Correlation and Regression

1. Analysis of Relationship, Positive and Negative Correlation, Perfect Correlation,
2. Correlation Matrix, Scatter Plots, Simple Linear Regression,
3. R Square, Adjusted R Square, Testing of Slope, Standard Error of Estimate, Overall Model Fitness,
4. Assumptions of Linear Regression, Multiple Regression, Coefficients of Partial Determination,
5. Durbin Watson Statistics, Variance Inflation Factor
6. Statistical Inference and Hypothesis Testing Population and Sample, Null and Alternate Hypothesis,
7. Level of Significance, Type I ,and Type II Errors, One Sample t Test,
8. Confidence Intervals, One Sample Proportion Test, Paired Sample t Test, Independent Samples t Test,
9. Two Sample Proportion Tests, One Way Analysis of Variance and Chi Square Test.

R Square, Adjusted R Square, Testing of Slope, Standard Error of Estimate, Overall Model Fitness,

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/weight-height.csv")
```

```
df
```

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801
...	...	...	...
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

```
df=df[df.Gender=='Female']
x=df[['Weight']]
y=df.Height
```

1. Step Calculation of SSxx

1. step calculating SSxx

$$SSxx = \sum(\bar{x} - x)^2$$

```
xmean=df.Weight.mean()
```

```
df['diffx']=xmean-x
```

```
df['diffx^2']=df.diffx**2
```

```
SSxx= df['diffx^2'].sum()
```

```
SSxx
```

```
1808909.5527405904
```

2. step calculating SSxy

$$SSxy = \sum(\bar{x} - x) * (\bar{y} - y)$$

Double-click (or enter) to edit

```
ymean=y.mean()
```

```
df['diffy']=ymean-y
```

```
SSxy =(df.diffx*df.diffy).sum()
```

```
SSxy
```

    217838.44441885184

Slope

$$m = \frac{SSxy}{SSxx}$$

```
m=SSxy/SSxx
```

```
m
```

    0.12042528278366127

Intercept

$$b = \bar{y} - m * \bar{x}$$

```
b=ymean-m*xmean
```

```
b
```

    47.34778348398618

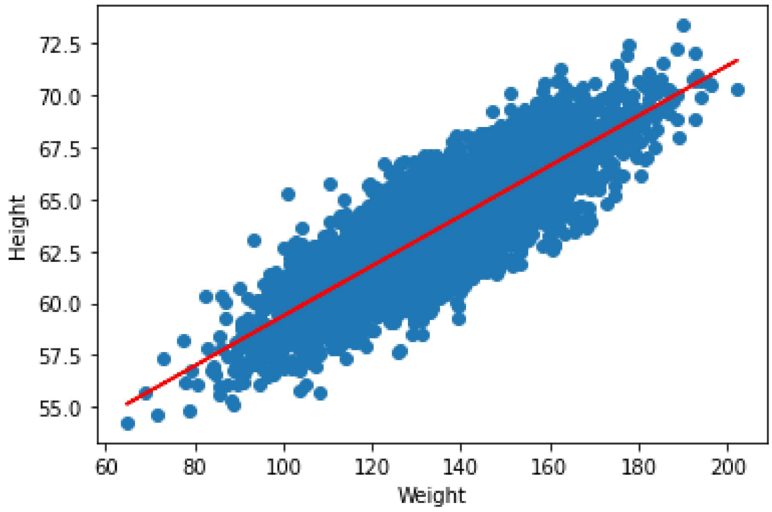
Visualisation

```
import matplotlib.pyplot as plt
```

```
plt.scatter(x,y)
```

```
plt.xlabel('Weight')
plt.ylabel('Height')
plt.plot(x,m*x+b,'r')
```

```
[<matplotlib.lines.Line2D at 0x7f3341bdb350>]
```



Predict values with the model

```
def predicted_y(value):
  predict =m*value+b
  return predict
```

R-squared

$$R^2 = 1 - \frac{SSE}{SST}$$

Total sum of squares

$$SST = \sum_i (y_i - \bar{y})^2$$

```
SST=((y-ymean)**2).sum()
```

```
SST
```

```
36342.46752085695
```


```
y_hat= predicted_y(x)
```

```
y_hat
```

	Weight
5000	59.641799
5001	64.364577
5002	63.128481
5003	62.782874
5004	62.976746
...	...
9995	63.819247
9996	67.924599
9997	62.819460
9998	67.079762
9999	61.034009

5000 rows × 1 columns

```
y_hat=y_hat.rename(columns={'Weight':'Height'})
```

```
y_hat
```

	Height
5000	59.641799
5001	64.364577
5002	63.128481
5003	62.782874

```
SSE=((y-y_hat.Height)**2).sum()
```

```
SSE
```

    10109.21125056385

9997	62.819460

```
R_Square=(SST-SSE)/SST
```

```
R_Square
```

    0.7218347586122992

```
from sklearn import linear_model
model =linear_model.LinearRegression()
```

```
model.fit(x,y)
```

    LinearRegression()

```
model.score(x,y)
```

    0.7218347586122992

Double-click (or enter) to edit

```
from google.colab import drive
drive.mount('/content/drive')
```

    Mounted at /content/drive

```
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/weight-height.csv")
```

```
df
```

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801
...	...	...	...
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461

Adusted- R Square

```
from sklearn.linear_model import LinearRegression
import pandas as pd

#define URL where dataset is located
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/mtcars.csv"

#read in data
data = pd.read_csv(url)

#fit regression model
model = LinearRegression()
X, y = data[["mpg", "wt", "drat", "qsec"]], data.hp
model.fit(X, y)

#display adjusted R-squared
1 - (1-model.score(X, y))*(len(y)-1)/(len(y)-X.shape[1]-1)
```

```
0.7787005290062521
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from sklearn.linear_model import LinearRegression
import pandas as pd
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/student_scores.csv")
```

```
df
```

	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30
5	1.5	20
6	9.2	88
7	5.5	60
8	8.3	81
9	2.7	25
10	7.7	85
11	5.9	62
12	4.5	41
13	3.3	42
14	1.1	17
15	8.9	95
16	2.5	30
17	1.9	24
18	6.1	67
19	7.4	69
20	2.7	30
21	4.8	54
22	3.8	35
23	6.9	76
24	7.8	86

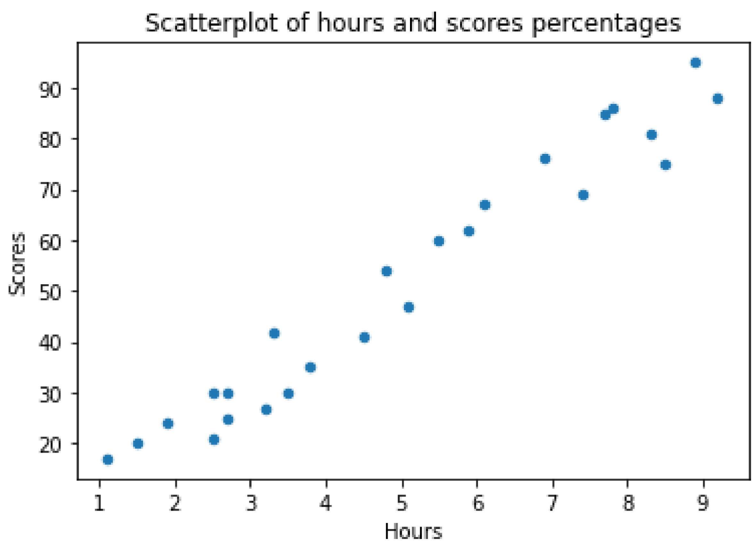
```
df.head()
```

	Hours	Scores
0	2.5	21

```
df.shape
```

```
(25, 2)
```

```
df.plot.scatter(x='Hours', y='Scores', title='Scatterplot of hours and scores percentages'
```



As the hours increase, so do the scores. There's a fairly high positive correlation here! Since the shape of the line the points are making appears to be straight -

we say that there's a positive linear correlation between the Hours and Scores variables. How correlated are they? The corr() method calculates and displays the correlations between numerical variables in a DataFrame:

```
print(df.corr())
```

```
           Hours    Scores
Hours   1.000000  0.976191
Scores  0.976191  1.000000
```

In this table, Hours and Hours have a 1.0 (100%) correlation, just as Scores have a 100% correlation to Scores, naturally.

Any variable will have a 1:1 mapping with itself! However, the correlation between Scores and Hours is 0.97. Anything above 0.8 is considered to be a strong positive correlation.

What Is Standard Deviation

Standard deviation is a measure of how far numbers lie from the average.

For example, if we look at a group of men we find that most of them are between 5’8” and 6’2” tall. Those who lie outside this range make up only a small percentage of the group. The standard deviation identifies the percentage by which the numbers tend to vary from the average.

The standard deviation follows the formula:

$$\sigma = \sqrt{\frac{\Sigma\left(x_i - \mu\right)^2}{N - 1}}$$

Where:

σ = sample standard deviation

N = the size of the population

x_i = each value from the population

μ = the sample mean (average)

How to Calculate Standard Deviation in Python

Assuming we do not use a built-in standard deviation function, we need to implement the above formula as a Python function to calculate the standard deviation.

Here is the implementation of standard deviation in Python:

```
import statistics

data = [7,5,4,9,12,45]

print("Standard Deviation of the sample is % s "% (statistics.stdev(data)))
print("Mean of the sample is % s " % (statistics.mean(data)))
```

```
Standard Deviation of the sample is 15.61623087261029
```

```
Mean of the sample is 13.666666666666666
```

Write Custom Function to Calculate Standard Deviation

```
import numpy as np #for declaring an array or simply use list

def mean(data):
  n = len(data)
  mean = sum(data) / n
  return mean

def variance(data):
  n = len(data)
  mean = sum(data) / n
  deviations = [(x - mean) ** 2 for x in data]
  variance = sum(deviations) / n
  return variance

def stdev(data):
  import math
  var = variance(data)
  std_dev = math.sqrt(var)
  return std_dev

data = np.array([7,5,4,9,12,45])

print("Standard Deviation of the sample is % s "% (stdev(data)))
print("Mean of the sample is % s " % (mean(data)))
```

```
Standard Deviation of the sample is 14.2556031868954
Mean of the sample is 13.666666666666666
```

Standard Error of Mean (SEM)

The SEM is used to measure how close sample means are likely to be to the true population mean. This gives a good indication as to where a given sample actually lies in relation to its corresponding population.

The standard error of the mean follows the following formula:

The standard error of the mean follows the following formula:

$$SE = \frac{\sigma}{\sqrt{n}}$$

Where σ is the standard deviation and n is the number of samples.

How to Implement Standard Error of Mean Function in Python

```
from math import sqrt
def stddev(data):
    N = len(data)
    mu = float(sum(data) / len(data))
    s = [(x_i - mu) ** 2 for x_i in data]
    return sqrt(float(sum(s) / (N - 1)))
def sem(data):
    return stddev(data) / sqrt(len(data))
data = [19, 2, 12, 3, 100, 2, 3, 2, 111, 82, 4]
sem_data = sem(data)
print(sem_data)
```

```
13.172598656753378
```

What Is Autocorrelation?

Autocorrelation is a mathematical representation of the degree of similarity between a given time series and a lagged version of itself over successive time intervals. It's conceptually similar to the correlation between two different time series, but autocorrelation uses the same time series twice: once in its original form and once lagged one or more time periods.

For example, if it's rainy today, the data suggests that it's more likely to rain tomorrow than if it's clear today. When it comes to investing, a stock might have a strong positive autocorrelation of returns, suggesting that if it's "up" today, it's more likely to be up tomorrow, too.

Naturally, autocorrelation can be a useful tool for traders to utilize; particularly for technical analysts.

KEY TAKEAWAYS

Autocorrelation represents the degree of similarity between a given time series and a lagged version of itself over successive time intervals.

Autocorrelation measures the relationship between a variable's current value and its past values.

An autocorrelation of +1 represents a perfect positive correlation, while an autocorrelation of negative 1 represents a perfect negative correlation. Technical analysts can use autocorrelation to measure how much influence past prices for a security have on its future price.

Durbin Watson Statistics

Autocorrelation, also known as serial correlation, can be a significant problem in analyzing historical data if one does not know to look out for it. For instance, since stock prices tend not to change too radically from one day to another, the prices from one day to the next could potentially be highly correlated, even though there is little useful information in this observation. In order to avoid autocorrelation issues, the easiest solution in finance is to simply convert a series of historical prices into a series of percentage-price changes from day to day.

Autocorrelation can be useful for technical analysis, which is most concerned with the trends of, and relationships between, security prices using charting techniques in lieu of a company's financial health or management. Technical analysts can use autocorrelation to see how much of an impact past prices for a security have on its future price.

Autocorrelation can show if there is a momentum factor associated with a stock. For example, if you know that a stock historically has a high positive autocorrelation value and you witnessed the stock making solid gains over the past several days, then you might reasonably expect the movements over the upcoming several days (the leading time series) to match those of the lagging time series and to move upward.

Example: Durbin-Watson Test in Python

Suppose we have the following dataset that describes the attributes of 10 basketball players: **bold text**

```
import numpy as np
import pandas as pd

#create dataset
df = pd.DataFrame({'rating': [90, 85, 82, 88, 94, 90, 76, 75, 87, 86],
                   'points': [25, 20, 14, 16, 27, 20, 12, 15, 14, 19],
                   'assists': [5, 7, 7, 8, 5, 7, 6, 9, 9, 5],
                   'rebounds': [11, 8, 10, 6, 6, 9, 6, 10, 10, 7]})
```

```
                    'rebounds': [11, 8, 10, 6, 6, 9, 6, 10, 10, 7]})

#view dataset
df
```

	rating	points	assists	rebounds
0	90	25	5	11
1	85	20	7	8
2	82	14	7	10
3	88	16	8	6
4	94	27	5	6
5	90	20	7	9
6	76	12	6	6
7	75	15	9	10
8	87	14	9	10
9	86	19	5	7

Suppose we fit a multiple linear regression model using rating as the response variable and the other three variables as the predictor variables:

```
from statsmodels.formula.api import ols

#fit multiple linear regression model
model = ols('rating ~ points + assists + rebounds', data=df).fit()

#view model summary
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 rating   R-squared:                       0.623
Model:                            OLS   Adj. R-squared:                  0.434
Method:                 Least Squares   F-statistic:                     3.299
Date:                Thu, 27 Oct 2022   Prob (F-statistic):             0.0995
Time:                        15:51:03   Log-Likelihood:                -26.862
No. Observations:                  10   AIC:                             61.72
Df Residuals:                       6   BIC:                             62.93
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
Intercept     62.4716     14.588      4.282      0.005      26.776      98.168
points         1.1193      0.411      2.724      0.034       0.114       2.125
assists        0.8834      1.381      0.640      0.546      -2.495       4.262
rebounds      -0.4278      0.851     -0.503      0.633      -2.510       1.655
==============================================================================
Omnibus:                        2.711   Durbin-Watson:                   2.392
Prob(Omnibus):                  0.258   Jarque-Bera (JB):                0.945
Skew:                          -0.751   Prob(JB):                        0.624
Kurtosis:                       3.115   Cond. No.                         217.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
/usr/local/lib/python3.7/dist-packages/scipy/stats/stats.py:1542: UserWarning: kurtos
  "anyway, n=%i" % int(n))
```

We can perform a Durbin Watson using the durbin_watson() function from the statsmodels library to determine if the residuals of the regression model are autocorrelated:

```
from statsmodels.stats.stattools import durbin_watson

#perform Durbin-Watson test
durbin_watson(model.resid)
```

```
2.3920546872335353
```

The test statistic is 2.392. Since this is within the range of 1.5 and 2.5, we would consider autocorrelation not to be problematic in this regression model.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy as sp
import seaborn as sns
import statsmodels.api as sm
import statsmodels.tsa.api as smt
import warnings
from google.colab import drive
from sklearn.model_selection import train_test_split
```

```
warnings.filterwarnings("ignore")
%matplotlib inline
```

```
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
path = "/content/drive/My Drive/Colab Notebooks/durbin_data.csv"
```

```
df = pd.read_csv(path)
```

```
df
```

	X1	X2	X3	Y
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

200 rows × 4 columns

```
print ("Total number of rows in dataset = {}".format(df.shape[0]))
print ("Total number of columns in dataset = {}".format(df.shape[1]))
```

```
Total number of rows in dataset = 200
Total number of columns in dataset = 4
```

```
df.head()
```

	X1	X2	X3	Y
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4

```
target_col = "Y"


X = df.loc[:, df.columns != target_col]
y = df.loc[:, target_col]


# Split the data into train and test with 70% data being used for training
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.30,
                                                    random_state=42)
```

Linear Regression using statsmodels

```
X_with_constant = sm.add_constant(X_train)
model = sm.OLS(y_train, X_with_constant)


results = model.fit()
results.params
```

```
const    2.708949
X1       0.044059
X2       0.199287
X3       0.006882
dtype: float64
```

```
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.906
Model:                            OLS   Adj. R-squared:                  0.903
Method:                 Least Squares   F-statistic:                     434.5
Date:                Sun, 30 Oct 2022   Prob (F-statistic):           1.88e-69
Time:                        15:54:50   Log-Likelihood:                -262.21
No. Observations:                 140   AIC:                             532.4
Df Residuals:                     136   BIC:                             544.2
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          2.7089      0.374      7.250      0.000       1.970       3.448
X1             0.0441      0.002     27.219      0.000       0.041       0.047
X2             0.1993      0.010     20.195      0.000       0.180       0.219
X3             0.0069      0.007      0.988      0.325      -0.007       0.021
==============================================================================
Omnibus:                       68.437   Durbin-Watson:                   2.285
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              325.342
Skew:                          -1.709   Prob(JB):                     2.25e-71
Kurtosis:                       9.640   Cond. No.                         500.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

Durbin-Watson test

Null Hypothesis $H_0 : \rho = 0$
Alternate Hypothesis $H_1 : \rho > 0$

$$dw = \frac{\sum_{i=2}^{n} (e_i - e_{i-1})^2}{\sum_{i=1}^{n} e_i^2}$$

$$e_i = y_i - \hat{y}_i$$

$\begin{array}{ll}{\text { If } dwd_{U}} & {\text { do not reject } H_{0}: \rho=0} \ {\text { If } d_{L}}

Double-click (or enter) to edit

```
y_pred = results.predict(X_with_constant)
```

```
y_pred
```

```
169    17.391498
97     15.191962
31     11.416507
12     11.206105
35     16.392562
         ...
106     6.207002
14     18.574600
92     19.382850
179    12.119172
102    17.214448
Length: 140, dtype: float64
```

```
y_pred.shape[0]
```

```
140
```

```
residual_df = pd.DataFrame(residual, columns=["ei"]).reset_index(drop=True)
```

```
residual_df.head()
```

```
residual_df.head()
```

	ei
0	-2.391498
1	0.308038
2	0.483493
3	-2.006105
4	-3.592562

```
residual_df['ei_square'] = np.square(residual_df['ei'])
```

```
sum_of_squared_residuals = residual_df.sum()["ei_square"]
sum_of_squared_residuals
```

```
347.1097250468101
```

```
residual_df.head()
```

	ei	ei_square
0	-2.391498	5.719262
1	0.308038	0.094888
2	0.483493	0.233765
3	-2.006105	4.024456
4	-3.592562	12.906499

```
residual_df['ei_minus_1'] = residual_df['ei'].shift()
```

```
residual_df.head()
```

	ei	ei_square	ei_minus_1
0	-2.391498	5.719262	NaN
1	0.308038	0.094888	-2.391498
2	0.483493	0.233765	0.308038
3	-2.006105	4.024456	0.483493
4	-3.592562	12.906499	-2.006105

```
residual_df.tail()
```

	ei	ei_square	ei_minus_1
135	0.992998	0.986044	1.785357
136	0.425400	0.180965	0.992998
137	0.017150	0.000294	0.425400
138	0.480828	0.231195	0.017150
139	-2.414448	5.829558	0.480828

```
residual_df.dropna(inplace=True)
```

```
residual_df.shape
```

```
(139, 3)
```

```
residual_df['ei_sub_ei_minus_1'] = residual_df['ei'] - residual_df['ei_minus_1']
```

```
residual_df['square_of_ei_sub_ei_minus_1'] = np.square(residual_df['ei_sub_ei_minus_1'])
```

```
residual_df.head()
```

	ei	ei_square	ei_minus_1	ei_sub_ei_minus_1	square_of_ei_sub_ei_minus_1
1	0.308038	0.094888	-2.391498	2.699536	7.287496
2	0.483493	0.233765	0.308038	0.175455	0.030784
3	-2.006105	4.024456	0.483493	-2.489598	6.198097
4	-3.592562	12.906499	-2.006105	-1.586457	2.516846
5	-0.305778	0.093500	-3.592562	3.286784	10.802948

```
sum_of_squared_of_difference_residuals = residual_df.sum()["square_of_ei_sub_ei_minus_1"]
sum_of_squared_of_difference_residuals
```

```
793.3116126261241
```

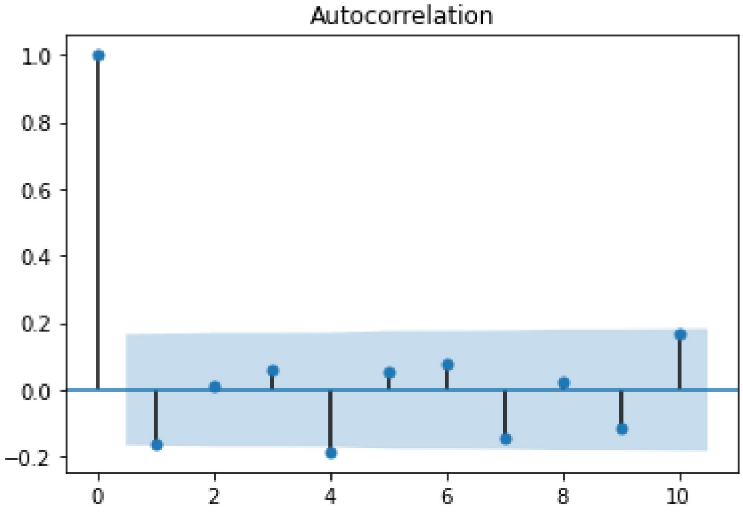
```
dw = sum_of_squared_of_difference_residuals/sum_of_squared_residuals
```

```
dw
```

```
2.2854779206175815
```

No autocorrelation of residuals

```
acf = smt.graphics.plot_acf(residual, lags=10 , alpha=0.05)
acf.show()
```



Colab paid products - Cancel contracts here

✓ 0s completed at 9:42 PM