I N D E X

Experiment : 1

Build a simple linear regression model and perform predictions on the test dataset. Consider a company's data, where there is the amount spent on different types of advertisements and its subsequent sales

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
```

```
#load the dataset
data=pd.read_csv("company.csv")
data.head()
```

	TV	Radio	Newspaper	Sales
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	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

```
X = data.drop('Sales', axis=1).values
y = data['Sales'].values
```

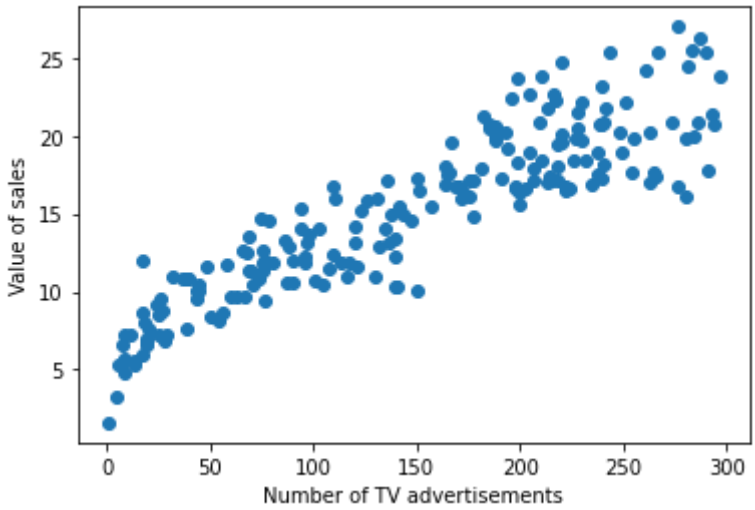
```
#correlation between features and label
data.corr()
```

	TV	Radio	Newspaper	Sales
TV	1.000000	0.054809	0.056648	0.901208
Radio	0.054809	1.000000	0.354104	0.349631
Newspaper	0.056648	0.354104	1.000000	0.157960
Sales	0.901208	0.349631	0.157960	1.000000

```
#As per the above table the relation between Tv and Sales is more
x_tv = X[:, 0]
```

```
#plotting a curve for input and label data
plt.scatter(x_tv, y)
```

```
plt.xlabel('Number of TV advertisements')
plt.ylabel('Value of sales')
plt.show()
```




```
x_tv = x_tv.reshape(-1,1)
y = y.reshape(-1, 1)
```

```
# Split dataset into training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_tv, y,test_size=0.2,random_state=42)
```

```
#size of training and testing data
X_train.shape,X_test.shape
```

```
((160, 1), (40, 1))
```

fitting the model

```
reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
```

```
LinearRegression()
```

```
from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error
y_pred1 = reg.predict(X_train)
print(f'Performance of the model on training data :\n')
print(f'MAE = {mean_absolute_error(y_train, y_pred1)}')
print(f'MSE = {mean_squared_error(y_train, y_pred1)}')
print(f'RMSE = {np.sqrt(mean_squared_error(y_train, y_pred1))}')
print(f'R_2 = {r2_score(y_train, y_pred1)}')
```

```
Performance of the model on training data :

MAE = 1.8005092256620792
MSE = 4.998442356450173
```

```
RMSE = 2.235719650683013
R_2 = 0.8134866044709264
```

```
from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error
y_pred2 = reg.predict(X_test)
print(f'Performance of the model on test data :\n')
print(f'MAE = {mean_absolute_error(y_test, y_pred2)}')
print(f'MSE = {mean_squared_error(y_test, y_pred2)}')
print(f'RMSE = {np.sqrt(mean_squared_error(y_test, y_pred2))}')
print(f'R_2 = {r2_score(y_test, y_pred2)}')
```

```
Performance of the model on test data :

MAE = 1.9502948931650088
MSE = 6.101072906773963
RMSE = 2.470035001123256
R_2 = 0.802561303423698
```

CONCLUSION : The r2 score of the training data is 0.81 and for testing data is 0.80

Experiment : 2

Build a Multiple Linear Regression model on a dataset (eg: 50_startups)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
```

```
#load the dataset
data=pd.read_csv("50_Startups.csv")
data.head()
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
x = data.drop('Profit', axis=1).values
y = data['Profit'].values
X=data.select_dtypes(include=np.number)
```

```
# Split dataset into training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#size of training data and testing data
X_train.shape, X_test.shape
```

```
((40, 4), (10, 4))
```

fitting the model

```
reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
```

```
LinearRegression()
```

```
from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error
y_pred1 = reg.predict(X_train)
```

```
print(f'Performance of the model on training data :\n')
print(f'MAE = {mean_absolute_error(y_train, y_pred1)}')
print(f'MSE = {mean_squared_error(y_train, y_pred1)}')
print(f'RMSE = {np.sqrt(mean_squared_error(y_train, y_pred1))}')
print(f'R_2 = {r2_score(y_train, y_pred1)}')
```

```
     Performance of the model on training data :

     MAE = 1.3960743672214448e-11
     MSE = 3.8985022269624195e-22
     RMSE = 1.974462515967933e-11
     R_2 = 1.0
```

```
from sklearn.metrics import mean_absolute_error,r2_score,mean_squared_error
y_pred2 = reg.predict(X_test)
print(f'Performance of the model on test data :\n')
print(f'MAE = {mean_absolute_error(y_test, y_pred2)}')
print(f'MSE = {mean_squared_error(y_test, y_pred2)}')
print(f'RMSE = {np.sqrt(mean_squared_error(y_test, y_pred2))}')
print(f'R_2 = {r2_score(y_test, y_pred2)}')
```

```
     Performance of the model on test data :

     MAE = 1.3824319466948509e-11
     MSE = 2.594038400966295e-22
     RMSE = 1.6106018753764987e-11
     R_2 = 1.0
```

CONCLUSION : The r2 score for the training data is 1.0 as well as for testing data is 1.0.

Experiment : 3

Cross validate the Simple Linear Regression and Multiple Linear Regression models

```
from numpy import mean
from numpy import absolute
from numpy import sqrt
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

a) Cross Validation on Simple Linear Regression

```
#loading the dataset
data = pd.read_csv("company.csv")
data.head()
```

	TV	Radio	Newspaper	Sales
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	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

```
X = data.drop('Sales', axis=1).values
y = data['Sales'].values
```

```
#By using the correlation factor the relation in between Sales and Tv is more
x_tv = X[:, 0]
```

```
x_tv = x_tv.reshape(-1,1)
y = y.reshape(-1, 1)
```

```
# Split dataset into training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_tv,y,test_size=0.2,random_state=42)
```

```
lm = LinearRegression()
```

KFold Cross Validation

```
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(lm, X_train, y_train, scoring='neg_mean_squared_error',cv=folds)
sqrt(mean(absolute(scores)))
```

```
2.2565669707801552
```

CONCLUSION : The root mean squared error (RMSE) was 2.256. The lower the RMSE, the more closely a model is able to predict the actual observations.

b) Cross Validation on Multiple Linear Regression

```
#loading the dataset
df = pd.read_csv("50_Startups.csv")
df.head()
```

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
x = df.drop('Profit', axis=1).values
y1 = df['Profit'].values
X=df.select_dtypes(include=np.number)
```

```
# Split dataset into training set and test set
from sklearn.model_selection import train_test_split
X_train1,X_test1,y_train1,y_test1=train_test_split(X,y1,test_size=0.2,random_state=42)
```

```
model = LinearRegression()
```

KFold Cross Validation

```
fold = KFold(n_splits = 5, shuffle = True, random_state = 1)
score = cross_val_score(model,X_train1,y_train1,scoring='neg_mean_squared_error',cv=folds)
sqrt(mean(absolute(score)))
```

```
2.5595449696029915e-11
```

CONCLUSION : The root mean squared error (RMSE) was 2.55e^-11. The lower the RMSE, the more closely a model is able to predict the actual observations.

Experiment : 4

Design a Naive Bayes classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

```
import pandas as pd
import numpy as np
from sklearn import datasets
wine = datasets.load_wine()
```

```
print("Features: ", wine.feature_names)
print("Labels: ", wine.target_names)
```

```
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_
Labels:  ['class_0' 'class_1' 'class_2']
```

```
X=wine.data
y=wine.target
# Split dataset into training set and test set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=109)
```

fitting the model

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
GaussianNB()
```

predicting the model

```
y_pred = gnb.predict(X_test)
```

Accuracy of the model is :

```
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9444444444444444
```

CONCLUSION : The accuracy of classifier is 0.94

Experiment : 5

Build a Logistic Regression classifier by considering a suitable dataset.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

```
#loading the dataset
data=pd.read_csv("https://assets.datacamp.com/production/repositories/628/datasets/444cdbf
data.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
X = data.drop('diabetes', axis=1).values
y = data['diabetes'].values
```

```
#split the dataset into training and testing data
X_train, X_test, y_train, y_test =train_test_split(X, y, test_size=0.1, random_state=42)
```

```
#normalization of data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

fitting the model

```
model = LogisticRegression(C=0.01,random_state=42)
model.fit(X_train, y_train)
```

```
LogisticRegression(C=0.01, random_state=42)
```

```
#normalization of testing data
X_test = scaler.transform(X_test)
```

```
y_pred = model.predict(X_test)
```

```
model.score(X_train, y_train)
```

```
0.7713458755426917
```

```
model.score(X_test, y_test)
```

```
0.7662337662337663
```

confusion matrix for evaluation of model

```
confusion_matrix(y_test, y_pred)
```

```
array([[44,  6],
       [12, 15]])
```

```
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.79      0.88      0.83        50
               1       0.71      0.56      0.63        27

        accuracy                           0.77        77
       macro avg       0.75      0.72      0.73        77
    weighted avg       0.76      0.77      0.76        77
```

CONCLUSION : The accuracy of the training data is 0.77 and for the testing data is 0.76.

Experiment : 6

Implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```
#load the dataset
iris=load_iris()
```

```
iris.data[:5]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

```
#labels are
print(iris.target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

```
X=iris.data
y=iris.target
#split the training and testing data
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=41)
```

fitting the model

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

predicting the label

```
y_pred = knn.predict(X_test)
```

```
knn.score(X_train, y_train), knn.score(X_test, y_test)
```

```
(0.9833333333333333, 0.9666666666666667)
```

Confusion matrix is :

```
confusion_matrix(y_test, y_pred)
```

```
array([[ 9,  0,  0],
       [ 0, 10,  1],
       [ 0,  0, 10]])
```

```
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00         9
               1       1.00      0.91      0.95        11
               2       0.91      1.00      0.95        10

        accuracy                           0.97        30
       macro avg       0.97      0.97      0.97        30
    weighted avg       0.97      0.97      0.97        30
```

CONCLUSION : The accuracy of the training data is 0.98 and for the testing data is 0.96

Experiment : 7

Design a classifier using Support vector machine on a suitable dataset

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

```
#loading the dataset
data=pd.read_csv("https://assets.datacamp.com/production/repositories/628/datasets/444cdbf
data.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
X = data.drop('diabetes', axis=1).values
y = data['diabetes'].values
```

```
#split the training and testing data
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.02,random_state=0)
```

```
#normalization of training data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

fitting the model

```
model = SVC()
model.fit(X_train, y_train)
```

```
SVC()
```

```
#scaling the testing data
X_test = scaler.transform(X_test)
```

```
y_pred = model.predict(X_test)
```

```
model.score(X_train, y_train)
```

```
0.8231382978723404
```

```
model.score(X_test, y_test)
```

```
0.9375
```

```
#confusion matrix
confusion_matrix(y_test, y_pred)
```

```
array([[9, 1],
       [0, 6]])
```

```
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       1.00      0.90      0.95        10
               1       0.86      1.00      0.92         6

        accuracy                           0.94        16
       macro avg       0.93      0.95      0.94        16
    weighted avg       0.95      0.94      0.94        16
```

CONCLUSION : The accuracy of the training data is 0.82 and for the testing dta is 0.93

Experiment : 8

Build a Decision Tree Classifier on a suitable dataset (eg. Pima Indians Diabetes dataset)

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
#load the dataset
data=pd.read_csv("https://assets.datacamp.com/production/repositories/628/datasets/444cdbf
data.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
X = data.drop('diabetes', axis=1).values
y = data['diabetes'].values
```

```
#split the data into training and testing data
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.02,random_state=0)
```

```
#scaling the training data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

fitting the model

```
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

```
#scaling the testing data
X_test = scaler.transform(X_test)
```

```
y_pred = model.predict(X_test)
```

```
model.score(X_train, y_train)
```

```
1.0
```

```
model.score(X_test, y_test)
```

```
0.8125
```

```
#confusion matrix
confusion_matrix(y_test, y_pred)
```

```
array([[9, 1],
       [2, 4]])
```

```
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.82      0.90      0.86        10
               1       0.80      0.67      0.73         6

        accuracy                           0.81        16
       macro avg       0.81      0.78      0.79        16
    weighted avg       0.81      0.81      0.81        16
```

CONCLUSION : The accuracy of training data is 1.0 and for the testing data is 0.81

Experiment - 9

K-Means Clustering

use the Mall Customer dataset to segment the customers in clusters based on their Age, Annual Income, Spending Score, etc.

```
from sklearn.cluster import KMeans
from sklearn import preprocessing
import sklearn.cluster as cluster
import sklearn.metrics as metrics
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
```

```
#Load the dataset
df = pd.read_csv("Mall_Customers.csv")
df.head()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
df.shape
```

```
(200, 5)
```

Objective

Customer segmentation deals with grouping clusters together based on some common patterns within their attributes. To keep the example simple and to visualize the clustering on a 2-D graph we will use only two attributes Annual Income and Spending Score.

Apply Feature Scaling

Clustering algorithms like K-means require feature scaling of the data as part of data preprocessing to produce good results. This is because clustering techniques use distance calculation between the data points. Hence it is proper to bring data of different units under a common scale.

```
scaler = MinMaxScaler()
scale = scaler.fit_transform(df[['Annual Income (k$)','Spending Score (1-100)']])
df_scale = pd.DataFrame(scale, columns = ['Annual Income (k$)','Spending Score (1-100)']);
df_scale.head(5)
```

	Annual Income (k$)	Spending Score (1-100)
0	0.000000	0.387755
1	0.000000	0.816327
2	0.008197	0.051020
3	0.008197	0.775510
4	0.016393	0.397959

Applying Kmeans with 2 Clusters (K=2)

Let us see how to apply K-Means in Sklearn to group the dataset into 2 clusters (0 and 1). The output shows the cluster (0th or 1st) corresponding to the data points in the dataset.

```
km=KMeans(n_clusters=2)
y_predicted = km.fit_predict(df_scale[['Annual Income (k$)','Spending Score (1-100)']])
y_predicted
```

```
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0], dtype=int32)
```

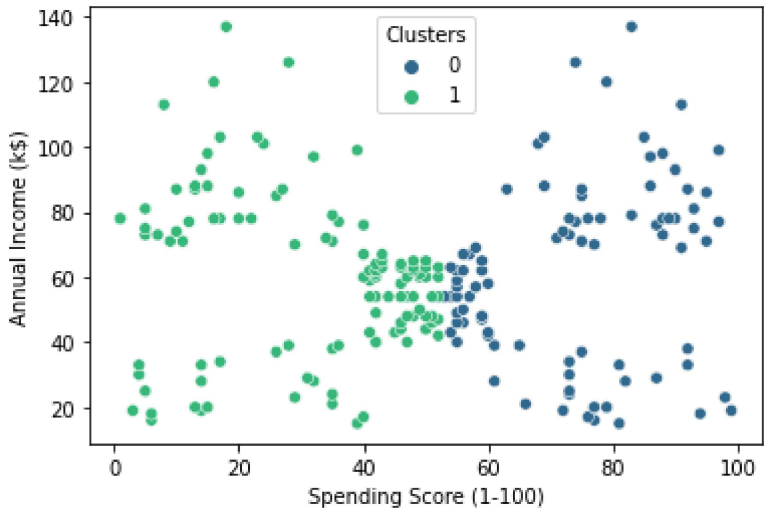
```
km.cluster_centers_
```

```
array([[0.37861485, 0.73950929],
       [0.36929553, 0.31163817]])
```

Finally, let us visualize the results. From the graph, it is evident that there is a scope for data to be grouped into more clusters than only 2. But how to know how many clusters? Let us understand this in the next section.

```
df['Clusters'] = km.labels_
sns.scatterplot(x="Spending Score (1-100)", y="Annual Income (k$)",hue = 'Clusters',  data=df
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5b019223d0>
```



Finding Optimum number of Clusters in K Means

The tricky part with K-Means clustering is you do not know in advance that in how many clusters the given data can be divided (hence it is an unsupervised learning algorithm). It can be done with the trial and error method but let us see a more proper technique for this.

i) Elbow Method with Within-Cluster-Sum of Squared Error (WCSS)

The Elbow Method is a popular technique for determining the optimal number of clusters. Here, we calculate the Within-Cluster-Sum of Squared Errors (WCSS) for various values of k and choose the k for which WSS first starts to diminish. In the plot of WSS-versus-k, this can be observed as an elbow.

- The Squared Error for a data point is the square of the distance of a point from its cluster center.
- The WSS score is the summation of Squared Errors for all given data points.
- Distance metrics like Euclidean Distance or the Manhattan Distance can be used.

Continuing with our example, we calculate the WCSS for K=2 to k=12 and calculate the WCSS in each iteration.

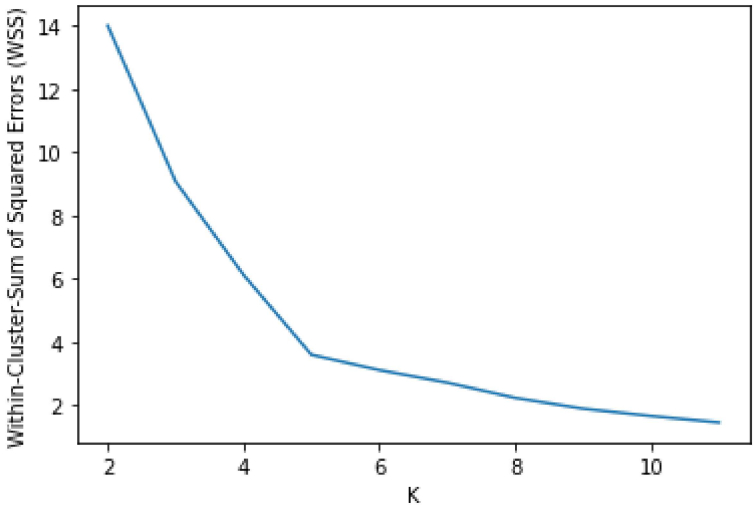
```
K=range(2,12)
```

```
wss = []

for k in K:
    kmeans=cluster.KMeans(n_clusters=k)
    kmeans=kmeans.fit(df_scale)
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)
```

Let us now plot the WCSS vs K cluster graph. It can be seen below that there is an elbow bend at K=5 i.e. it is the point after which WCSS does not diminish much with the increase in value of K.

```
plt.xlabel('K')
plt.ylabel('Within-Cluster-Sum of Squared Errors (WSS)')
plt.plot(K,wss)
```

```
[<matplotlib.lines.Line2D at 0x7f5afeb88450>]
```



Applying Kmeans with 5 Clusters (K=5)

Now that we have identified that the optimum value of K is 5

```
km=KMeans(n_clusters=5)
y_predicted = km.fit_predict(df_scale[['Annual Income (k$)','Spending Score (1-100)']])
y_predicted
```

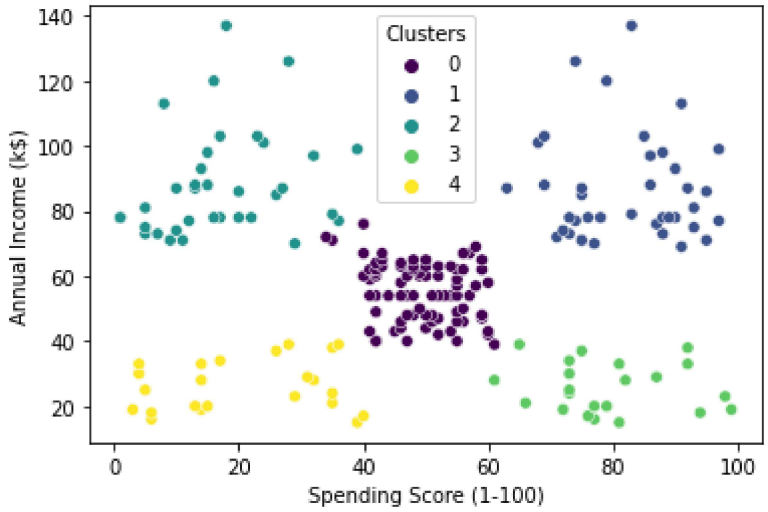
```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 0,
       4, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 1, 0, 1, 2, 1, 2, 1,
       0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 0, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
```

```
       2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,
       2, 1], dtype=int32)
```

```
df['Clusters'] = km.labels_
```

```
sns.scatterplot(x="Spending Score (1-100)", y="Annual Income (k$)",hue = 'Clusters',  data=df
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5afe967650>
```



Experiment - 10

Principal Component Analysis

Perform Dimensionality reduction using PCA on Breast Cancer Dataset to reduce the number of features to 2

```
#import libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
```

The Data

Let's work with the cancer data set since it had so many features.

```
from sklearn.datasets import load_breast_cancer
```

```
cancer = load_breast_cancer()
```

```
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filenam
```

```
print(cancer['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
```

```
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        worst/largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 0 is Mean Radius, field
        10 is Radius SE, field 20 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
```

```
df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])
#(['DESCR', 'data', 'feature_names', 'target_names', 'target'])
```

```
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	symm
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0

PCA Visualization

As we've noticed before it is difficult to visualize high dimensional data, we can use PCA to find the first two principal components, and visualize the data in this new, two-dimensional space, with a single scatter-plot. Before we do this though, we'll need to scale our data so that each feature has a single unit variance.

[ ] ↳ *12 cells hidden*

Interpreting the components

Unfortunately, with this great power of dimensionality reduction, comes the cost of being able to easily understand what these components represent.

The components correspond to combinations of the original features, the components themselves are stored as an attribute of the fitted PCA object:

```
pca.components_
```

```
array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
         0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
         0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
         0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
         0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
         0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
       [-0.23385713, -0.05970609, -0.21518136, -0.23107671,  0.18611302,
         0.15189161,  0.06016536, -0.0347675 ,  0.19034877,  0.36657547,
        -0.10555215,  0.08997968, -0.08945723, -0.15229263,  0.20443045,
         0.2327159 ,  0.19720728,  0.13032156,  0.183848  ,  0.28009203,
        -0.21986638, -0.0454673 , -0.19987843, -0.21935186,  0.17230435,
         0.14359317,  0.09796411, -0.00825724,  0.14188335,  0.27533947]])
```

In this numpy matrix array, each row represents a principal component, and each column relates back to the original features. we can visualize this relationship with a heatmap:

Experiment *11*

Time series modeling - Predict number of air passengers per month. (Dataset : AirPassengers)

Build a model to forecast the demand (passenger traffic) in Airplanes. The data is classified in date/time and thepassengers travelling per month

```
import numpy as np
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 10,6
```

```
data = pd.read_csv("AirPassengers.csv")
```

```
data.head(3)
```

	Month	#Passengers
0	1949-01	112
1	1949-02	118
2	1949-03	132

```
data.shape
```

```
(144, 2)
```

```
data.tail(3)
```

	Month	#Passengers
141	1960-10	461
142	1960-11	390
143	1960-12	432

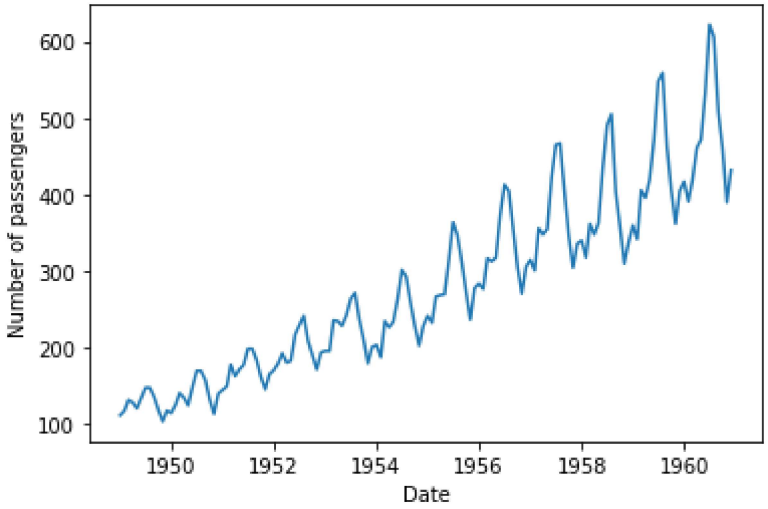
```
#parse strings to datetime type
data['Month'] = pd.to_datetime(data['Month'], infer_datetime_format=True)
indexedData = data.set_index(['Month'])
```

```
indexedData.head(2)
```

Month	#Passengers
1949-01-01	112
1949-02-01	118

```
#plot the graph
plt.xlabel("Date")
plt.ylabel("Number of passengers")
plt.plot(indexedData)
```

```
[<matplotlib.lines.Line2D at 0x7f305d296fd0>]
```

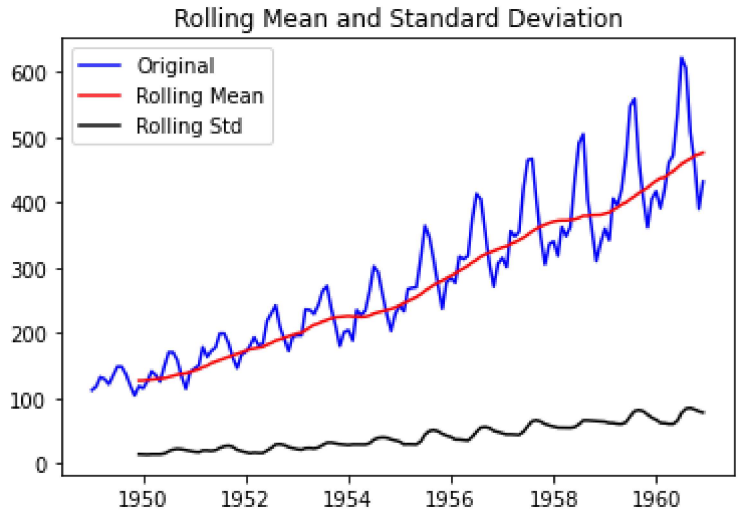



```
#Determining Rolling statistics
rolmean = indexedData.rolling(window=12).mean()
rolstd = indexedData.rolling(window=12).std()
print(rolmean, rolstd)
```

```
             #Passengers
Month
1949-01-01           NaN
1949-02-01           NaN
1949-03-01           NaN
1949-04-01           NaN
1949-05-01           NaN
...                  ...
1960-08-01   463.333333
1960-09-01   467.083333
1960-10-01   471.583333
1960-11-01   473.916667
1960-12-01   476.166667

[144 rows x 1 columns]             #Passengers
Month
```

```
1949-01-01          NaN
1949-02-01          NaN
1949-03-01          NaN
1949-04-01          NaN
1949-05-01          NaN
...                 ...
1960-08-01    83.630500
1960-09-01    84.617276
1960-10-01    82.541954
1960-11-01    79.502382
1960-12-01    77.737125

[144 rows x 1 columns]
```

```
#plot rolling statistics
orig = plt.plot(indexedData, color='blue', label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label="Rolling Std")
plt.legend(loc='best')
plt.title("Rolling Mean and Standard Deviation")
plt.show()
```



As you can see, the rolling mean and rolling standard deviation increase with time. Therefore, we can conclude that the time series is not stationary.

```
result = adfuller(indexedData['#Passengers'])
print('ADF Statistic: {}'.format(result[0]))
print('p-value: {}'.format(result[1]))
print('Critical Values:')
for key, value in result[4].items():
    print('\t{}: {}'.format(key, value))
```

```
ADF Statistic: 0.8153688792060472
p-value: 0.991880243437641
Critical Values:
```

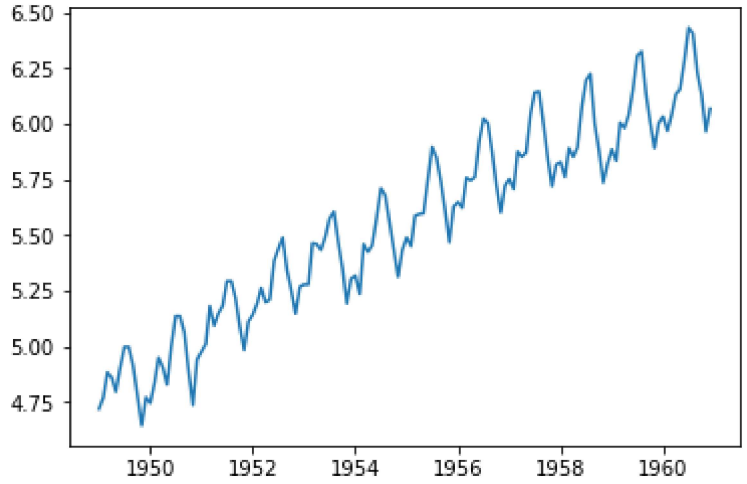
```
1%: -3.4816817173418295
5%: -2.8840418343195267
10%: -2.578770059171598
```

The ADF Statistic is far from the critical values and the p-value is greater than the threshold (0.05). Thus, we can conclude that the time series is not stationary.

Taking the log of the dependent variable is as simple way of lowering the rate at which rolling mean increases.

```
df_log = np.log(indexedData)
plt.plot(df_log)
```

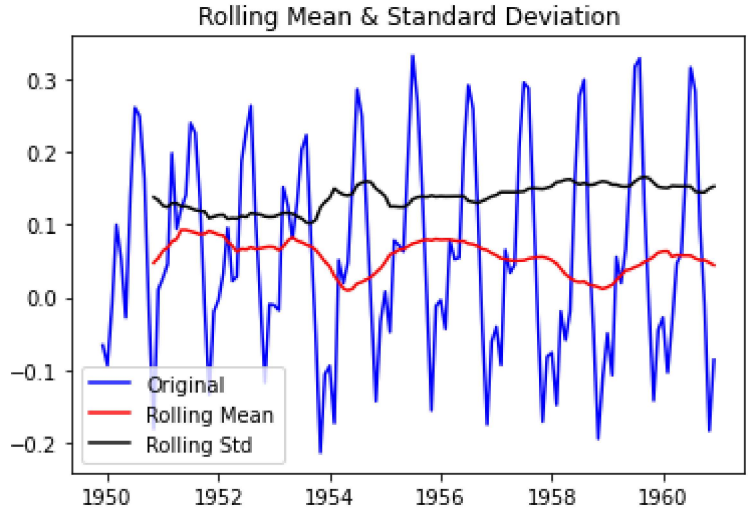
```
[<matplotlib.lines.Line2D at 0x7f304d1648d0>]
```




```
def get_stationarity(timeseries):

    # rolling statistics
    rolling_mean = timeseries.rolling(window=12).mean()
    rolling_std = timeseries.rolling(window=12).std()

    # rolling statistics plot
    original = plt.plot(timeseries, color='blue', label='Original')
    mean = plt.plot(rolling_mean, color='red', label='Rolling Mean')
    std = plt.plot(rolling_std, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    # Dickey-Fuller test:
    result = adfuller(timeseries['#Passengers'])
    print('ADF Statistic: {}'.format(result[0]))
    print('p-value: {}'.format(result[1]))
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t{}: {}'.format(key, value))
```

There are multiple transformations that we can apply to a time series to render it stationary. For instance, we subtract the rolling mean.

```
rolling_mean = df_log.rolling(window=12).mean()
df_log_minus_mean = df_log - rolling_mean
df_log_minus_mean.dropna(inplace=True)
get_stationarity(df_log_minus_mean)
```

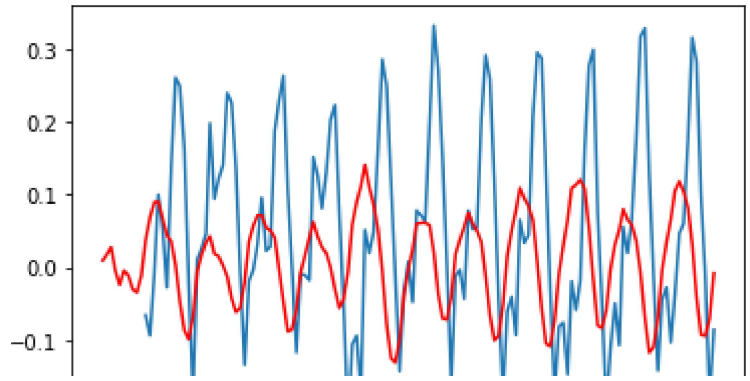



```
ADF Statistic: -3.162907991300889
p-value: 0.02223463000124189
Critical Values:
        1%: -3.4865346059036564
        5%: -2.8861509858476264
        10%: -2.579896092790057
```

As we can see, after subtracting the mean, the rolling mean and standard deviation are approximately horizontal. The p-value is below the threshold of 0.05 and the ADF Statistic is close to the critical values. Therefore, the time series is stationary.

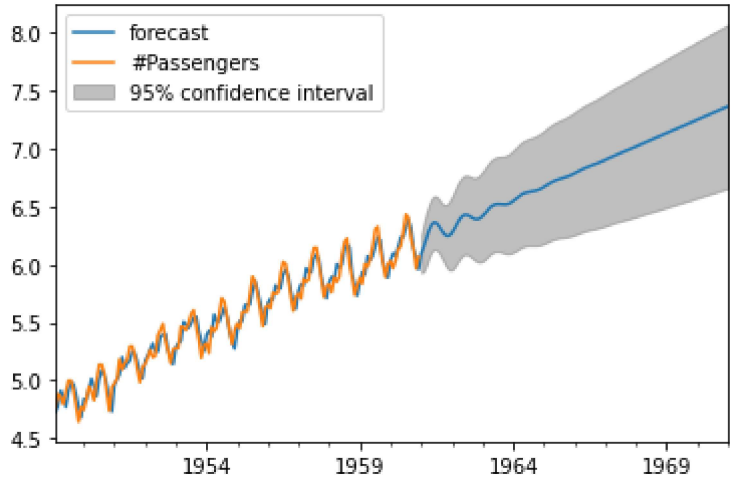
```
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA

decomposition = seasonal_decompose(df_log)
model = ARIMA(df_log, order=(2,1,2))
results = model.fit(disp=-1)
plt.plot(df_log_minus_mean)
plt.plot(results.fittedvalues, color='red')
```

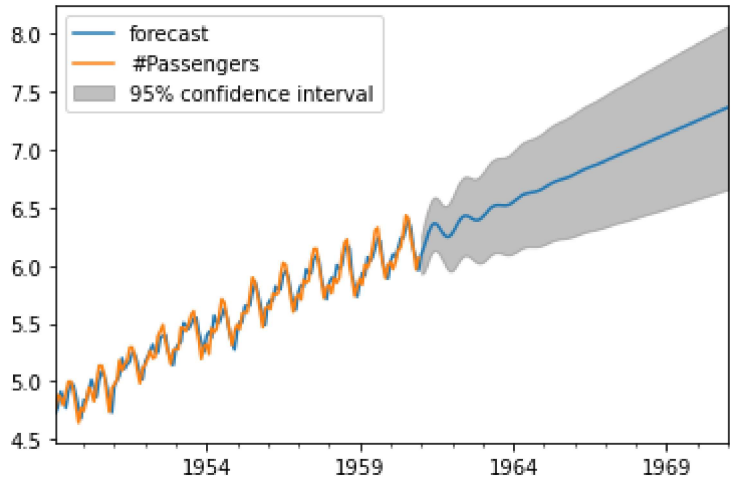
```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/base/tsa_model.py:165: ValueWarni
  % freq, ValueWarning)
[<matplotlib.lines.Line2D at 0x7f304844cdd0>]
```



Given that we have data going for every month going back 12 years and want to forecast the number of passengers for the next 10 years, we use (12 x12)+ (12 x 10) = 264.

```
results.plot_predict(1,264)
```





Experiment – 12

Perform Noise Removal on text by writing appropriate functions for stopword removal and punctuation removal.

Program:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string

def remove_punctuation(sentence):
    for i in sentence:
        if i in string.punctuation:
            sentence = sentence.replace(i, "")
    return sentence

def remove_stopwords(sentence):
    stop_words = set(stopwords.words('english'))
    word_tokens = word_tokenize(example_sent)

    filtered_sentence = []

    for w in word_tokens:
        if w not in stop_words:
            filtered_sentence.append(w)

    return filtered_sentence

example_sent = """We are learning Natural Language Processing (NLP) as part of Fundamentals of Machine Learning (FML) course in our second year B.Tech."""

print(f"Original Sentence : \n{example_sent}")
example_sent = remove_punctuation(example_sent)
print(f"\nSentence after removing punctuations : \n{example_sent}")
example_sent = " ".join(remove_stopwords(example_sent))
print(f"\nSentence after removing stopwords : \n{example_sent}")
```

Output:

Original Sentence :
We are learning Natural Language Processing (NLP) as part of Fundamentals of Machine Learning (FML) course in our second year B.Tech.

Sentence after removing punctuations :
We are learning Natural Language Processing NLP as part of Fundamentals of Machine Learning FML course in our second year BTech

Sentence after removing stopwords :
We learning Natural Language Processing NLP part Fundamentals Machine Learning FML course second year BTech

Experiment – 13

Perform Text Tokenization (words, sentences) on sample text using the following three techniques and write your observations

a. split() method
b. re library
c. nltk library

1. Tokenization using Pythons split() function:

```
# Word Tokenization
text = '''Once there lived a greedy man in a small town. He was very rich, and he loved gold a
nd all things fancy. But he loved his daughter more than anything. One day, he chanced upo
n a fairy. The fairy's hair was caught in a few tree branches. He helped her out, but as his gre
ediness took over, he realised that he had an opportunity to become richer by asking for a w
ish in return (by helping her out). The fairy granted him a wish. He said, "All that I touch sho
uld turn to gold." And his wish was granted by the grateful fairy.'''

tokens = text.split()

print(tokens)
print("No.of tokens : ", len(tokens))
```

Output:

```
['Once', 'there', 'lived', 'a', 'greedy', 'man', 'in', 'a', 'small', 'town.', 'He', 'was', 'very', 'rich,', 'and',
'he', 'loved', 'gold', 'and', 'all', 'things', 'fancy.', 'But', 'he', 'loved', 'his', 'daughter', 'more', 'than',
'anything.', 'One', 'day,', 'he', 'chanced', 'upon', 'a', 'fairy.', 'The', 'fairy's', 'hair', 'was', 'caught',
'in', 'a', 'few', 'tree', 'branches.', 'He', 'helped', 'her', 'out,', 'but', 'as', 'his', 'greediness', 'took',
'over,', 'he', 'realised', 'that', 'he', 'had', 'an', 'opportunity', 'to', 'become', 'richer', 'by', 'asking',
'for', 'a', 'wish', 'in', 'return', '(by', 'helping', 'her', 'out).', 'The', 'fairy', 'granted', 'him', 'a', 'wish.',
'He', 'said,', '"All', 'that', 'I', 'touch', 'should', 'turn', 'to', 'gold."', 'And', 'his', 'wish', 'was',
'granted', 'by', 'the', 'grateful', 'fairy.']
No.of tokens :  103
```

```
# Sentence Tokenization
text = '''Once there lived a greedy man in a small town. He was very rich, and he loved gold a
nd all things fancy. But he loved his daughter more than anything. One day, he chanced upo
n a fairy. The fairy's hair was caught in a few tree branches. He helped her out, but as his gre
ediness took over, he realised that he had an opportunity to become richer by asking for a w
ish in return (by helping her out). The fairy granted him a wish. He said, "All that I touch sho
uld turn to gold." And his wish was granted by the grateful fairy.'''
sentences = text.split('.')

print(sentences)
print("No.of sentences : ", len(sentences))
```

Output:
['Once there lived a greedy man in a small town', ' He was very rich, and he loved gold and all things fancy', ' But he loved his daughter more than anything', ' One day, he chanced upon a fairy', ' The fairy's hair was caught in a few tree branches', ' He helped her out, but as his greediness took over, he realised that he had an opportunity to become richer by asking for a wish in return (by helping her out)', ' The fairy granted him a wish', ' He said, "All that I touch should turn to gold', '" And his wish was granted by the grateful fairy', '']
No.of sentences : 10

2. Tokenization using Regular Expressions (RegEx)

```
Import re

#word tokenization
tokens = re.findall("[\w']+", text)
print(tokens)
print("No.of tokens : ", len(tokens))
```

output:
['Once', 'there', 'lived', 'a', 'greedy', 'man', 'in', 'a', 'small', 'town', 'He', 'was', 'very', 'rich', 'and', 'he', 'loved', 'gold', 'and', 'all', 'things', 'fancy', 'But', 'he', 'loved', 'his', 'daughter', 'more', 'than', 'anything', 'One', 'day', 'he', 'chanced', 'upon', 'a', 'fairy', 'The', 'fairy', 's', 'hair', 'was', 'caught', 'in', 'a', 'few', 'tree', 'branches', 'He', 'helped', 'her', 'out', 'but', 'as', 'his', 'greediness', 'took', 'over', 'he', 'realised', 'that', 'he', 'had', 'an', 'opportunity', 'to', 'become', 'richer', 'by', 'asking', 'for', 'a', 'wish', 'in', 'return', 'by', 'helping', 'her', 'out', 'The', 'fairy', 'granted', 'him', 'a', 'wish', 'He', 'said', 'All', 'that', 'I', 'touch', 'should', 'turn', 'to', 'gold', 'And', 'his', 'wish', 'was', 'granted', 'by', 'the', 'grateful', 'fairy']
No.of tokens : 104

```
#sentence tokenization
sentences = re.compile('[.?!] ').split(text)
print(sentences)
print("No.of sentences : ", len(sentences))
```

Output:
['Once there lived a greedy man in a small town', 'He was very rich, and he loved gold and all things fancy', 'But he loved his daughter more than anything', 'One day, he chanced upon a fairy', 'The fairy's hair was caught in a few tree branches', 'He helped her out, but as his greediness took over, he realised that he had an opportunity to become richer by asking for a wish in return (by helping her out)', 'The fairy granted him a wish', 'He said, "All that I touch should turn to gold." And his wish was granted by the grateful fairy.']
No.of sentences : 8

3. Tokenization using NLTK:

```
import nltk
nltk.download('punkt')

#word tokenization
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
print(tokens)
print("No.of tokens : ", len(tokens))
```

Output:

```
['Once', 'there', 'lived', 'a', 'greedy', 'man', 'in', 'a', 'small', 'town', '.', 'He', 'was', 'very', 'rich', ',', 'and', 'he', 'loved', 'gold', 'and', 'all', 'things', 'fancy', '.', 'But', 'he', 'loved', 'his', 'daughter', 'more', 'than', 'anything', '.', 'One', 'day', ',', 'he', 'chanced', 'upon', 'a', 'fairy', '.', 'The', 'fairy', '’', 's', 'hair', 'was', 'caught', 'in', 'a', 'few', 'tree', 'branches', '.', 'He', 'helped', 'her', 'out', ',', 'but', 'as', 'his', 'greediness', 'took', 'over', ',', 'he', 'realised', 'that', 'he', 'had', 'an', 'opportunity', 'to', 'become', 'richer', 'by', 'asking', 'for', 'a', 'wish', 'in', 'return', '(', 'by', 'helping', 'her', 'out', ')', '.', 'The', 'fairy', 'granted', 'him', 'a', 'wish', '.', 'He', 'said', ',', '“', 'All', 'that', 'I', 'touch', 'should', 'turn', 'to', 'gold.', '”', 'And', 'his', 'wish', 'was', 'granted', 'by', 'the', 'grateful', 'fairy', '.']
No.of tokens :  122
```

```
#sentence tokenization
from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(text)
print(sentences)
print("No.of sentences : ", len(sentences))
```

output:

```
['Once there lived a greedy man in a small town.', 'He was very rich, and he loved gold and all things fancy.', 'But he loved his daughter more than anything.', 'One day, he chanced upon a fairy.', 'The fairy’s hair was caught in a few tree branches.', 'He helped her out, but as his greediness took over, he realised that he had an opportunity to become richer by asking for a wish in return (by helping her out).', 'The fairy granted him a wish.', 'He said, “All that I touch should turn to gold.” And his wish was granted by the grateful fairy.']
No.of sentences :  8
```

Experiment – 14

Implement a function which takes a sentence and returns the stemmed sentence (hint : use PorterStemmer of NLTK)

Program:

```
from nltk.stem import PorterStemmer
porter = PorterStemmer()

sentence="Pythoners are very intelligent and work very pythonly and now they are pythonin
g their way to success."

from nltk.tokenize import sent_tokenize, word_tokenize

def stemSentence(sentence):
   token_words=word_tokenize(sentence)
   print(token_words)
   stem_sentence=[]
   for word in token_words:
      stem_sentence.append(porter.stem(word))
      stem_sentence.append(" ")
   return "".join(stem_sentence)

x=stemSentence(sentence)
print("Sentence after stemming :", x)
```

output:

['Pythoners', 'are', 'very', 'intelligent', 'and', 'work', 'very', 'pythonly', 'and', 'now', 'they', 'are', 'pythoning', 'their', 'way', 'to', 'success', '.']

Sentence after stemming : python are veri intellig and work veri pythonli and now they are python their way to success .

Experiment – 15

Implement a function which takes a sentence and returns the lemmatized sentence (hint : use WordNetLemmatizer of NLTK)

Program:

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

sentence = "He was running and eating at same time. He has bad habit of swimming after pl
aying long hours in the Sun."
punctuations="?:!.,;"

from nltk.tokenize import sent_tokenize, word_tokenize

def lemmatizeSentence(sentence):
   token_words=word_tokenize(sentence)
   print(token_words)
   lemma_sentence=[]
   for word in token_words:
     lemma_sentence.append(wordnet_lemmatizer.lemmatize(word))
     lemma_sentence.append(" ")
   return "".join(lemma_sentence)

x=lemmatizeSentence(sentence)
print("Sentence after Lemmatization :", x)
```

Output:

['He', 'was', 'running', 'and', 'eating', 'at', 'same', 'time', '.', 'He', 'has', 'bad', 'habit', 'of', 'swimming', 'after', 'playing', 'long', 'hours', 'in', 'the', 'Sun', '.']

Sentence after Lemmatization : He wa running and eating at same time . He ha bad habit of swimming after playing long hour in the Sun .

Experiment – 16

Create an N-gram language model by using Reuters corpus of the NLTK library

Program:

```
from nltk.corpus import reuters
from nltk import bigrams, trigrams
from collections import Counter, defaultdict

# Create a placeholder for model
model = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurance
for sentence in reuters.sents():
   for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
      model[(w1, w2)][w3] += 1

# Let's transform the counts to probabilities
for w1_w2 in model:
   total_count = float(sum(model[w1_w2].values()))
   for w3 in model[w1_w2]:
      model[w1_w2][w3] /= total_count
```

Output :

```
{'public': 0.05555555555555555,
'European': 0.05555555555555555,
'Bank': 0.05555555555555555,
'price': 0.1111111111111111,
'emirate': 0.05555555555555555,
'overseas': 0.05555555555555555,
'newspaper': 0.05555555555555555,
'company': 0.16666666666666666,
'Turkish': 0.05555555555555555,
'increase': 0.05555555555555555,
'options': 0.05555555555555555,
'Higher': 0.05555555555555555,
'pound': 0.05555555555555555,
'Italian': 0.05555555555555555,
'time': 0.05555555555555555}
```

Experiment – 17

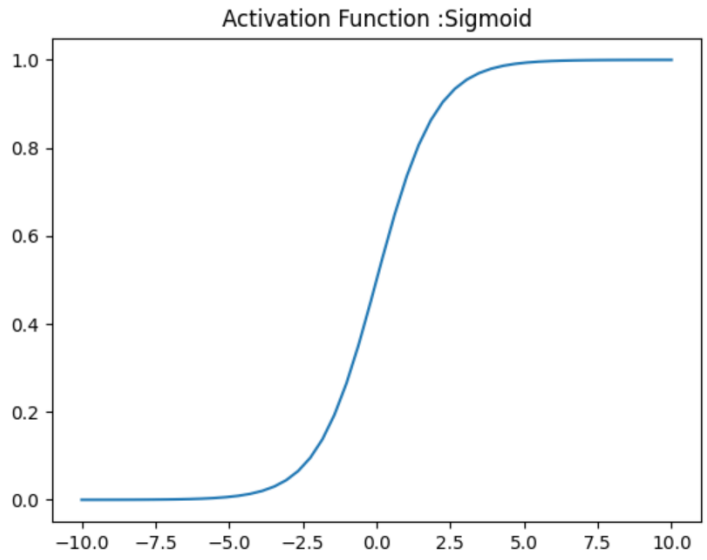
Write python implementations for the following activation functions and also plot them for a linearly spaced 50 values ranging from -10 to 10

d. Sigmoid
e. Tanh
f. Relu
g. Softmax

Sigmoid :

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
   return 1 / (1+np.exp(-x))

x = np.linspace(-10, 10)
plt.plot(x, sigmoid(x))
plt.axis('tight')
plt.title('Activation Function :Sigmoid')
plt.show()
```

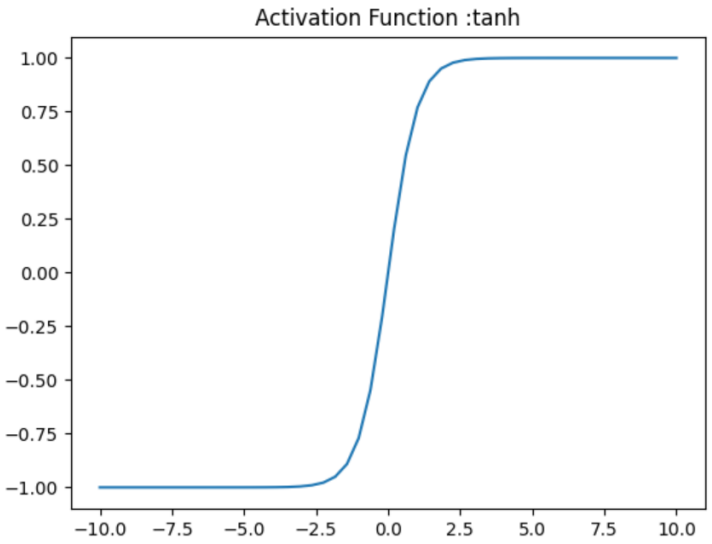
Output:



tanh

```
import numpy as np
import matplotlib.pyplot as plt

def tanh(x):
   return np.tanh(x)

x = np.linspace(-10, 10)
plt.plot(x, tanh(x))
plt.axis('tight')
plt.title('Activation Function :tanh')
plt.show()
```

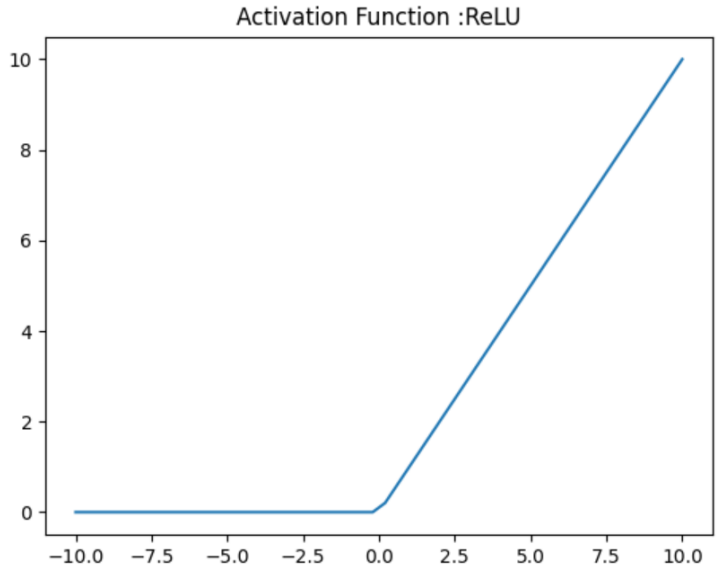
Output:



ReLU:

```
import numpy as np
import matplotlib.pyplot as plt

def RELU(x):
   x1=[]
   for i in x:
      if i<0:
         x1.append(0)
      else:
         x1.append(i)

   return x1

x = np.linspace(-10, 10)
plt.plot(x, RELU(x))
plt.axis('tight')
plt.title('Activation Function :ReLU')
plt.show()
```

Output:



Softmax:

```
import numpy as np
import matplotlib.pyplot as plt

def softmax(x):
    ''' Compute softmax values for each sets of scores in x. '''
    return np.exp(x) / np.sum(np.exp(x), axis=0)

x = np.linspace(-10, 10)
plt.plot(x, softmax(x))
plt.axis('tight')
plt.title('Activation Function :Softmax')
plt.show()
```