**Hypothesis:** A hypothesis is an educated prediction that can be tested. A hypothesis is a specific, testable prediction. A hypothesis is used in an experiment to define the relationship between two variables.

**Null Hypothesis:**

* A null hypothesis is a hypothesis that says there is no statistical significance between the two variables.
* It is usually the hypothesis a researcher or experimenter will try to disprove or discredit.

**Alternative Hypothesis:**

* Analternative hypothesis is one that states there is a statistically significant relationship between two variables.
* The alternate hypothesis is simply the opposite of the null hypothesis

**Hypothesis Tests**

Although there are hundreds of statistical hypothesis tests that you could use, there is only a small subset that you may need to use in a machine learning project.

How to draw conclusions based on the test statistic and p-value:

We can interpret the test statistic with the requisite number of degres of freedom as follows:

* **If Statistic >= Critical Value**: significant result, reject null hypothesis (H0), dependent.
* **If Statistic < Critical Value**: not significant result, fail to reject null hypothesis (H0), independent.

In terms of a p-value and a chosen significance level (alpha), the test can be interpreted as follows:

* **If p-value <= alpha**: significant result, reject null hypothesis (H0), dependent.
* **If p-value > alpha**: not significant result, fail to reject null hypothesis (H0), independent.

**Normality Tests:** These statistical tests that can be used to check if the data has a Gaussian distribution or not. Normality check is important as many statistical

inference techniques assume that the data is following distribution. There are several methods of assessing whether data are normally distributed or not. They fall into two broad categories:

**Graphical :**

1. Q-Q plots
2. Cumulative frequency (P-P) plots
3. Probability plots

**Statistical:**

1. W/S test
2. Jarque-Bera test
3. Shapiro-Wilks test
4. Kolmogorov-Smirnov test
5. D’Agostino test

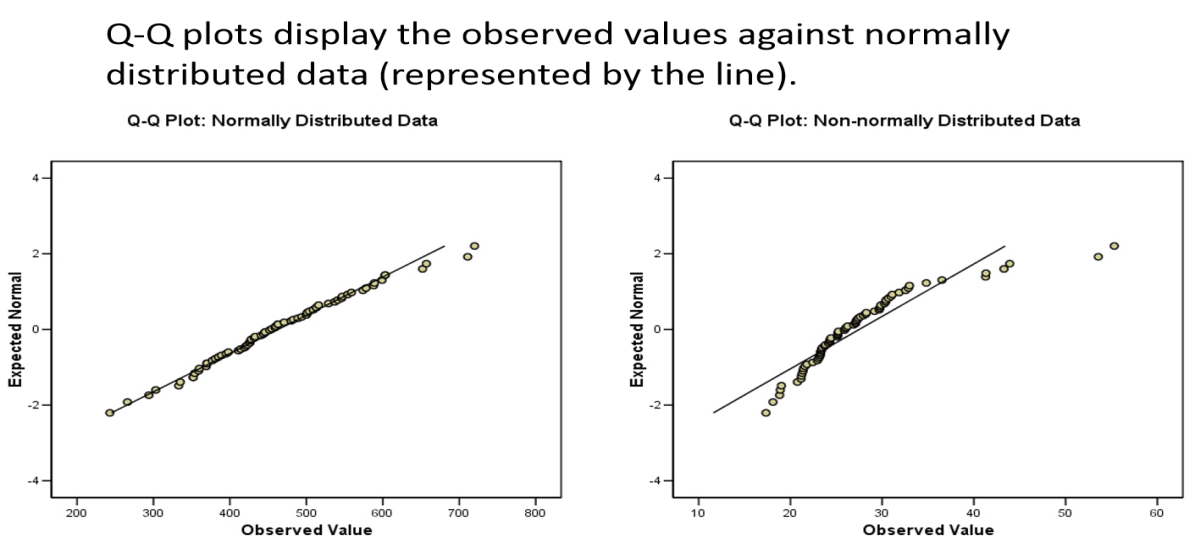
**Graphical methods for Normality check:** A number of graphical methods exist for comparing two probability distributions by plotting their quantiles, or closely related parameters, against each other.

1. **QQ-Plots:** The “Q” in QQ-plot stands for quantile. The quantiles of a given data set are plotted against the quantiles of a reference distribution, typically the standard normal distribution.

This plot generates its own samples of the Gaussian and divides them into groups, called quantiles. Each data point in the sample is paired with a similar member from the Gaussian distribution at the same cumulative distribution. The resulting points are plotted as a scatter plot with the idealized value on the x-axis and the data sample on the y-axis.

A perfect match for the distribution will be shown by a line of dots on a 45-degree angle from the bottom left of the plot to the top right.

Deviations by the dots from the line show a deviation from the Gaussian distribution.



We can develop a QQ plot in Python using the qqplot() statsmodels function. The function takes the data sample and by default assumes we are comparing it to a Gaussian distribution.

**Python Code:**

import numpy as np

import matplotlib.pyplot as plt

from numpy.random import randn

from statsmodels.graphics.gofplots import qqplot

data = 5 \* randn(50) + 50 # generate univariate observations

qqplot(data, line='s') # q-q plot

plt.show()

**b. PP-Plots:** Plot the CDF (cumulative-distribution-function) of a given data set against the CDF of a reference distribution.

A P-P plot compares the empirical cumulative distribution function of a data set with a specified theoretical cumulative distribution function F(·)

**Python code:**

import numpy as np

import matplotlib.pyplot as plt

from numpy.random import randn

import statsmodels.api as sm

data = 5 \* randn(50) + 50 # generate univariate observations

sm.ppplot(data, line='s')

plt.show()

1. **Probability Plots:** Plot the ordered values of a given data set against the quantiles of a reference distribution.

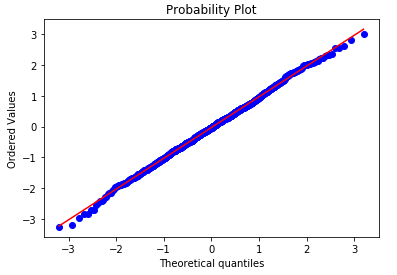
**Python Code:**

from scipy import stats

from scipy.stats import norm

data = norm.rvs(loc=0, scale=1, size=1000)

stats.probplot(data, plot=plt)



The above graphical methods gives only the quantitative measures and are difficult to interpret the results, but they are simple to generate

**Statistical based tests for normality check:** The other set of methods that can be used for normality check are the following which gives quantitative measures.

1. **Shapiro-Wilk Test:** Tests whether a data sample has a Gaussian distribution.

**Assumptions:** Observations in each sample are independent and identically distributed (iid).

**Interpretation:**

**H0:** the sample has a Gaussian distribution.

**H1:** the sample does not have a Gaussian distribution.

**Python Code:**

from numpy.random import seed

from numpy.random import randn

from scipy.stats import shapiro

seed(1) # seed the random number generator

data = 5 \* randn(100) + 50 # generate univariate observations

stat, p = shapiro(data) # Shapiro-Wilk Test # normality test

print('Statistics=%.3f, p=%.3f' % (stat, p))

# interpret

alpha = 0.05

if p > alpha:

print('Sample looks Gaussian (fail to reject H0)')

else:

print('Sample does not look Gaussian (reject H0)')

1. **D’Agostino’s K^2 Test:** Tests whether a data sample has a Gaussian distribution.

**Assumptions:** Observations in each sample are independent and identically distributed (iid).

**Interpretation:**

**H0:** the sample has a Gaussian distribution.

**H1:** the sample does not have a Gaussian distribution.

**Python Code:**

from numpy.random import seed

from numpy.random import randn

from scipy.stats import normaltest

seed(1) # seed the random number generator

data = 5 \* randn(100) + 50 # generate univariate observations

stat, p = normaltest(data) # normality test

print('Statistics=%.3f, p=%.3f' % (stat, p))

# interpret

alpha = 0.05

if p > alpha:

print('Sample looks Gaussian (fail to reject H0)')

else:

print('Sample does not look Gaussian (reject H0)')

1. **Anderson-Darling Test:** Tests whether a data sample has a Gaussian distribution.

**Assumptions:** Observations in each sample are independent and identically distributed (iid).

**Interpretation:**

**H0:** the sample has a Gaussian distribution.

**H1:** the sample does not have a Gaussian distribution.

**Python Code:**

from numpy.random import seed

from numpy.random import randn

from scipy.stats import anderson

seed(1) # seed the random number generator

data = 5 \* randn(100) + 50 # generate univariate observations

stat, p = anderson(data) # Anderson Darling Test # normality test

print('Statistics=%.3f, p=%.3f' % (stat, p))

# interpret

alpha = 0.05

if p > alpha:

print('Sample looks Gaussian (fail to reject H0)')

else:

print('Sample does not look Gaussian (reject H0)')

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Correlation Tests:** These are the statistical tests that can be used to check if two data samples are related or not. Correlation is a bivariate analysis that measures the strength of association between two variables and the direction of the relationship. A correlation could be :

Positive : both variables change in the same direction.

Neutral : No relationship in the change of the variables.

Negative : variables change in opposite directions.

Linear correlation is a measure of dependence between two random variables.

**Covariancce:** This relationship can be summarized between two variables, called the covariance. It is calculated as the average of the product between the values from each sample, where the values haven been centered (had their mean subtracted).

The calculation of the sample covariance is as follows:

cov(X, Y) = (sum (x - mean(X)) (y - mean(Y))) 1/(n-1)

**Python code:**

# generate related variables

from numpy import mean

from numpy import cov

from numpy.random import randn

# prepare data

data1 = 20 \* randn(1000) + 100

data2 = data1 + (10 \* randn(1000) + 50)

covariance = cov(data1, data2)

print(covariance)

**Problems with covariance:** One problem with covariance as a statistical tool alone is that it is challenging to interpret. This leads us to the introduction of the following correlation coefficients.

1. Pearson’s Correlation Coefficient
2. Spearman’s Rank Correlation
3. Kendall’s Rank Correlation

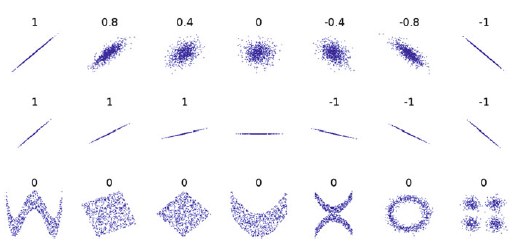


Figure: Several sets of .*x*; *y*/ points, with the correlation coefficient of *x* and *y* for each set.

The coefficient returns a value between -1 and 1 that represents the limits of correlation from a full negative correlation to a full positive correlation.

A value of 0 means no correlation. A value below -0.5 or above 0.5 indicates a notable correlation, Values below those values suggests a less notable correlation.

1. **Pearson’s Correlation Coefficient:** Tests whether two samples have a linear relationship.

Pearson's correlation coefficient = covariance(X, Y) / (stdv(X) \* stdv(Y))

**Assumptions:**

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample are normally distributed.
* Observations in each sample have the same variance.

**Interpretation**

* H0: the two samples are independent.
* H1: there is a dependency between the samples.

**Python Code:**

# # calculate the Pearson's correlation between two variables

from numpy.random import randn

from numpy.random import seed

from scipy.stats import pearsonr

seed(1) # seed random number generator

data1 = 20 \* randn(1000) + 100 # prepare data

data2 = data1 + (10 \* randn(1000) + 50) # prepare data

corr, \_ = pearsonr(data1, data2) # calculate Pearson's correlation

print('Pearsons correlation: %.3f' % corr)

1. **Spearman’s Rank Correlation:** Tests whether two samples have a monotonic relationship.

Spearman's correlation coefficient = covariance(rank(X), rank(Y)) / (stdv(rank(X)) \* stdv(rank(Y)))

**Assumptions:**

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample can be ranked.

**Interpretation:**

* H0: the two samples are independent.
* H1: there is a dependency between the samples.

The spearmanr() SciPy function can be used to calculate the Spearman’s correlation coefficient between two data samples with the same length.

**Python Code:**

# calculate the spearmans's correlation between two variables

from numpy.random import randn

from numpy.random import seed

from scipy.stats import spearmanr

seed(1) # seed random number generator

data1 = 20 \* randn(1000) + 100 # prepare data

data2 = data1 + (10 \* randn(1000) + 50) # prepare data

corr, \_ = spearmanr(data1, data2) # calculate spearman's correlation

print('Spearmans correlation: %.3f' % corr)

3. **Kendall’s Rank Correlation:** Tests whether two samples have a monotonic relationship.

**Assumptions:**

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample can be ranked.

**Interpretation:**

* H0: the two samples are independent.
* H1: there is a dependency between the samples.

**Python Code:**

from numpy.random import randn

from numpy.random import seed

from scipy.stats import kendalltau

seed(1) # seed random number generator

data1 = 20 \* randn(1000) + 100 # prepare data

data2 = data1 + (10 \* randn(1000) + 50) # prepare data

corr, p = kendalltau(data1, data2) # calculate kendal's correlation

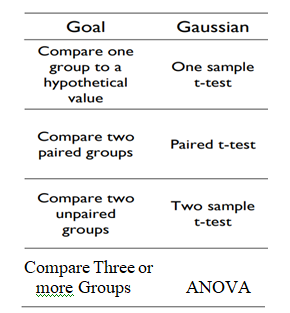
print('Kendals correlation: %.3f' % corr)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Parametric Statistical Hypothesis Tests:** These statistical tests can be used to compare data samples. These Hypothesis tests for comparing the mean values of groups and following are different type of comparisons:

* Comparison of one group with a ﬁxed value.
* Comparison of two groups with respect to each other.
* Comparison of three or more groups with each other

A one sample t-test is used to compare the mean of a sample to a known constant value value (often  0, but not always). A two sample t test is used to compare the means of two different samples.



1. **One- sample t-test:** The one-sample t-test compares a sample’s mean with a known value, when the variance of the population is unknown.

* The One Sample *t* Test is a parametric test.
* This test is also known as: Single Sample *t* Test
* The variable used in this test is known as: Test variable
* In a One Sample *t* Test, the test variable is compared against a "test value" (constant), which is a known or hypothesized value of the mean in the population.

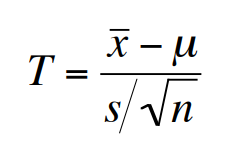
**Ex:** Consider we want to assess the percent of college educated adults in India and compare it to a certain value

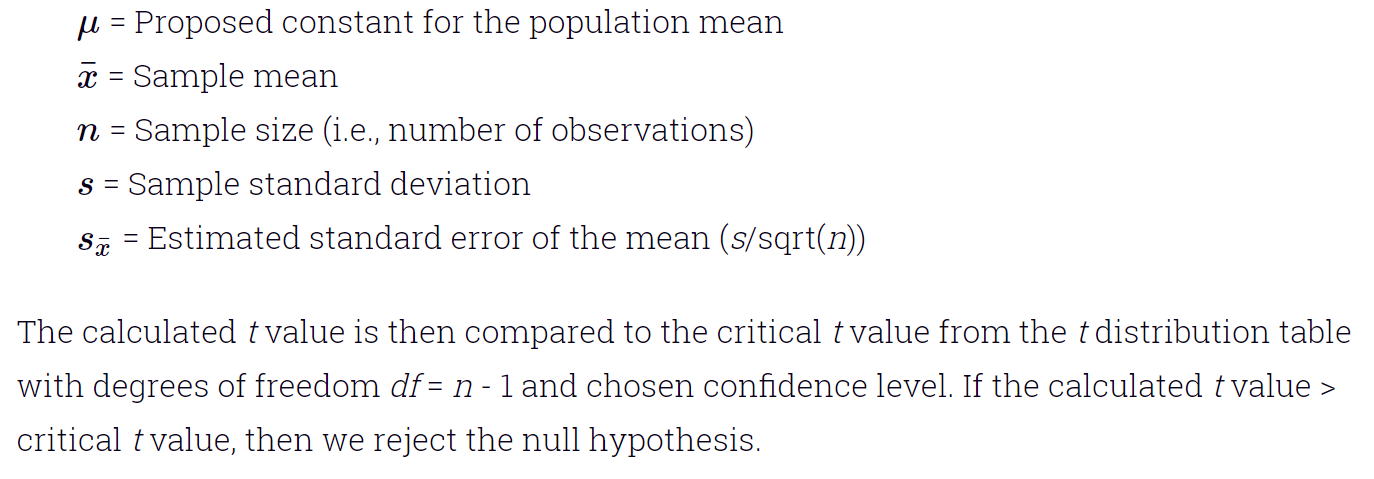
**Interpretation:** *H*0 and *H*1 of the one sample *T* test can be expressed as:

*H*0: µ = x  ("the sample mean is equal to the [proposed] population mean")  
*H*1: µ ≠ x  ("the sample mean is not equal to the [proposed] population mean")

where µ is a constant proposed for the population mean and x is the sample mean.

The test statistic for a One Sample *t* Test is denoted *t*, which is calculated using the following formula:



****

**Python Code:**

|  |  |
| --- | --- |
| import numpy as np  from scipy.stats import norm  from scipy import stats  np.random.seed(7)  x1=norm.rvs(loc=0,scale=1,size=100)  m=0  results = stats.ttest\_1samp(x1, m)  print(results)  Ttest\_1sampResult(statistic=0.11784116818175697, pvalue=0.9064322657112877) |  |

1. **Two- sample t-test:** Tests whether the means of two independent samples are significantly different.

Ex: compare the differences between the average percent of college educated adults in India versus United status.

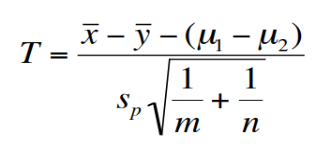
Assumptions

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample are normally distributed.
* Observations in each sample have the same variance.

Interpretation

* H0: the means of the samples are equal.
* H1: the means of the samples are unequal.

The test statistic for two Sample *t* Test is denoted *t*, which is calculated using the following formula:

****

**Assumptions:**

* Observations in the sample are independent and identically distributed (iid).
* Observations in the sample are normally distributed.

**Interpretation:**

* H0: the means of the samples are equal.
* H1: the means of the samples are unequal.

**Python code:**

import numpy as np

from scipy.stats import norm

from scipy import stats

np.random.seed(7)

x1=norm.rvs(loc=0,scale=1,size=100)

x2=norm.rvs(loc=0,scale=1,size=100)

stats.ttest\_ind(x1,x2)

test\_indResult(statistic=0.5691860234273843, pvalue=0.5698750154420147)

1. **Paired Student’s t-test:** A paired t-test is used to compare two population means where you have two samples in which observations in one sample can be paired with observations in the other sample.

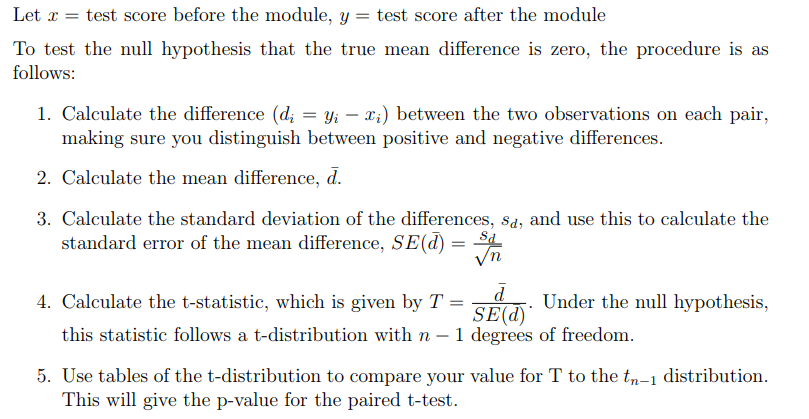
Examples of where this might occur are:

* Before-and-after observations on the same subjects (e.g. students’ diagnostic test results before and after a particular course).
* A comparison of two different methods of measurement where the measurements are applied to the same subjects (e.g. blood pressure measurements using a stethoscope and a dynamap).

**Assumptions:**

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample are normally distributed.
* Observations in each sample have the same variance.
* Observations across each sample are paired.

**Procedure to carry out a paired t-test:**

****

**Interpretation**

* H0: the means of the samples are equal.
* H1: the means of the samples are unequal.

**Python Code:**

import pandas as pd

from scipy.stats import ttest\_rel

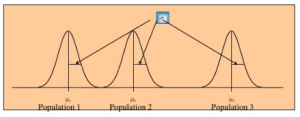
df = pd.read\_csv("blood\_pressure.csv")

df[['bp\_before','bp\_after']].describe()

df['bp\_difference'] = df['bp\_before'] - df['bp\_after']

stat, p = stats.ttest\_rel(df['bp\_before'], df['bp\_after'])

1. **Analysis of Variance Test (ANOVA): ANOVA** is a statistical tool used to compare population means of two or more groups.



**One way ANOVA:** Variance in the ANOVA is partitioned into total variance, variance due to groups, and variance due to individual differences.

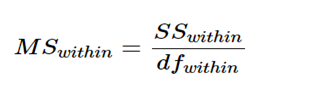
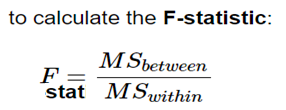
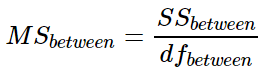
**Assumptions:**

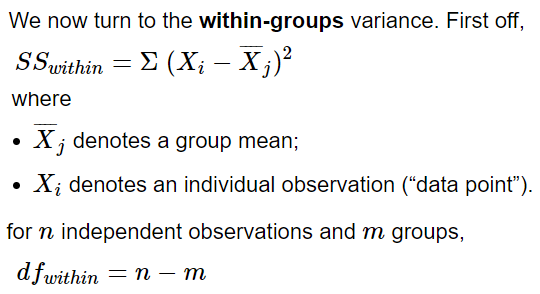
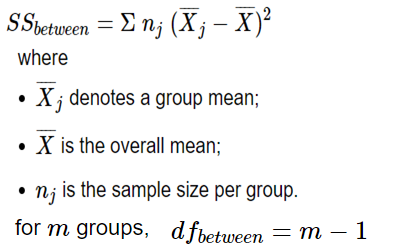
* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample are normally distributed.
* Observations in each sample have the same variance.

**Interpretation**

* **H0: the means of the samples are equal.**
* **H1: one or more of the means of the samples are unequal.**

**Process:** Sum of squares between expresses the total amount of dispersion among the sample means.

****

****

To reject the null hypothesis we check if the obtained Fstat-value is above the critical value for rejecting the null hypothesis. We could look it up in a F-value table based on the DFwithin and DFbetween.

**Now find F Critcal**

**Python Code:**

import pandas as pd

from scipy.stats import f\_oneway

datafile = "PlantGrowth.csv"

data = pd.read\_csv(datafile)

print(data.head())

ctrl = data['weight'][data.group == 'ctrl']

trt1 = data['weight'][data.group == 'trt1']

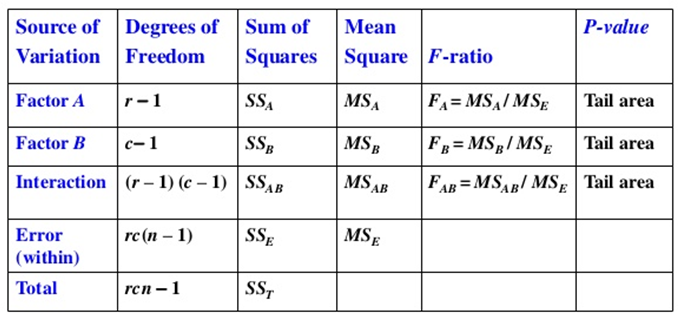
trt2 = data['weight'][data.group == 'trt2']

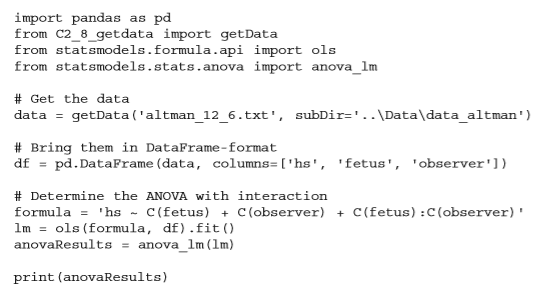
F, p = stats.f\_oneway(d\_data['ctrl'], d\_data['trt1'], d\_data['trt2'])

print(F, p)

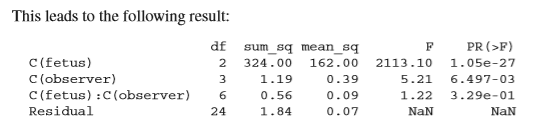
|  |  |
| --- | --- |
|  |  |

**Two-way ANOVA:** When the outcome or dependent variable (in our case the test scores) is affected by two independent variables/factors we use two-way ANOVA. The following table values have to be computed.



****

**Sample results:**

****

## My p-value is greater than 0.05, what do I do now?

Report the result of the one-way ANOVA (e.g., "There were no statistically significant differences between group means as determined by one-way ANOVA (F(2,27) = 1.397, p = .15)"). Not achieving a statistically significant result does not mean you should not report group means ± standard deviation also. However, running a post hoc test is usually not warranted and should not be carried out.

## My p-value is less than 0.05, what do I do now?

Firstly, you need to report your results as highlighted in the "How do I report the results of a one-way ANOVA?" section on the previous page. You then need to follow-up the one-way ANOVA by running a post hoc test.

**Post-hoc Testing**

The overall model was significant, now to test which groups differ. Deciding which groups to compare should be theory driven. There are a few different techniques that can be used. Each of these techniques have different ways of controlling for familywise error rate. 3 common methods are:

* + *Fisher’s Least Significant Difference (LSD)*: Take the groups you want to compare and conduct multiple t-tests. This method requires that the ANOVA model be significant. This method is easy, but receives push back since it doesn’t account for familywise error rate. The argument is that since the overall model was significant, one is protected from increasing the familywise error rate.
  + *Bonferroni correction*: Take the alpha the ANOVA was tested at, 0.05, then divide it by the number of planned comparisons. In this case, 0.05/3 = 0.0167. A post-hoc test would have to have an alpha level < 0.0167 to be considered significant. To test the groups, conduct multiple t-tests, but set the alpha value to the corrected value. This method is quick, but often considered too conservative.
* *Tukey’s HSD*: Method also controls for familywise error rate with a different method than Bonferroni, and is also considered conservative.

There are many other techniques out there that can be used for post-hoc testing each with different guidelines for when they should be used, you are encouraged to learn about them!

**4. Nonparametric Statistical Hypothesis Tests**

**a.Mann-Whitney U Test**

Tests whether the distributions of two independent samples are equal or not.

**Assumptions**

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample can be ranked.

**Interpretation**

* H0: the distributions of both samples are equal.
* H1: the distributions of both samples are not equal.

**Python Code**

|  |  |
| --- | --- |
| 1  2  3 | from scipy.stats import mannwhitneyu  data1, data2 = ...  stat, p = mannwhitneyu(data1, data2) |

**b. Wilcoxon Signed-Rank Test**

Tests whether the distributions of two paired samples are equal or not.

**Assumptions**

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample can be ranked.
* Observations across each sample are paired.

**Interpretation**

* H0: the distributions of both samples are equal.
* H1: the distributions of both samples are not equal.

**Python Code**

|  |  |
| --- | --- |
| **1**  **2**  **3** | from scipy.stats import wilcoxon  data1, data2 = ...  stat, p = wilcoxon(data1, data2) |

1. **Kruskal-Wallis H Test**

Tests whether the distributions of two or more independent samples are equal or not.

**Assumptions**

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample can be ranked**.**

**Interpretation**

* H0: the distributions of all samples are equal.
* H1: the distributions of one or more samples are not equal.

**Python Code**

|  |  |
| --- | --- |
| **1**  **2**  **3** | from scipy.stats import kruskal  data1, data2, ... = ...  stat, p = kruskal(data1, data2, ...) |

**Friedman Test**

Tests whether the distributions of two or more paired samples are equal or not.

**Assumptions**

* Observations in each sample are independent and identically distributed (iid).
* Observations in each sample can be ranked.
* Observations across each sample are paired.

**Interpretation**

* H0: the distributions of all samples are equal.
* H1: the distributions of one or more samples are not equal.

**Python Code**

|  |  |
| --- | --- |
| 1  2  3 | from scipy.stats import friedmanchisquare  data1, data2, ... = ...  stat, p = friedmanchisquare(data1, data2 |

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Frequency table:** A table summarization of categorical variables.

**Contingency table:** A frequency table summarized in the form of a table with one variable corresponding to columns and another variable corresponding to rows is called a [contingency table](https://en.wikipedia.org/wiki/Contingency_table).

**One-way Chi-Squared Test:** Pearson’s **Chi-square test (**χ2**)** tells us if the observed frequencies from a sample are consistent with a defined expected frequencies or not. The χ2**-statistic** is computed as follows:

 and 

Degrees of freedom (df) = Number of possible outcomes - 1

* **If Statistic >= Critical Value**: significant result, reject null hypothesis (H0), dependent.
* **If Statistic < Critical Value**: not significant result, fail to reject null hypothesis (H0), independent.

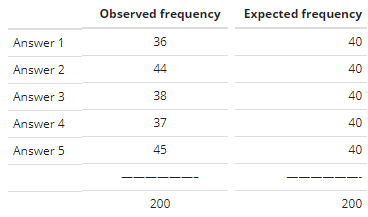
**Example:** A survey of **200** individuals is conducted where the respondents are asked to select one of five answers, one might want to determine if all answers have an equal chance of being chosen.

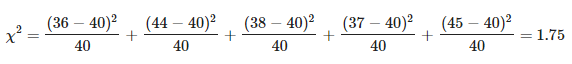
**Null hypothesis:** Each answer had an equal chance of being selected.

**Alternate hypothesis:** Each answer had an unequal chance of being selected.

Expected Frequency = 200 / 5 = 40

The observed and expected values can be summarized in a **frequency table**





df=5−1=4

p-value = 0.7816

Hance, We can conclude that our observed frequencies are consistent with our null hypothesis and that any variability can be explained by chance alone.

**Python code:**

from scipy.stats import chisquare

from scipy.stats import chi2

observed = [36, 44, 38, 37, 45]

expected = [40, 40, 40, 40, 40]

stat, pval = chisquare(observed, expected )

dof = 4

# interpret test-statistic

prob = 0.95

critical = chi2.ppf(prob, dof)

print('probability=%.3f, critical=%.3f, stat=%.3f' % (prob, critical, stat))

if abs(stat) >= critical:

print('Dependent (reject H0)')

else:

print('Independent (fail to reject H0)')

**output:** probability=0.950, critical=9.488, stat=1.750,

Independent (fail to reject H0)

**Two-way Chi-Squared Test:**  to determine the statistical significance of two independent categorical groups of data. That is to test whether two categorical variables are related or independent.

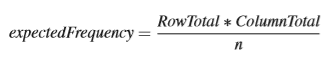
**Assumptions:**

* Observations used in the calculation of the contingency table are independent.
* 25 or more examples in each cell of the contingency table.

**Interpretation:**

* H0: the two samples are independent.
* H1: there is a dependency between the samples.

In order to determine whether we accept or reject the null hypothesis. We have to compute p-value, we use the Chi-squared test.

and 

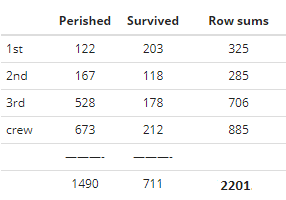
****

Where X2 is the test statistic, observed are values we have in the contingency table, expected are values we would expect assuming the null hypothesis is true.

**Example:** passengers/crew members of different class status (i.e. 1st, 2nd, 3rd or crew) equally likely to perish in the Titanic?

H0: There is no statistically significant relationship between class of travel and survival Ha: There is a statistically significant relationship between class of travel and survival

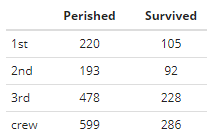
The next step is to format the data into a frequency count table. This is called a **Contingency Table**.

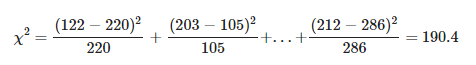
****

 For example, for row1/col1 (i.e. the total number of 1st class passengers that perished), the expected count assuming Ho is:



Expected table values are :



****

**Df = (r - 1) \* (c – 1) = (4 – 1) \* (2 – 1) = 3**

**Python Code:**

from scipy.stats import chi2\_contingency

from scipy.stats import chi2

data = np.array([[122, 167, 528, 673], [203, 118, 178, 212]])

stat, p, dof, expected = chi2\_contingency(data)

# interpret test-statistic

prob = 0.95

critical = chi2.ppf(prob, dof)

print('probability=%.3f, critical=%.3f, stat=%.3f' % (prob, critical, stat))

if abs(stat) >= critical:

print('Dependent (reject H0)')

else:

print('Independent (fail to reject H0)')

Output: probability=0.950, critical=7.815, stat=190.401

Dependent (reject H0)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Correlation**

**Correlation** is a bivariate analysis that measures the strength of association between two variables and the direction of the relationship.

Linear correlation is a measure of dependence between two random variables.

A correlation could be :

**Positive:** both variables change in the same direction.

**Neutral:** No relationship in the change of the variables.

**Negative:** variables change in opposite directions.

**Python code to find covariance between two samples:**

# generate related variables

from numpy import cov

from numpy.random import randn

from numpy.random import seed

# prepare data

data1 = 20 \* randn(1000) + 100

data2 = data1 + (10 \* randn(1000) + 50)

covariance = cov(data1, data2)

print(covariance)

One problem with covariance as a statistical tool alone is that it is challenging to interpret. This leads us to the Pearson’s correlation coefficient next.

**Correlation Coefﬁcient:** The correlation coefﬁcient between two variables answers the question: “Are the two variables related? That is, if one variable changes, does the other also change?”

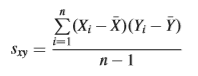
* In terms of the strength of relationship, the value of the correlation coefficient varies between +1 and -1.
* A value of ± 1 indicates a perfect degree of association between the two variables.
* As the correlation coefficient value goes towards 0, the relationship between the two variables will be weaker.
* The direction of the relationship is indicated by the sign of the coefficient; a + sign indicates a positive relationship and a – sign indicates a negative relationship.

**Pearson’s correlation coefficient ( product moment correlation coefficient” (PMCC)) :**

Also called as population correlation coefficient and can be used to summarize the strength of the linear relationship between two data samples. If the two variables are normally distributed, the standard measure of determining the correlation coefﬁcient, often measured by Pearson. The Pearson’s correlation coefficient is calculated as the covariance of the two variables divided by the product of the standard deviation of each data sample.

Pearson's correlation coefficient = covariance(X, Y) / (stdv(X) \* stdv(Y))

With the sample covariance sxy deﬁned as:



and sx, sy the sample standard deviations of the x and y values, respectively can also be written as:



Pearson’s correlation coefﬁcient, can take any value from -1 to +1.

**Python code:**

# calculate the Pearson's correlation between two variables

from numpy.random import randn

from numpy.random import seed

from scipy.stats import pearsonr

# seed random number generator

seed(1)

# prepare data

data1 = 20 \* randn(1000) + 100

data2 = data1 + (10 \* randn(1000) + 50)

# calculate Pearson's correlation

corr, \_ = pearsonr(data1, data2)

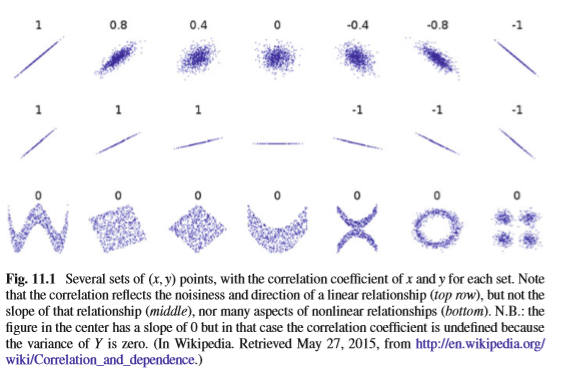
print('Pearsons correlation: %.3f' % corr)

The coefficient returns a value between -1 and 1 that represents the limits of correlation from a full negative correlation to a full positive correlation.

A value of 0 means no correlation.

A value below -0.5 or above 0.5 indicates a notable correlation,

Values below those values suggests a less notable correlation.

****

**Rank based correlation:**

If the data distribution is not normal, a different approach is necessary. In that case one can rank the set of data for each variable and compare the orderings.

There are two commonly used methods of calculating the rank correlation.

* Spearman’s
* Kendall’s

Two variables may be related by a nonlinear relationship, such that the relationship is stronger or weaker across the distribution of the variables.

1. **Spearman’s Correlation (**ρ)**:** If you are unsure of the distribution and possible relationships between two variables, Spearman correlation coefficient is a good tool to use. These statistics are calculated from the relative rank of values on each sample. This is a common approach used in non-parametric statistics, e.g. statistical methods where we do not assume a distribution of the data such as Gaussian.

Spearman's correlation coefficient = covariance(rank(X), rank(Y)) / (stdv(rank(X)) \* stdv(rank(Y)))

The spearmanr() SciPy function can be used to calculate the Spearman’s correlation coefficient between two data samples with the same length.

**Python code:**

# calculate the spearmans's correlation between two variables

from numpy.random import randn

from numpy.random import seed

from scipy.stats import spearmanr

# seed random number generator

seed(1)

# prepare data

data1 = 20 \* randn(1000) + 100

data2 = data1 + (10 \* randn(1000) + 50)

# calculate spearman's correlation

corr, \_ = spearmanr(data1, data2)

print('Spearmans correlation: %.3f' % corr)

1. **Kendall’s (τ):** is also a rank correlation coefﬁcient, measuring the association between two measured quantities. It is harder to calculate than Spearman’s, but it has been argued that conﬁdence intervals for Spearman’s are less reliable and less interpretable than conﬁdence intervals for Kendall’s parameters.

**Python code:**

from numpy.random import randn

from numpy.random import seed

from scipy.stats import kendalltau

# seed random number generator

seed(1)

# prepare data

data1 = 20 \* randn(1000) + 100

data2 = data1 + (10 \* randn(1000) + 50)

# calculate kendal's correlation

corr, p = kendalltau(data1, data2)

print('Kendals correlation: %.3f' % corr)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Regression**

**Regression analysis** is used to find mathematical relationship between one variable (Dependent variable) and a set of independent variables.

**Ex:** relationship between people’s weights and heights, or study time and test scores, or two animal populations

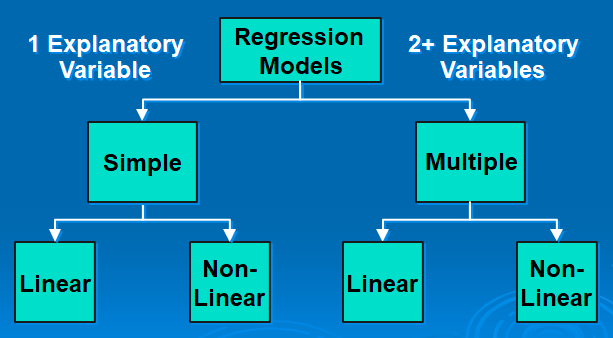
**Types of regression:**

**Simple regression:** single independent variable

**Muliple regression:** with two or more independent variables

**Linear regression:** The relationship between independent and dependent variables is linear

**Non-Linear regression:** The relationship between independent and dependent variables is non-linear

****

**Linear regression assumetions:**

1. The relationship between X and Y is linear

2. Y is distributed normally at each value of X

3. The variance of Y at every value of X is the same (homogeneity of variances)

4. The observations are independent

**General Linear Regression Model**

We can use the method of Linear Regression when we want to predict the value of one variable (dependent variable) from the value(s) of one or more other variables (independent variables).

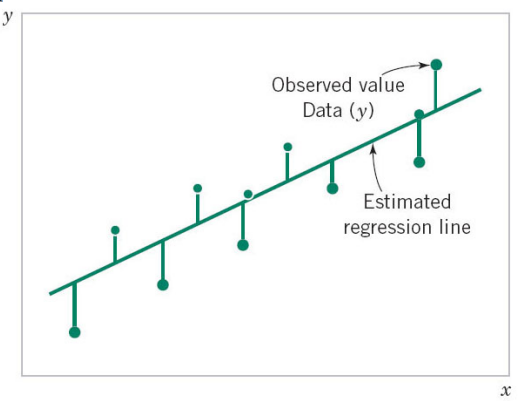
**Simple linear regression model:** We search for the best-ﬁt line to a given dataset {xi; yi} we are looking for the parameters (β0; β1) which minimize the sum of the squared residuals ϵi.

yi = β0 + β1 \* xi + ϵi

where β0 is the slope or inclination of the line, and β1 the intercept. The residuals are the differences between observed values and predicted values.

We use the sample data to estimate the population parameters: the slope β1 and the intercept β0. One such method to estimate parameters is the Least squares estimation.

**Least squares method:** To choose the ‘best fitting line’ using least squares estimation, we minimize the sum of the squared vertical distances of each point to the fitted line.



We let ‘hats’ denote predicted values or estimates of parameters, so we have:

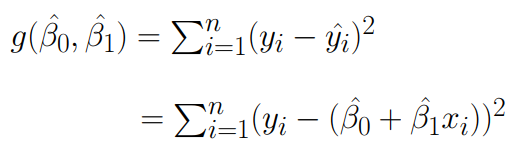


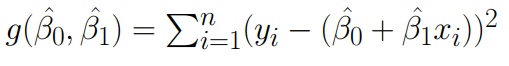
– This vertical distance of a point from the fitted line is called a residual. The residual for observation i is denoted ei and 

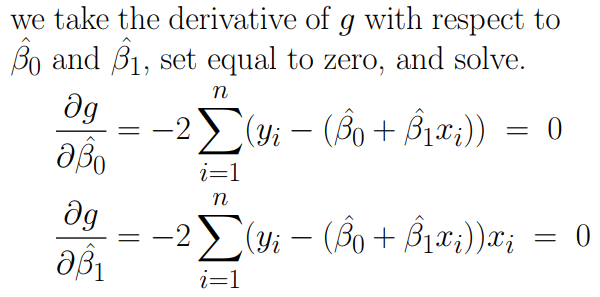
We wish to choose βˆ 0 and βˆ 1 such that we minimize the sum of the squared vertical distances of each point to the fitted line, So, in least squares estimation, we wish to minimize the sum of the squared residuals (or error sum of squares SSE).

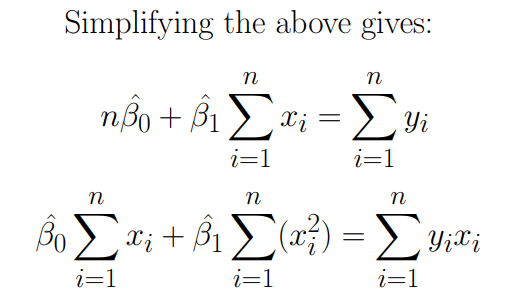
i.e. minimize 

Minimize:

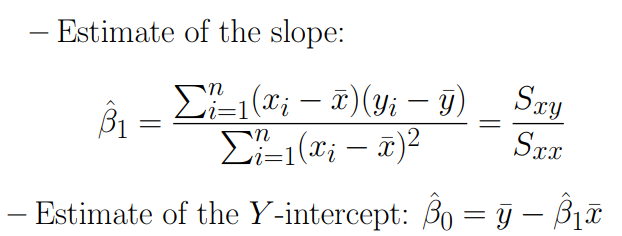


To minimize 





And these two equations are known as the least squares normal equations



Python code:

import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

from numpy.random import randn

from sklearn.model\_selection import train\_test\_split

# Load the diabetes dataset

diabetes = datasets.load\_diabetes()

# Use only one feature

diabetes\_X = diabetes.data[:, np.newaxis, 2]

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(diabetes\_X, diabetes.target, test\_size = 0.2)

# Create linear regression object

regr = linear\_model.LinearRegression()

# Train the model using the training sets

regr.fit(X\_train, y\_train)

y\_pred = regr.predict(X\_test)

# The coefficients

print('Coefficients: \n', regr.coef\_)

# The mean squared error

print("Mean squared error: %.2f"

% mean\_squared\_error(y\_test, y\_pred))

# Explained variance score: 1 is perfect prediction

print('Variance score: %.2f' % r2\_score(y\_test, y\_pred))

Since the linear regression equation is solved to minimize the square sum of the residuals, linear regression is sometimes also called Ordinary Least-Squares (OLS) Regression.

**Python code:**

import statsmodels.api as sm

# Note the difference in argument order

model = sm.OLS(y\_train,X\_train).fit()

predictions = model.predict(X\_test) # make the predictions by the model

# Print out the statistics

model.summary()

**Fitting different models:**

import numpy as np

import matplotlib.pyplot as plt

np.set\_printoptions(precision=3)

#generate the data

x = np.arange(100)

y = 150 + 3\*x + 0.03\*x\*\*2 + 5\*np.random.randn(len(x))

#Design the design matrices

**Linefit**

M1 = np.vstack( (np.ones\_like(x), x) ).T

#print(M1)

# Solve the equations

p1 = np.linalg.lstsq(M1, y)

#estimated parameters

print('The coefficients from the linear fit: {0}'.format(p1[0]))

#line fit

y\_pred1= p1[0][1] \* x + p1[0][0]

**quadratic fit**

#Design the design matrices

M2 = np.vstack( (np.ones\_like(x), x, x\*\*2) ).T

#print(M1)

# Solve the equations

p2 = np.linalg.lstsq(M2, y)

#estimated parameters

print('The coefficients from the quadratic fit: {0}'.format(p2[0]))

#quadratic fit

y\_pred2= p2[0][0] + p2[0][1] \* x + p2[0][2] \* x \*x #quadratic fit

**cubic fit**

#Design the design matrices

M3 = np.vstack( (np.ones\_like(x), x, x\*\*2, x\*\*3) ).T

# Solve the equations

p3 = np.linalg.lstsq(M3, y)

#estimated parameters

print('The coefficients from the quadratic fit: {0}'.format(p3[0]))

#cubic fit

y\_pred3= p3[0][0] + p3[0][1] \* x + p3[0][2] \* x \*x + p3[0][3] \* x \* x \* x #cubic fit

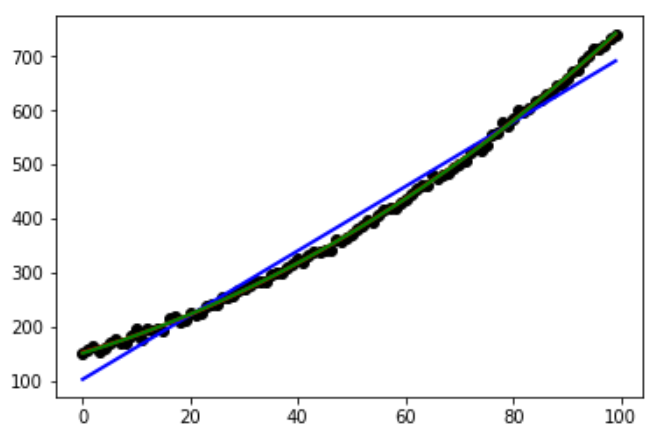
**plot the data**

plt.scatter(x, y, color='black')

plt.plot(x, y\_pred1, color='blue', linewidth=2)

plt.plot(x, y\_pred2, color='red' , linewidth=2 )

plt.plot(x, y\_pred3, color='green' , linewidth=2 )



# Fit models USING STATS MODELS

import statsmodels.api as sm

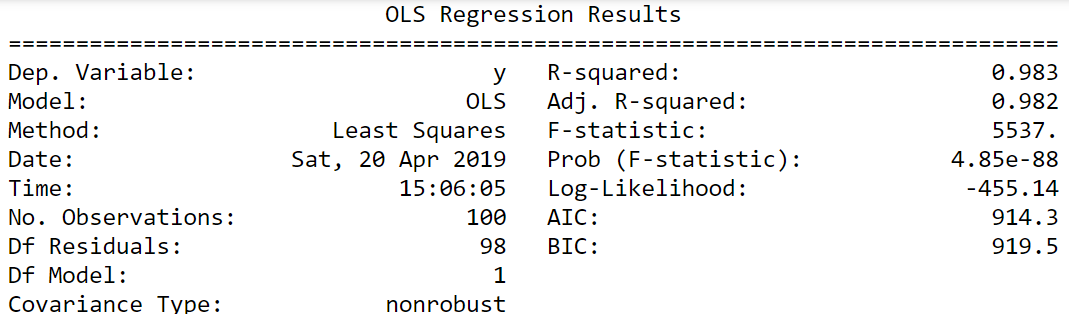
Res1 = sm.OLS(y, M1).fit()

Res2 = sm.OLS(y, M2).fit()

Res3 = sm.OLS(y, M3).fit()

print(Res1.summary())

**Summary of the first model (OLS):**



# the AIC-value,

The Akaike Information Criterion, which can be used to assess the quality of the model: the lower the AIC value, the better the model. We see that the quadratic model has the lowest AIC value and therefore is the best model: it provides the same quality of fit as the cubic model, but uses fewer parameters to achieve that quality.

# The degrees of freedom (Df )

df of the model are the number of predictors, or explanatory, variables.

The Df of the residuals is the number of observations minus the degrees of freedom of the model, minus one (for the offset).

# R-Squared

R2 value indicates the proportion of variation in the y-variable that is due to variation in the x-variables.

For simple linear regression, the R2 value is the square of the sample correlation r\_xy.

For multiple linear regression with intercept (which includes simple linear regression), the R2 value is defined as

R^2 = SS\_mod / SS\_tot

R^2 is close to 1 means the model is better