, axis=0, equal_var=True)
df = len(x) + len(y) - 2 # Degree of freedom
one_sided_left_p_value = sc.t.cdf(test_statistic, df)
one_sided_right_p_value = 1 - sc.t.cdf(test_statistic, df)
print("Test statistic: ", test_statistic)
print("2-sided test P-value:", p_value)
print("1-sided-left test P-value:", one_sided_left_p_value)
print("1-sided-right test P-value:", one_sided_right_p_value)



import statistics as st
## Calculate confidence interval
df = len(x) + len(y) - 2
xbar_minus_ybar = np.mean(x) - np.mean(y)
alpha = 0.05 # 95% confidence interval
half_alpha = alpha / 2
sp2 = ((len(x) - 1) * st.variance(x) + (len(y) - 1) * st.variance(y)) / (len(x) + len(y) - 2)
se = np.sqrt(sp2) * np.sqrt((1/len(x)) + (1/len(y)))
q = (sc.t.ppf(1-half_alpha, df=df)) * se
conf_interval = [xbar_minus_ybar - q, xbar_minus_ybar + q]
print("95% confidence interval:", conf_interval)
```

```
## First, check the 2 variances. Must be similar for this test.
protein = read.csv("/Users/gan/Desktop/Mods/ST2137/data/protein_and_weight_gain.csv")
x = protein$weight_gain[protein$level == "high"]
y = protein$weight_gain[protein$level == "low"]
var.test(x, y) # p-value should be large
## Then compute test statistic and p-value
t.test(x, y, mu=0, var.equal=T) # indicate 2 variances same, then use pooled variance
## Compute manually:
sp2 = ((length(x) - 1)*var(x) + (length(y) - 1)*var(y)) / (length(x) + length(y) - 2)
se = sqrt(sp2)*sqrt((1/length(x)) + (1/length(y)))
test_statistic = ((mean(x) - mean(y)) - 0) / se
p_value = pt(test_statistic, df=(length(x) + length(y) - 2), lower.tail=F) * 2 # two-sided
```

```
/* Import data first */
PROC TTEST data=work.protein;
        var weight_gain;
        class level;
        /* Then read off pooled p-value if variance are similar. Folded F p-value is to check 2
variances similarity. p-value should be high for similar */
run;
```

To get 1 sided:

```
/* Import data first */
PROC TTEST data=protein alpha=0.1;
        var weight_gain;
        class level;
        ods output TTests=TTests;
        /* Then read off pooled p-value if variance are similar. Folded F p-value is to check 2
variances similarity. p-value should be high for similar */
run;

data result;
        set TTests;
        one_sided_left_pvalue=cdf('T', tValue, DF);
        one_sided_right_pvalue=1-cdf('T', tValue, DF);
        put one_sided_left_pvalue=one_sided_right_pvalue=;
run;

/* Run this independently to get the p-value */
proc print data=result;
        var one_sided_left_pvalue;
        var one_sided_right_pvalue;
run;
```

5. Conclude. If confidence interval contain 0, do not reject. Otherwise, reject H0. Also means p-value is small, reject.

Two Sample Mann-Whitney U-Test (Non-parametric)

1. Assumption. Two independent samples. Samples no need to be normally distributed. Sample sizes can be small. Sample can even have data that are categorical, or ranked.

- Mann-Whitney test is also can be called as **Wilcoxon Rank Sum test**.
- Let $X_1, ..., X_n$ be IID with cdf F, and $Y_1, ..., Y_m$ IID with cdf G.
- We consider the null hypothesis $H_0 : F = G$.
- We are interested in whether X values are on the whole larger than the Y values or vice–versa.

H0: The 2 populations (not mean) are equal, i.e. distributions of both populations are equal.
H1: The 2 populations are not equal.

2. Compute test statistic.

```
import scipy.stats as sc
test_statistic, p_value = sc.mannwhitneyu(x, y, use_continuity=True, alternative="two-sided")
print(test_statistic)
print(p_value)
```

```
with_breakfast = c(87, 96, 92, 84) # Exam score students who ate breakfast
no_breakfast = c(93, 83, 79, 73) # Exam score students who dont eat breakfast
wilcox.test(with_breakfast, no_breakfast)
```

```
proc npar1way data=work.protein wilcoxon;
        var weight_gain;
        class level;
run;
```

Two Dependent Samples (Paired t-Test, Parametric)

1. Assumption. 2 dependent variables. MuX – muY follow approx. normal dist, or large size.
2. Hypothesis. H0: $\mu_X - \mu_y = 0$ vs. H1: $\mu_X - \mu_y \; ! = 0$. muX and muY are the population mean.

```
before = [25, 25, 27, 44, 30, 67, 53, 53, 52, 60, 28] # ate breakfast
after = [27, 29, 37, 56, 46, 82, 57, 80, 61, 59, 43] # no breakfast

# 2-sided t-test
test_statistic, p_value = sc.ttest_rel(after, before, axis=0, nan_policy="propagate")

df = len(before) - 1 # Degree of freedom
one_sided_left_p_value = sc.t.cdf(test_statistic, df)
one_sided_right_p_value = 1 - sc.t.cdf(test_statistic, df)
print("Test statistic: ", test_statistic)
print("2-sided test P-value:", p_value)
print("1-sided-left test P-value:", one_sided_left_p_value)
print("1-sided-right test P-value:", one_sided_right_p_value)



import statistics as st
## Calculate confidence interval
alpha = 0.05 # 95% confidence interval
xbar_minus_ybar = np.mean(after) - np.mean(before)
dbar = np.array([after]) - np.array([before])
half_alpha = alpha / 2
se = sc.sem(dbar.flatten()) # std error
q = (sc.t.ppf(1-half_alpha, df=df)) * se
conf_interval = [xbar_minus_ybar - q, xbar_minus_ybar + q]
print("95% confidence interval:", conf_interval)
```

```
before = c(25, 25,27, 44,30,67, 53, 53,52, 60, 28)
after = c(27, 29, 37, 56, 46, 82, 57, 80,61,59,43)
t.test(after, before, mu = 0, paired = TRUE, conf.level = 0.9)
```

```
data platelet;
        input before after;
        datalines;
25 27
25 29
27 37
44 56
```

```
30 46
67 82
53 57
53 80
52 61
60 59
28 43
;
        /* 2-sided read off Pr > |t| */
proc ttest data=platelet alpha=0.05;
        paired after*before;
run;
```

To get 1-sided:

```
/* 1-sided-left we store t value and run cdf */
proc ttest data=platelet;
        paired after*before;
        ods output TTests=TTests;
run;

data result;
        set TTESTS;
        one_sided_left_pvalue=cdf('T', tValue, DF);
        one_sided_right_pvalue=1-cdf('T', tValue, DF);
        put one_sided_left_pvalue=one_sided_right_pvalue=;
run;

/* Run this independently to get the p-value */
proc print data=result;
        var one_sided_left_pvalue;
        var one_sided_right_pvalue;
        var Probt; /*2-sided */
run;
```

Non-Parametric Paired Sample

1. Can be used for data small, non-normal data.

```
drugA = np.array([20, 40, 30, 45, 19, 27, 32, 26])
drugB = np.array([18, 36, 32, 46, 15, 22, 29, 25])
diff = drugA - drugB


# Signed Test
p_value = sc.binom_test([(diff < 0).sum(), (diff > 0).sum()], p=0.5)
print(p_value)
```

```
# Wilcoxon Signed Rank Test
test_statistic, p_value = sc.wilcoxon(x=diff, y=None, zero_method='wilcox', correction=True,
alternative='two-sided')
print("Test statistic:", test_statistic)
print("P-value:", p_value)
```

```
drugA = c(20, 40, 30, 45, 19, 27, 32, 26)
drugB = c(18, 36, 32, 46, 15, 22, 29, 25)
diff = drugA - drugB
ncount <-sum(sign(diff[diff>0]))# the number of positive signs
binom.test(ncount, length(diff),0.5) # Signed Test
wilcox.test(diff) # Wilcoxon Signed Rank Test
```

```
data drug;
input DrugA DrugB;
datalines;
20 18
40 36
30 32
45 46
19 15
27 22
32 29
26 25
;

data drug;
set drug;
diff = DrugA - DrugB;
run;

/* Look for signed rank pr */
proc univariate data=drug mu0=0;
var diff;
run;
```

Expectation of Sum of squares (SSD)

I = Number of treaments/groups
J = Number of measurements in each group
For groups with same size, i.e. same J:

$$SS_{TOT} = SS_W + SS_B = \sum_{i=1}^{I}\sum_{j=1}^{J}(Y_{ij} - Y_i)^2 + J\sum_{i=1}^{I}(Y_i - overallAve)^2$$

$$E(SS_W) = I(J-1)\sigma^2$$

$$E(SS_B) = J\sum_{i=1}^{I}\alpha_i^2 + (I-1)\sigma^2$$

$$E(Y_{ij}) = E(Y_i) = \mu + \alpha_i$$

Unbias estimate $s_p^2 = \frac{SS_W}{I(J-1)}$

$SS\_W = \sum_{i=1}^{I}(J-1)s_i^2$, where s_i^2 is the sample variance in the i-th group.

ANOVA

1. Assumption: Random samples. Equal variance (different group share the same variance, if unequal variance, use Kruskal straight). We check normality of response (check individually for each group) first before proceed (if sample large but not normal, also can). Then, after anova we check normality of errors. Independence of errors.
   Use barlette or levene to test variance, QQPlot or shapiro to test normality.
2. H0: $\alpha_1 = \alpha_2 = \cdots = \alpha_I$
3. Test statistic F = $\frac{SS_B/(I-1)}{SS_W/[I(J-1)]} \sim F_{df_1,df_2}$, $df_1 = (I-1)$,
   $df_2 = I(J-1)$ $\{for\ different\ group\ size, df2 = n - I, where\ n\ is\ total\ samples\ in\ all\ groups\}$
   Under H0, F should be close to 1.
   If H0 is false, number of F reflects variation between different groups and variations within groups, denominator reflects only variations within groups.
   H0 is rejected for large values of F.

```
newtablets$lab = as.factor(newtablets$lab)
model1 = aov(amount~lab, data=newtablets) # amount and lab are cols in newtablets, dep~ind
summary(model1)
anova_vector = unlist(summary(model1))
df1 = anova_vector[1]
df2 = anova_vector[2]
SS_B = anova_vector[3]
SS_W = anova_vector[4]
test_statistic = anova_vector[7]
p_value = anova_vector[9]
```

```
> anova<-aov(amount~lab, data = newdata)
> summary(anova)
            Df Sum Sq Mean Sq F value   Pr(>F)
lab          6 0.1247 0.020790    5.66 9.45e-05 ***
Residuals   63 0.2314 0.003673
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

SS_B

SS_W

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

```
newtablets.lab = newtablets.lab.astype(object) # declare lab as a factor
model = ols('amount~lab', data=newtablets).fit()  # amount and lab are col names of newtablets
anova = sm.stats.anova_lm(model, typ=2)
print(anova)
p_value = anova.iloc[0, 3]
test_statistic = anova.iloc[0, 2]
SS_B = anova.iloc[0, 0]
SS_W = anova.iloc[1, 0]
df1 = anova.iloc[0, 1]
df2 = anova.iloc[1, 1]
print("P_value:", p_value)
print("Test-statistic F:", test_statistic)
print("Sum of square between (SSB):", SS_B)
print("Sum of square within (SSW):", SS_W)
print("df 1:", df1)
print("df 2:", df2)
```

```
proc anova data=mysql_table2;
class lab;
model amount = lab;
means;
run;
```

ANOVA Assumption checking

Normality Test:

H0: Sample that is tested follows normal distribution.

shapiro.test(newtablets$amount) # Reject low p-value indicating not normal.
anova_result = aov(amount~lab, data=newtablets)
shapiro.test(anova_result$res) # Check residuals normality, i.e. normality of the error. Should be large for ANOVA becos we assumed this to be normal when using ANOVA. If small, then might want consider using Kruskal-Wallis instead.

```
# Check variable amount normality
shapiro_test_statistic, shapiro_p_value = sc.shapiro(newtablets.amount)
print("Shapiro Test statistic:", shapiro_test_statistic)
print("Shapiro P_value", shapiro_p_value)


# Check residual error normality
```

```
newtablets.lab = newtablets.lab.astype(object)
model = ols('amount~lab', data=newtablets).fit()  # amount and lab are col names of newtablets
shapiro_test_statistic, shapiro_p_value = sc.shapiro(model.resid)
print("Shapiro Test statistic:", shapiro_test_statistic)
print("Shapiro P_value", shapiro_p_value)
```

```
/* look at test for normality section */
proc univariate data=tablets normal;
var amount;
histogram amount / normal; /* Optional */
qqplot / normal (mu=est sigma=est); /* Optional */
run;
```

To test for residual:

```
proc glm data=tablets;
title2 'Proc glm analysis';
class lab;
model amount = lab;
/* Store fitted values and fitted residuals in dataset 'amountfit'*/
output out=amountfit p=yhat r=resid; /*p is fitted value, r is residual */

proc univariate data=amountfit normal;
var resid;
run;
```

Equal Variance Test:

bartlett.test(newtablets$amount, newtablets$lab) # Assume sample normal

OR

install.packages("car")

library(car)

leveneTest(newtablets$amount, newtablets$lab) # Suitable if sample distribution unknown)

We want p-value to be high so as not to reject H0 = same variances

```
# Only used when samples assumed nomality.
bar_test, p_value = sc.bartlett(newtablets.amount[newtablets.lab == 1],
newtablets.amount[newtablets.lab == 2],
                newtablets.amount[newtablets.lab == 3], newtablets.amount[newtablets.lab == 4],
                newtablets.amount[newtablets.lab == 5], newtablets.amount[newtablets.lab == 6],
                newtablets.amount[newtablets.lab == 7])
print("Barlette Test statistic:", bar_test)
print("Barlette Test p_value:", p_value)
```

```
# Can be use for non-normal
bar_test, p_value = sc.levene(newtablets.amount[newtablets.lab == 1],
```

```
newtablets.amount[newtablets.lab == 2],
                newtablets.amount[newtablets.lab == 3], newtablets.amount[newtablets.lab == 4],
                newtablets.amount[newtablets.lab == 5], newtablets.amount[newtablets.lab == 6],
                newtablets.amount[newtablets.lab == 7])
print("Levene Test statistic:", bar_test)
print("Levene Test p_value:", p_value)
```

```
/* Levene test for equal variances */
proc anova data=tablets;
class lab; /* independent var */
model amount = lab; /* amount is dependent */
means lab / hovtest=levene alpha=0.05;
run;
/* Barlette test for equal variances */
proc anova data=tablets;
class lab; /* independent var */
model amount = lab; /* amount is dependent */
means lab / hovtest=bartlett alpha=0.05;
run;
```

Both:

```
# Use this to draw plot
mcheck=function(obj){
res=obj$resid
fit=obj$fitted
par(mfrow=c(2,1))
plot(fit,res,xlab="Fitted values",ylab="Residuals")
abline(h=0,lty=2)
qqnorm(res,datax = TRUE, xlab="Z scores",ylab="Residuals")
qqline(res,datax = TRUE, lty=2)
par(mfrow=c(1,1))
}
```

Kruskal-Wallis Test (Non-parametric ANOVA)

1. Assumption. Can violate ANOVA. Observations can be categorical or non-normal. Can have different variances. No need assume any distribution form. Assumed independent observation.

```
kruskal.test(newtablets$amount~newtablets$lab)
```

```
kruskal_test_statistic, p_value = sc.kruskal(newtablets.amount[newtablets.lab == 1],
newtablets.amount[newtablets.lab == 2], newtablets.amount[newtablets.lab == 3],
                newtablets.amount[newtablets.lab == 4])
```

```
print("Kruskal Test statistic:", kruskal_test_statistic)
print("P-value:", p_value)
```

```
proc npar1way data=mysql_table2 wilcoxon dscf;
class lab;
var amount;
run;
```

Multiple Comparison Tests:
Bonferroni

If we test k null hypothesis and we want α family error rate,
Each individual null hypothesis is tested at level $\frac{\alpha}{k}$
Give good results if k is not too large.
Can use for both ANOVA and Kruskal.

R:

```
pairwise.t.test(newtablets$amount, newtablets$lab, p.adjust.method = "bonf")
# Then, compare the p-value with family error rate.
```

```
import statsmodels.stats.multicomp as mc
comparison = mc.MultiComparison(newtablets.amount, newtablets.lab)
## IF All groups satisfy ANOVA, then use t-test for pairwise comparison
result, tb1, tb2 = comparison.allpairtest(sc.ttest_ind, method="bonf", alpha=0.05)
## IF group doesn't satisfy ANOVA (aka using Kruskal-W test), then use M.Whitney for pairwise comparison
result, tb1, tb2 = comparison.allpairtest(sc.mannwhitneyu, method="bonf", alpha=0.05)
print(result)
# The result:
# stat: Test-statistic
# pval: The p-value if conducting the 2-independent t-test with the 2 groups with equal variance
# pval_corr: The adjusted p-value we using to compare with our alpha=0.05
# reject: If False (pval_corr > alpha), the 2 groups have similar means.
```

```
/*
Doesn't give p-value. Rather, the *** represents those rejected, i.e. the groups are
significantly different from each other.
*/
proc anova data=mysql_table2;
class lab;
model amount = lab;
means lab / BON CLDIFF alpha=0.05;
run;
```

Tukey Test (Only used for those applicable to ANOVA, i.e. not Kruskal)

Generally better than Bonferroni's correction.
Let α be the family type 1 error rate for k tests.

R:
TukeyHSD(aov(amount~lab, data=newtablets), conf.level = 0.95) # Alpha=0.05
Then, compare value of 'p adj' to alpha and reject those below alpha. E.g. if lab 4-1 p adj is small, meaning lab4 and lab1 are significantly different from each other.

```
import statsmodels.stats.multicomp as mc

comparison = mc.MultiComparison(newtablets.amount, newtablets.lab)

tukey = comparison.tukeyhsd(alpha=0.05)

print(tukey.summary())

# Its rejecting those whose Confidence Interval doesn't contain 0.
```

```
/*
Doesn't give p-value. Rather, the *** represents those rejected, i.e. the groups are
significantly different from each other.
*/
proc anova data=mysql_table2;
class lab;
model amount = lab;
means lab / tukey CLDIFF alpha=0.05;
run;
```

Standard error

```
se = function(x) {
  return(sqrt(var(x) / length(x)))
}
```

Estimating Regressor

Simple model: $y = \beta_0 + \beta_1 x$
Y is response, x is Regressor aka explanatory.
Before fitting linear model, we check and make sure response is normal first.

1. Assumption: Relationship of X and Y are linear (check with scatter plot). Normality assumption for errors and constant variance. All errors are uncorrelated to each other. All regressors are uncorrelated.
2. H0: All coefficients (not counting intercept) in the model are 0
   H1: At least one coefficient is non-zero.
   Simple model: H0: B1 = 0, H1: B1 != 0

Using Built-in python:

Single:

```
import statsmodels.api as sm
weight = ex10.weight # response
n = len(weight)
inter = [1]*n
X = np.column_stack((inter, ex10.height))
model1 = sm.OLS(weight, X)
results1 = model1.fit()
print(results1.summary())

## Anova:
from statsmodels.formula.api import ols
import statsmodels.api as sm
mod  = ols('weight ~ height', data=ex10).fit()
anova1 = sm.stats.anova_lm(mod  , typ=2)
print(anova1)
```

Multiple:

```
import statsmodels.api as sm
weight = ex10.weight # response
n = len(weight)
inter = [1]*n
X = np.column_stack((inter, ex10.height, ex10.age))
model2 = sm.OLS(weight, X)
results2 = model2.fit()
print(results2.summary())

## Anova:
from statsmodels.formula.api import ols
import statsmodels.api as sm
mod2 = ols('weight ~ height + age', data=ex10).fit()
anova2 = sm.stats.anova_lm(mod2, typ=2)
print(anova2)
```

With indicators (Categorical):

```
ex10['gender'] = ex10['gender'].astype(object)
dummy = pd.get_dummies(ex10.gender).values # convert categorical into dummy
print(dummy)
print(dummy[:, 1]) # Choose this that sets M = 1 and F = 0
print(dummy[:, 0]) # Choose this that sets F = 1 and M = 0
import statsmodels.api as sm
weight = ex10.weight # response
n = len(weight)
inter = [1]*n
X = np.column_stack((inter, ex10.height, ex10.age, dummy[:, 1])) # X3 will be M=1 F=0
model3 = sm.OLS(weight, X)
results3 = model3.fit()
print(results3.summary())

# OR if we want to explicitly state which one is reference
import statsmodels.api as sm
crab['spine'] = crab['spine'].astype(object)
weight = crab.weight # response
n = len(weight)
s1 = np.zeros(n)
s1[(crab.spine == 1)] = 1
s2 = np.zeros(n)
s2[(crab.spine == 2)] = 1
print(s1)
inter = [1]*n
X = np.column_stack((inter, crab.width, s1, s2))
model3 = sm.OLS(weight, X)
results3 = model3.fit()
print(results3.summary())
```

With Interaction Term (e.g. height related with gender):

```
import statsmodels.api as sm
weight = ex10.weight # response
```

```
n = len(weight)
inter = [1]*n
ex10['gender'] = ex10['gender'].astype(object)
dummy = pd.get_dummies(ex10.gender).values # convert categorical into dummy
HG = ex10['height']*dummy[:, 1] # Height*Gender
X = np.column_stack((inter, ex10.height, ex10.age, dummy[:, 1], HG)) # X1: height, X2: age, X3: gender, X4: height*gender
model4 = sm.OLS(weight, X)
results4 = model4.fit()
print(results4.summary())
```

Fitted values:

```
print(results1.fittedvalues)
```

Raw Residuals:

```
print(results1.resid)
```

Standardized Residual:

```
analysis = results1.get_influence()
SR = analysis.resid_studentized_internal
print(SR)
```

Leverage:

```
analysis = results1.get_influence()
leverage = analysis.hat_matrix_diag
print(leverage)
```

Cook's distance:

```
analysis = results1.get_influence()
cooks_d, p = analysis.cooks_distance
print(cooks_d)
```

Predict and confidence:

```
newpoint = [1,27,1,0] # intercept, x1 value, x2 value, x3 value
y = results3.predict(newpoint) # predict out of sample
print(y)


predictions = results3.get_prediction(newpoint)
print(predictions.summary_frame(alpha=0.05))
```

Scatter Plot of Standardized Residuals vs Fitted values:

```
fitted_values = results1.fittedvalues
analysis = results1.get_influence()
SR = analysis.resid_studentized_internal
plt.scatter(fitted_values, SR)
plt.xlabel("fitted weight")
plt.ylabel("Standardized Residuals")
plt.title("SR vs Fitted")
plt.hlines(0, xmin=min(fitted_values), xmax=max(fitted_values))
plt.show()
```

QQPlot of Standardized Residual

```
import pylab
import scipy.stats as scst

analysis = results3.get_influence()
SR = analysis.resid_studentized_internal
scst.probplot(SR, dist="norm", plot=pylab)
pylab.title("QQ Plot")
pylab.ylabel("Standardized Residuals")
pylab.show()
```

OLS Regression Results

Dep. Variable:	weight	R-squared:	0.944
Model:	OLS	Adj. R-squared:	0.933
Method:	Least Squares	F-statistic:	84.45
Date:	Sat, 10 Oct 2020	Prob (F-statistic):	0.000256
Time:	00:03:13	Log-Likelihood:	-26.068
No. Observations:	7	AIC:	56.14
Df Residuals:	5	BIC:	56.03
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const (b0)	-592.6446	81.542	-7.268	0.001	-802.255	-383.034
x1 (b1)	11.1913	1.218	9.190	0.000	8.061	14.322

SE(B1)=sqrt(MS_res / S_xx)

B1/SE(B1) ~ t(df=5)

Omnibus:	nan	Durbin-Watson:	2.161
Prob(Omnibus):	nan	Jarque-Bera (JB):	0.741
Skew:	-0.033	Prob(JB):	0.690
Kurtosis:	1.407	Cond. No.	1.22e+03

The Anova table

```
from statsmodels.formula.api import ols

mod1 = ols('weight ~ height', data=data).fit()
anova1 = sm.stats.anova_lm(mod1, typ=2)
print(anova1)
```

Regression of sum of sq, i.e. get_SSr

	sum_sq	df	F	PR(>F)
height	11880.327238	1.0	84.450853	0.000256
Residual	703.387048	5.0	NaN	NaN

Sum of squared residuals, i.e. get_SSres

Write manually:

```
ex10 = pd.read_csv("/Users/gan/Desktop/Mods/ST2137/data/ex10_1.txt", sep=" ")
x = ex10.height
y = ex10.weight


# For  y=b0 + b1x model
# x is vectors of x, y is vectors of y
# x is regressor (explanatory)
def get_b1(x, y):
    return (np.sum(y * x) - ((np.sum(y) * np.sum(x)) / len(x))) / (np.sum(x**2) - (np.sum(x)**2 / len(x)))



def get_b0(x, y):
    return np.mean(y) - get_b1(x, y) * np.mean(x)



def get_Sxx(x):
    return np.sum((x - np.mean(x)) ** 2)



def get_Sxy(x, y):
    return np.sum(y * (x - np.mean(x)))



# Param x,y are vectors of x-y values
# Returned a vector fitted values of yhats based on OLS
def get_fitted_model_yhats(x, y):
    b1 = get_b1(x, y)
    b0 = get_b0(x, y)
    yhat = b0 + b1*x
    return yhat



# Get the raw residuals after having the fitted model, i.e. (y-yhat)
# Accept param of raw x and y and calculate the fitted model.
def get_individ_res(x, y):
    yhat = get_fitted_model_yhats(x, y)
```

```
    return y - yhat


# Return the Sum of squared residuals after having the fitted model, i.e. sum( (y-yhat)^2 )
# Accept param of raw x and y and calculate the fitted model.
def get_SSres(x, y):
    yhat = get_fitted_model_yhats(x, y)
    return np.sum((y - yhat) ** 2)

# Regression sum of squares, with df1 in simple model, df=k in multiple model with k coefficients(less intercept)
def get_SSr(x, y):
    return get_b1(x, y) * get_Sxy(x, y)

# Return the total sum of square errors. SS_T is fixed regardless of model
def get_SStotal(x, y):
    return get_SSr(x, y) + get_SSres(x, y)

# Return the residual mean square of simple model, i.e. sigmahat^2
def get_MSres(x, y):
    return get_SSres(x, y) / (len(x) - 2)  # n-2 becos simple model 2 coefficients. n - #regressors + 1 for multi.

def get_MSr(x, y):
    return get_SSr(x, y) / 1  # n-2 becos simple model 2 coefficients


# Return residual standard error or standard error of regression
def get_StdMSres(x, y):
    return np.sqrt(get_MSres(x, y))

# F-test test statistic ~ F(1, length(x) -2)
def get_F_statistic(x, y):
    return (get_SSr(x, y) / 1) / (get_SSres(x, y) / (len(x) - 2))

# If p-value small, reject H0 => implies x helps in explaning variability in y.
# If p-value small, fail to reject H0 => there is no linear relationship between y and x.
def get_Ftest_p_value(x, y):
    F0 = get_F_statistic(x, y)
    return 1 - sc.f.cdf(F0, 1, len(x) - 2)

def get_T_statistic(x, y):
    return get_b1(x, y) / np.sqrt(get_MSres(x, y) / get_Sxx(x))

# 2-sided ttest p-value
def get_Ttest_p_value(x, y):
    T0 = get_T_statistic(x, y)
    return sc.t.cdf(np.abs(T0)*-1, len(x) - 2) * 2

# Return the coefficient of determination, R^2
```

```
def get_r2(x, y):
    return get_SSr(x, y) / get_SStotal(x, y)
```

Using built in R:
Single Model:
model1 = lm(weight~height, data=ex10) #Y~X
summary(model1)
anova(model1)

Multiple Model:
model2 = lm(weight~height+age, data=ex10)
summary(model2)
anova(model2)

With indicators (let R choose):
ex10$gender = as.factor(ex10$gender)
model3 = lm(weight ~ height + age + gender, data=ex10)
summary(model3)
anova(model3)

With indicators (we choose):
#Model 2: We choose the indicator for spine where spine = 3 is the reference.
s1 = c(rep(0,173)) #173 is nrow(crab)
s2 = c(rep(0,173))
s1[which(spine==1)] = 1
s2[which(spine==2)] = 1
data$s1 = s1
data$s2 = s2
model2 = lm(weight ~ width + s1 + s2, data = data)

With interaction term:
ex10$gender = as.factor(ex10$gender)
model4 = lm(weight ~ height + age + gender + height*gender, data=ex10)
summary(model4)
anova(model4)

Fitted values:
model1$fitted.values

Residuals & Standardize Residuals:
model1$res # Raw residuals of model1
rstandard(model1) # Standardized residuals of model 1
cooks.distance(model1)

Leverage:
x2 = cbind(c(rep(1, len(x))), x) # x is the regressor
hat = x2 %*% solve(t(x2) %*% x2) %*% t(x2)

leverage = diag(hat)

Predict:
newpoint=data.frame(width = 27,s1 = 1, s2 = 0)
# the 95% CI for the mean weight is:
predict(model2, newpoint, interval = "confidence", level = 0.95)

Plot Standardise Residual vs Fitted weight:
plot(model1$fitted.values, rstandard(model1), main="SR vs Fitted", xlab="fitted weight", ylab="Standadized Residuals")
x_range = seq(min(model1$fitted.values) - 5, max(model1$fitted.values) + 5, length.out=length(model1$fitted.values))
y_range = x_range * 0
lines(x_range, y_range)

Plot QQ Plot of Standardized Residuals:
SR = rstandard(model1)
qqnorm(SR,datax = TRUE, ylab = "Standardized Residuals", xlab = "Z scores", main = "QQ Plot", pch = 20)
qqline(SR,datax = TRUE)

```
Residuals:
      1       2       3       4       5       6       7
-13.361   8.977   2.595  14.256  -9.553  -9.788   6.873

Coefficients:                    Fitted model: Weight = -592.645 + 11.191 * height
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -592.645     81.542  -7.268 0.000771 ***
height        11.191      1.218   9.190 0.000256 ***
---                  SE(B1)=sqrt(MS_res / S_xx)  B1 / SE(B1) ~ t(df = n - #Regressor + 1)
                                                 p-value of F or T-test
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
                3 stars = p-value <= 0.001, 2 stars = p-value <= 0.01, 1 star = p-value <= 0.05
                    sqrt(MS_res)
Residual standard error: 11.86 on 5 degrees of freedom
Multiple R-squared:  0.9441,        Adjusted R-squared:  0.9329
F-statistic: 84.45 on 1 and 5 DF,  p-value: 0.000256

Response: weight
          Regression of sum of sq, i.e. get_SSr
          Df  Sum Sq Mean Sq F value   Pr(>F)
height     1 11880.3 11880.3  84.451 0.000256 ***
Residuals  5   703.4   140.7
---          Sum of squared residuals, i.e. get_SSres
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residuals:
      1       2       3       4       5       6       7
  1.252   2.281   6.232   5.877 -12.783  -8.513   5.654

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -254.1068   277.3604  -0.916    0.427
height         5.2309     4.6831   1.117    0.345
age            1.5870     0.9204   1.724    0.183
genderM       15.6511    23.2239   0.674    0.549
Residual standard error: 10.77 on 3 degrees of freedom
Multiple R-squared:  0.9723,    Adjusted R-squared:  0.9447
F-statistic: 35.16 on 3 and 3 DF,  p-value: 0.007742
```

To interpret coefficient 5.2309: At the same age and same gender, when the height increases by 1 unit, the mean weight (response) will increase by 5.2309 units

Fitted model: Weight_hat = -254.1068 + 5.2309*H + 1.5870*Age + 15.6511*Indicator(Gender==M)

Indicator(Gender==M) = 1 if Male, 0 if Female (Chosen by R, as seen in genderM)

This 15.6511 tells us the diff betw mean weight of male and female given they have same height and age

Calculate manually:

```
# For  y=b0 + b1x model
# x is vectors of x, y is vectors of y
# x is regressor (explanatory)
get_b1 = function(x, y) {
 return((sum(y*x) - ((sum(y)*sum(x))/length(x)) ) / ( sum(x^2) - (sum(x)^2 / length(x))))
}

get_b0 = function(x, y) {
 return(mean(y) - get_b1(x, y)*mean(x))
}

get_Sxx = function(x) {
 return(sum((x - mean(x))^2))
}

get_Sxy = function(x, y) {
 return(sum(y * (x - mean(x))))
}


# Param x,y are vectors of x-y values
# Returned a vector fitted values of yhats based on OLS
get_fitted_model_yhats = function(x, y) {
 b1 = get_b1(x, y)
 b0 = get_b0(x, y)
 yhat = b0 + b1*x
 return(yhat)
}


# Get the raw residuals after having the fitted model, i.e. (y-yhat)
# Accept param of raw x and y and calculate the fitted model.
get_individ_res = function(x, y) {
 yhat = get_fitted_model_yhats(x, y)
 return((y - yhat))
}


# Return the Sum of squared residuals after having the fitted model, i.e. sum( (y-yhat)^2 )
# Accept param of raw x and y and calculate the fitted model.
get_SSres = function(x, y) {
```

```
 yhat = get_fitted_model_yhats(x, y)
 return(sum((y - yhat)^2))
}

# Regression sum of squares, with df1 in simple model, df=k in multiple model with k coefficients(less intercept)
get_SSr = function(x, y) {
 return(get_b1(x, y) * get_Sxy(x, y))
}

# Return the total sum of square errors. SS_T is fixed regardless of model.
get_SStotal = function(x, y) {
 return(get_SSr(x, y) + get_SSres(x, y))
}

# Return the residual mean square of simple model, i.e. sigmahat^2
get_MSres = function(x, y) {
 return(get_SSres(x, y) / (length(x) - 2)) # n-2 becos simple model 2 coefficients. n - #regressor + 1 (intercept) for multi.
}

get_MSr = function(x, y) {
 return(get_SSr(x, y)/1) # n-2 becos simple model 2 coefficients
}

# Return residual standard error or standard error of regression
get_StdMSres = function(x, y) {
 return(sqrt(get_MSres(x, y)))
}

# F-test test statistic ~ F(1, length(x) -2)
get_F_statistic = function(x, y) {
 return((get_SSr(x, y)/1) / (get_SSres(x, y)/(length(x)-2)))
}

# If p-value small, reject H0 => implies x helps in explaning variability in y.
# If p-value small, fail to reject H0 => there is no linear relationship between y and x.
get_Ftest_p_value = function(x, y) {
 F0 = get_F_statistic(x, y)
 return(pf(F0, 1, length(x)-2, lower.tail=F))
}

get_T_statistic = function(x, y) {
 return(get_b1(x, y) / sqrt(get_MSres(x, y)/get_Sxx(x)))
}

# 2-sided ttest p-value
get_Ttest_p_value = function(x, y) {
 T0 = get_T_statistic(x, y)
 return(pt(abs(T0), length(x)-2, lower.tail=F) * 2)
}

# Return the coefficient of determination, R^2
get_r2 = function(x, y) {
 return(get_SSr(x, y) / get_SStotal(x, y))
}
```

Single:

```
proc reg data=ex10;
model weight = height; /* Y (response) = X (Explanatory/Regressor) */
output out=analysis P=yhat R=residual STUDENT=resid cookd=cooks H=leverage;
/* P=fitted, R=raw residuals, student is standadized residuals; */
```

```
run;
```

Multiple:

```
proc reg data=ex10;
model weight = height age/SS1;
run;
quit;
*/ SS1 is the SS_R (Type I SS) as in anova table in R */
```

With indicator (categorical variable):

```
data ex10_gen;
set ex10;
if gender = "M" then gen = 1;
if gender = "F" then gen = 0;
run;
/* when fitting categorical var into model, use this gen instead of raw gender */

proc reg data=ex10_gen;
model weight = height age gen/SS1;
run;
quit;
```

OR easier:

```
proc glm data=ex10;
  class gender;          /* generates dummy variables internally. Can add more */
  model weight = height age gender / solution;
run;
quit;
```

OR with graphs and no need auto set categorical

```
proc glmmod data=ex10 outdesign=GLMDesign outparm=GLMParm;
  class gender;
  model weight = height age gender;
run;

/* regression analysis by using dummy variables */
proc reg data=GLMDesign;
  DummyVars: model weight = COL2-COL5; /* dummy variables except intercept */
quit;
```

With interaction term (height related to gender):

```
data ex10;
set ex10;
if gender = "M" then gen = 1;
if gender = "F" then gen = 0;
height_gen = height*gen;
run;

proc reg data=ex10;
model weight = height age gen height_gen;
```

```
run;
quit;
```

Predicted and confidence:

```
/* Alr have columns s1 and s2 */
data crab;
set crab end=last;
output;
if last then do;
   VAR1 = . ;
   color = . ; *information for color is missed;
   spine = .; *spine variable is not needed for the prediction;
   width = 27; *width is given, here it = 27cm;
   satell =. ;
   weight =. ;
   s1 = 1 ; *spine = 1 means s1 = 1 and s2 = 0;
   s2 = 0 ;
   output;
end;
run;

*Using the model to predict the weight when width = 27 cm and spine = 1, output is saved in
table "predict";
proc reg data=crab alpha = 0.05;
  model weight = width s1 s2;
output out=predict(where=(weight=.)) p=predicted uclm=UCL_Pred lclm=LCL_Pred;
run;
quit;
* Use lclm - ulcm: lower and upper bounds of the CI for the prediction of Mean weight;
*Check the dataset "Predict" under the My Libararies --> WORK folder in the LEFT side;
*Predicted weight and 95% CI is: 2.6520425038       (2.5658029279,          2.7382820798);
```

Test Significance of Model (F-test)

1. Assumption: Relationship between X and Y is linear (check by using scatter plot of x and y). 2nd assumption is normality assumption and constant variance assumption, for $\in_i \sim N(0, \sigma^2)$, where variance is constant and most of time unknown. 3rd assumption is $\in_i, \in_j$ uncorrelated for all i != j. For model > 1 regressor, all regressors are uncorrelated.
2. Hypothesis:
   H0: All coefficients in the model are 0. For simple model: $H_0: \beta_1 = 0$, i.e. model is not significant
   H1: At least one coefficient is non-zero. For simple model, $H_1: \beta_1 \neq 0$
3. Test statistic:
   F = $\frac{SS_R/k}{SS_{Res}/(n-k-1)} = \frac{MS_R}{MS_{Res}} \sim F_{k,n-k-1}$
   Note: Degree of freedom only for simple model df1=1, df2=n-2
   k = number of regressor (exclude intercept) in the model. K=1 for simple.

4. P-value of F test stastitic is right area under F(k, n-k-1).
5. Reject H0 if p-value less than alpha. Fail to reject H0 implies there is no linear relationship between y and x. If reject H0, it implies x helps in variability in y, i.e. straight line model is adequate, or a better model could be obtained by adding higher order term(s) of x. In multi, rejection of H0 implies at least one regressor contributes significantly to the model.
6. To get calculation, refer to above Estimating Regressor

Test significance of regressor (T-test)

1. Assumption same as F-test above.
2. $H_0: \beta_1 = 0$ vs $H_1: \beta_1 \neq 0$
3. Test statistic:

   T = $\frac{\beta_1 hat}{SE(\beta_1 hat)} = \frac{\beta_1 hat}{\sqrt{(MS_{res}/S_{xx})}} \sim t_{n-2}$ for simple model

   T = $\frac{B_i hat}{SE(B_i hat)} = \frac{B_i hat}{\sqrt{(MS_{res} * C_{ii})}} \sim t_{n-k-1}$ for multi, where c_{ii} is the diagonal element of $(X'X)^{-1}$ corresponding to $B_i hat$. This is a test of contribution of x_i given the other regressors in the model

   Note: T^2 = F

   To get calculation, refer to above Estimating Regressor.

Coefficient of Determination R^2

$$R^2 = \frac{SS_R}{SS_T} = 1 - \frac{SS_{Res}}{SS_T}$$

0 <= R^2 <= 1. The larger R^2, the better fitted model is. Does not indicate appropriateness of linear model.

To get calculation, refer to above Estimating Regressor.

Adjust R^2:

$$R^2_{Adj} = 1 - \frac{SS_{Res}/(n-p)}{SS_T/(n-1)}$$

p = # Regressors (not counting intercept) + 1. i.e. total number of coefficients including intercept.

Here R^2 adjusted will drop if add regressor that is not significant, whereas R^2 will always increase if adding more regressors.

Confidence Interval

Determine 95% confidence interval for B1 of variable width

confint(model2, "width", level=0.95)

```
model3 = sm.OLS(weight, X)
results3 = model3.fit()
print(results3.summary())


print(results3.conf_int(alpha=0.05)) # 95% CI
```

```
/* Basically add alpha and clparam */
proc glm data=crab alpha=0.05; /* 95% */
  class spine;            /* generates dummy variables internally. Can add more */
  model weight = width spine / solution clparm;
run;
quit;

/* If normal reg, add alpha and clb */
proc reg data=ex10_gen alpha=0.05;
model weight = height age gen/SS1 clb;
run;
quit;
```

Various variable

Variance, sigma^2 of the error term:

```
model3 = sm.OLS(weight, X)
results3 = model3.fit()
print(results3.summary())


est_variance = results3.scale # variance of the error term
```

est_variance = summary(model2)$sigma ^ 2

```
* sigma = Root MSE = 0.26564;  /* Read off Root MSE */
*the estimation of variance of error term is = 0.26564^2;
```