

Machine Learning

Index

Sr. No.	Title	Date	Sign
1a	Design a simple machine learning model to train the training instances and test the same.		
1b	Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file		
2a	Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.		
2b	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.		
3a	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets		
3b	Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.		
4a	For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm		
4b	For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm.		
5a	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.		
5b	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set.		
6a	Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.		
6b	Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.		

7	Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix		
8a	Exploratory Data Analysis (EDA)		
8b	Time Series Analysis		
9	Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.		
10	Perform Text pre-processing, Text clustering, classification with Prediction, Test Score and Confusion Matrix		

Practical 1

A. Design a simple machine learning model to train the training instances and test the same.

```
In [1]: # Generating the Training Set

# python library to generate random numbers
from random import randint

# the limit within which random numbers are generated
TRAIN_SET_LIMIT = 1000

# to create exactly 100 data items
TRAIN_SET_COUNT = 100

# list that contains input and corresponding output
TRAIN_INPUT = list()
TRAIN_OUTPUT = list()
```

```
In [4]: # Loop to create 100 data items with three columns each
for i in range(TRAIN_SET_COUNT):
    a = randint(0, TRAIN_SET_LIMIT)
    b = randint(0, TRAIN_SET_LIMIT)
    c = randint(0, TRAIN_SET_LIMIT)

    # creating the output for each data item
    op = a + (2 * b) + (3 * c)
    TRAIN_INPUT.append([a, b, c])

    # adding each output to output list
    TRAIN_OUTPUT.append(op)
```

```
In [6]: # printing first 10 records
TRAIN_OUTPUT[:10]
```

```
Out[6]: [1122, 2518, 5291, 3471, 3522, 4008, 4547, 3760, 4414, 1890]
```

```
In [7]: # printing first 10 records
TRAIN_INPUT[:10]
```

```
Out[7]: [[306, 270, 92],
          [477, 233, 525],
          [514, 908, 987],
          [576, 84, 909],
          [517, 664, 559],
          [652, 424, 836],
          [672, 559, 919],
          [805, 210, 845],
          [221, 763, 889],
          [300, 666, 86]]
```

The Model can be created in two steps:-

1. Training the model with Training Data

2. Testing the model with Test Data

```
In [8]: # Training the Model  
# The data that was created using the above code is used to train the model  
  
# Sk-Learn contains the linear regression model  
from sklearn.linear_model import LinearRegression  
  
# Initialize the linear regression model  
predictor = LinearRegression(n_jobs=-1)  
  
# Fill the Model with the Data  
predictor.fit(X = TRAIN_INPUT, y = TRAIN_OUTPUT)
```

Out[8]: LinearRegression(n_jobs=-1)

Testing the Data

The testing is done Manually. Testing can be done using some random data and testing if the model gives the correct result for the input data.

```
In [9]: # Random Test data  
X_TEST = [[ 10, 20, 30 ]]    #---> 10 + 20*2 + 30*3 = 140.  
  
# Predict the result of X_TEST which holds testing data  
outcome = predictor.predict(X = X_TEST)  
  
# Predict the coefficients  
coefficients = predictor.coef_  
  
# Print the result obtained for the test data  
print('Outcome : {}\nCoefficients : {}'.format(outcome, coefficients))
```

Outcome : [140.]
Coefficients : [1. 2. 3.]

In []:

B. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.

```
In [2]: import random
import csv
```

```
attributes = [['Sunny', 'Rainy'],
              ['Warm', 'Cold'],
              ['Normal', 'High'],
              ['Strong', 'Weak'],
              ['Warm', 'Cool'],
              ['Same', 'Change']]
```

```
num_attributes = len(attributes)
```

```
In [3]: print ("\n The most general hypothesis : ['?', '?', '?', '?', '?', '?']\n")
print ("\n The most specific hypothesis : ['0', '0', '0', '0', '0', '0']\n")
```

```
The most general hypothesis : ['?', '?', '?', '?', '?', '?']
```

```
The most specific hypothesis : ['0', '0', '0', '0', '0', '0']
```

```
In [4]: a = []
print("\n The Given Training Data Set \n")

with open('/content/data.csv', 'r') as csvFile:
    reader = csv.reader(csvFile)
    for row in reader:
        a.append (row)
    print(row)
```

```
The Given Training Data Set
```

```
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
```

```
In [5]: print("\n The initial value of hypothesis: ")
hypothesis = ['0'] * num_attributes
print(hypothesis)

# Comparing with First Training Example
for j in range(0,num_attributes):
    hypothesis[j] = a[0][j];

# Comparing with Remaining Training Examples of Given Data Set

print("\n Find S: Finding a Maximally Specific Hypothesis\n")

for i in range(0,len(a)):
    if a[i][num_attributes]=='Yes':
```

```
        for j in range(0,num_attributes):
            if a[i][j]!=hypothesis[j]:
                hypothesis[j]='?'
            else :
                hypothesis[j]= a[i][j]
        print(" For Training Example No :{0} the hypothesis is ".format(i),hypothesis)

print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")
print(hypothesis)
```

The initial value of hypothesis:
['0', '0', '0', '0', '0', '0']

Find S: Finding a Maximally Specific Hypothesis

```
For Training Example No :0 the hypothesis is  ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
For Training Example No :1 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For Training Example No :2 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
For Training Example No :3 the hypothesis is  ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

The Maximally Specific Hypothesis for a given Training Examples :

```
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

In []:

Practical 2

A. Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking.

Principal component analysis, or PCA, is a statistical technique to convert high dimensional data to low dimensional data by selecting the most important features that capture maximum information about the dataset. The features are selected on the basis of variance that they cause in the output. The feature that causes highest variance is the first principal component. The feature that is responsible for second highest variance is considered the second principal component, and so on. It is important to mention that principal components do not have any correlation with each other.

```
In [22]: import numpy as np  
import pandas as pd
```

```
In [23]: url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"  
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']  
dataset = pd.read_csv(url, names=names)
```

```
In [24]: dataset.head()
```

```
Out[24]:
```

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Preprocessing

The first preprocessing step is to divide the dataset into a feature set and corresponding labels. The following script performs this task:

```
In [25]: X = dataset.drop('Class', 1)  
y = dataset['Class']
```

```
<ipython-input-25-a134714d2c8b>:1: FutureWarning: In a future version of pandas al  
l arguments of DataFrame.drop except for the argument 'labels' will be keyword-only  
y  
X = dataset.drop('Class', 1)
```

```
In [26]: y
```

```
Out[26]: 0      Iris-setosa
          1      Iris-setosa
          2      Iris-setosa
          3      Iris-setosa
          4      Iris-setosa
          ...
         145     Iris-virginica
         146     Iris-virginica
         147     Iris-virginica
         148     Iris-virginica
         149     Iris-virginica
Name: Class, Length: 150, dtype: object
```

The script above stores the feature sets into the X variable and the series of corresponding labels in to the y variable.

The next preprocessing step is to divide data into training and test sets. Execute the following script to do so:

```
In [27]: # Splitting the dataset into the 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_st
```

PCA performs best with a normalized feature set. We will perform standard scalar normalization to normalize our feature set. To do this, execute the following code:

```
In [28]: from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_s = sc.fit_transform(X_train)
X_test_s = sc.transform(X_test)
```

Applying PCA

It is only a matter of three lines of code to perform PCA using Python's Scikit-Learn library. The PCA class is used for this purpose. PCA depends only upon the feature set and not the label data. Therefore, PCA can be considered as an unsupervised machine learning technique.

Performing PCA using Scikit-Learn is a two-step process:

Initialize the PCA class by passing the number of components to the constructor. Call the fit and then transform methods by passing the feature set to these methods. The transform method returns the specified number of principal components.

```
In [29]: from sklearn.decomposition import PCA

pca = PCA()
X_train = pca.fit_transform(X_train_s)
X_test = pca.transform(X_test_s)
```

In the code above, we create a PCA object named pca.

We did not specify the number of components in the constructor. Hence, all four of the features in the feature set will be returned for both the training and test sets.

The PCA class contains explained_varianceratio which returns the variance caused by each of the principal components.

Execute the following line of code to find the "explained variance ratio".

```
In [30]: explained_variance = pca.explained_variance_ratio_
```

```
In [31]: explained_variance
```

```
Out[31]: array([0.72226528, 0.23974795, 0.03338117, 0.0046056 ])
```

It can be seen that first principal component is responsible for 72.22% variance. Similarly, the second principal component causes 23.9% variance in the dataset.

Collectively we can say that $(72.22 + 23.9) / 4 = 28 / 4 = 7$ 96.21% percent of the classification information contained in the feature set is captured by the first two principal components.

Let's first try to use 1 principal component to train our algorithm. To do so, execute the following code:

```
In [32]: from sklearn.decomposition import PCA  
  
pca = PCA(n_components=1)  
X_train = pca.fit_transform(X_train)  
X_test = pca.transform(X_test)
```

Training and Making Predictions

In this case we'll use random forest classification for making the predictions.

```
In [33]: from sklearn.ensemble import RandomForestClassifier  
  
classifier = RandomForestClassifier(max_depth=2, random_state=0)  
classifier.fit(X_train, y_train)  
  
# Predicting the Test set results  
y_pred = classifier.predict(X_test)
```

```
In [34]: #Performance Evaluation  
  
from sklearn.metrics import confusion_matrix  
from sklearn.metrics import accuracy_score  
  
cm = confusion_matrix(y_test, y_pred)  
print(cm)  
print()  
print('Accuracy', accuracy_score(y_test, y_pred))  
  
[[11  0  0]  
 [ 0 12  1]  
 [ 0  1  5]]
```

Accuracy 0.9333333333333333

It can be seen from the output that with only one feature, the random forest algorithm is able to correctly predict 28 (11+12+5) out of 30 instances, resulting in 93.33% accuracy.

Now lets check, Results with 2 and 3 Principal Components

```
In [44]: # Splitting the dataset into the 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_st
```

```
In [45]: from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_s = sc.fit_transform(X_train)
X_test_s = sc.transform(X_test)
```

```
In [46]: from sklearn.decomposition import PCA

pca = PCA()
X_train = pca.fit_transform(X_train_s)
X_test = pca.transform(X_test_s)
```

```
In [48]: explained_variance = pca.explained_variance_ratio_
explained_variance
```

```
Out[48]: array([0.72226528, 0.23974795, 0.03338117, 0.0046056 ])
```

```
In [49]: # 2 components
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

```
In [50]: from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(max_depth=2, random_state=0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

```
In [51]: #Performance Evaluation

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)
print()
print('Accuracy', accuracy_score(y_test, y_pred))

[[11  0  0]
 [ 0  9  4]
 [ 0  2  4]]
```

Accuracy 0.8

Now for 3 components same process

```
In [52]: # Splitting the dataset into the 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_st
```

```
In [53]: from sklearn.preprocessing import StandardScaler  
  
sc = StandardScaler()  
X_train_s = sc.fit_transform(X_train)  
X_test_s = sc.transform(X_test)
```

```
In [54]: from sklearn.decomposition import PCA  
  
pca = PCA()  
X_train = pca.fit_transform(X_train_s)  
X_test = pca.transform(X_test_s)
```

```
In [55]: explained_variance = pca.explained_variance_ratio_  
explained_variance
```

```
Out[55]: array([0.72226528, 0.23974795, 0.03338117, 0.0046056 ])
```

```
In [56]: # 3 components  
from sklearn.decomposition import PCA  
  
pca = PCA(n_components=3)  
X_train = pca.fit_transform(X_train)  
X_test = pca.transform(X_test)
```

```
In [57]: from sklearn.ensemble import RandomForestClassifier  
  
classifier = RandomForestClassifier(max_depth=2, random_state=0)  
classifier.fit(X_train, y_train)  
  
# Predicting the Test set results  
y_pred = classifier.predict(X_test)
```

```
In [58]: #Performance Evaluation  
  
from sklearn.metrics import confusion_matrix  
from sklearn.metrics import accuracy_score  
  
cm = confusion_matrix(y_test, y_pred)  
print(cm)  
print()  
print('Accuracy', accuracy_score(y_test, y_pred))
```

[[11 0 0]
[0 8 5]
[0 1 5]]

```
Accuracy 0.8
```

```
In [ ]:
```

B. For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Step1: Load Data set

Step2: Initialize General Hypothesis and Specific Hypothesis.

Step3: For each training example

Step4: If example is positive example

if attribute_value == hypothesis_value:

Do nothing

else:

replace attribute value with '?' (Basically generalizing it)

Step5: If example is Negative example

Make generalize hypothesis more specific.

```
In [1]: import csv

with open("trainingexamples.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)

    specific = data[1][:-1]
    general = [[? for i in range(len(specific))] for j in range(len(specific))]

    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"

    print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")
    print(specific)
    print(general)

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
```

```

    if j != '?':
        gh.append(i)
        break
print("\nFinal Specific hypothesis:\n", specific)
print("\nFinal General hypothesis:\n", gh)

```

Step 1 of Candidate Elimination Algorithm

```

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[[ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'],
 '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'],
 [ '?', '?', '?', '?', '?', '?']]

```

Step 2 of Candidate Elimination Algorithm

```

['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[[ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'],
 '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'],
 [ '?', '?', '?', '?', '?', '?']]

```

Step 3 of Candidate Elimination Algorithm

```

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[[ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'],
 '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'],
 [ '?', '?', '?', '?', '?', '?']]

```

Step 4 of Candidate Elimination Algorithm

```

['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[[ 'Sunny', '?', '?', '?', '?', '?'], [ '?', 'Warm', '?', '?', '?', '?'], [ '?', '?', '?',
 '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'],
 [ '?', '?', '?', '?', '?', 'Same']]

```

Step 5 of Candidate Elimination Algorithm

```

['Sunny', 'Warm', '?', 'Strong', '?', '?']
[[ 'Sunny', '?', '?', '?', '?', '?'], [ '?', 'Warm', '?', '?', '?', '?'], [ '?', '?', '?',
 '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'], [ '?', '?', '?', '?', '?', '?'],
 [ '?', '?', '?', '?', '?', '?']]

```

Final Specific hypothesis:

```

['Sunny', 'Warm', '?', 'Strong', '?', '?']

```

Final General hypothesis:

```

[['Sunny', '?', '?', '?', '?', '?'], [ '?', 'Warm', '?', '?', '?', '?']]

```

In []:

Practical 3

A. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
In [2]: import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns
```

```
In [4]: df=pd.read_csv('train.csv')  
df.head()
```

```
Out[4]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38.0	1	0	PC 17599	71.2833	C85
2	3	1	3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN
3	4	1	1	Allen, Mr. William Henry	male	35.0	1	0	113803	53.1000	C123
4	5	0	3				0	0	373450	8.0500	NaN

```
In [5]: df.ndim
```

```
Out[5]: 2
```

```
In [6]: df.shape
```

```
Out[6]: (891, 12)
```

```
In [7]: df.columns
```

```
Out[7]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',  
'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],  
dtype='object')
```

```
In [8]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId 891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object  
 9   Fare          891 non-null    float64 
 10  Cabin        204 non-null    object  
 11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
In [9]: df.isna().sum()
```

```
Out[9]: PassengerId      0
Survived        0
Pclass          0
Name            0
Sex             0
Age            177
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          687
Embarked       2
dtype: int64
```

```
In [10]: df.isna().any()
```

```
Out[10]: PassengerId    False
Survived       False
Pclass         False
Name           False
Sex            False
Age            True
SibSp          False
Parch          False
Ticket         False
Fare           False
Cabin          True
Embarked       True
dtype: bool
```

```
In [11]: df.describe().transpose()
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
PassengerId	891.0	446.000000	257.353842	1.00	223.5000	446.0000	668.5	891.0000
Survived	891.0	0.383838	0.486592	0.00	0.0000	0.0000	1.0	1.0000
Pclass	891.0	2.308642	0.836071	1.00	2.0000	3.0000	3.0	3.0000
Age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
SibSp	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
Parch	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
Fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

In [12]:

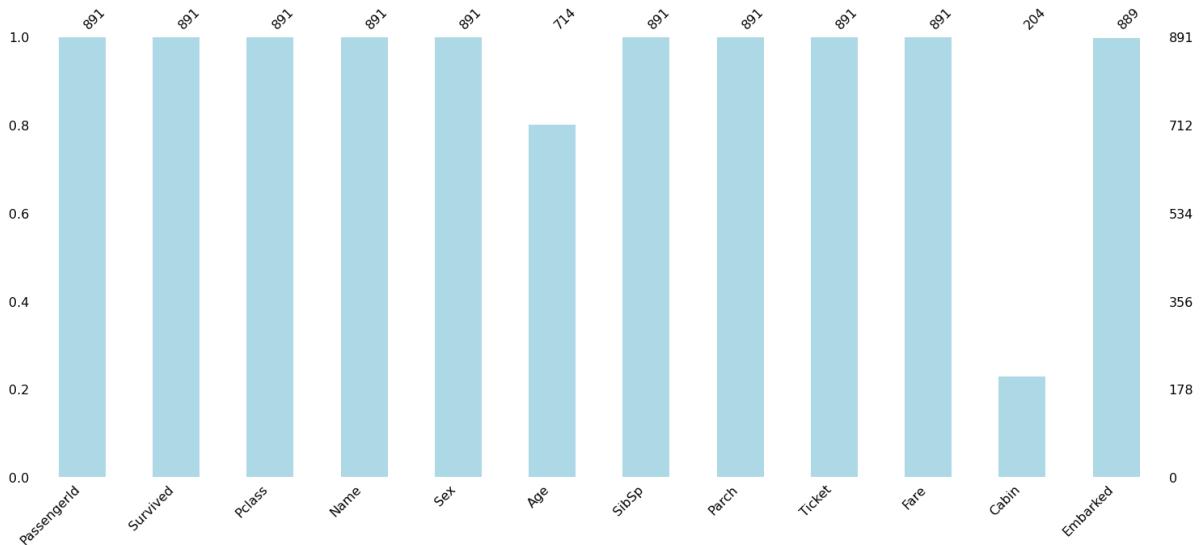
```
null_df=pd.DataFrame()
null_df['Features']=df.isnull().sum().index
null_df['Null values']=df.isnull().sum().values
null_df['% Null values']=(df.isnull().sum().values / df.shape[0])*100
null_df.sort_values(by='% Null values',ascending=False)
```

Out[12]:

	Features	Null values	% Null values
10	Cabin	687	77.104377
5	Age	177	19.865320
11	Embarked	2	0.224467
0	PassengerId	0	0.000000
1	Survived	0	0.000000
2	Pclass	0	0.000000
3	Name	0	0.000000
4	Sex	0	0.000000
6	SibSp	0	0.000000
7	Parch	0	0.000000
8	Ticket	0	0.000000
9	Fare	0	0.000000

In [13]:

```
import missingno as no
no.bar(df,color='lightblue')
plt.show()
```



```
In [14]: df['Age'].fillna(df.Age.median(), inplace=True)
df.Age.isna().any()
```

```
Out[14]: False
```

```
In [15]: df.drop(columns='Cabin' , inplace=True)
```

```
In [16]: df.dropna(subset=['Embarked'],inplace=True)
```

```
In [17]: df.shape
```

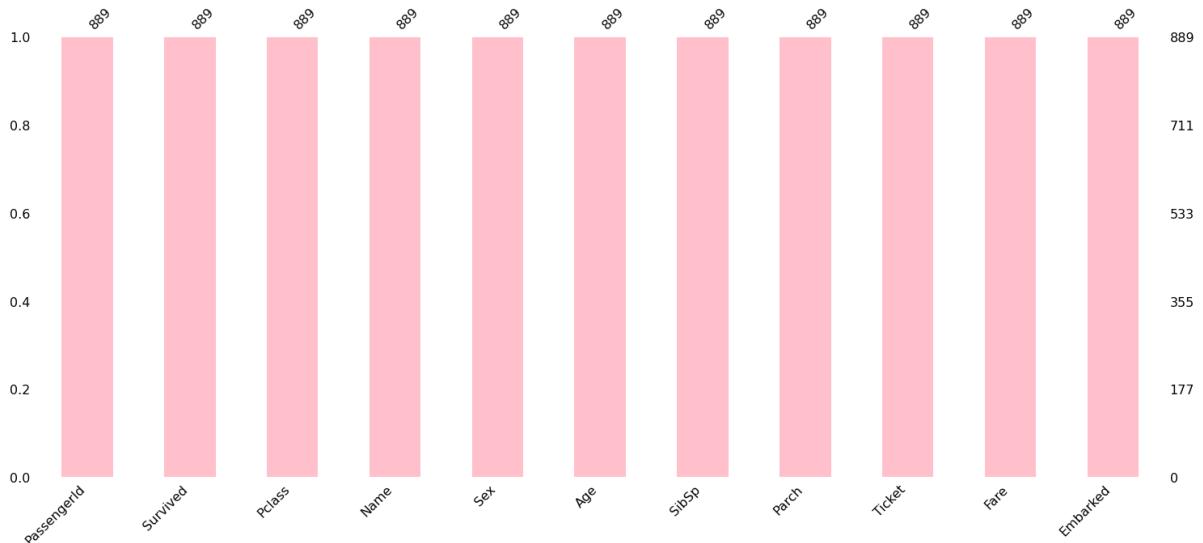
```
Out[17]: (889, 11)
```

```
In [18]: df.isna().sum()
```

```
Out[18]: PassengerId      0
Survived          0
Pclass            0
Name              0
Sex               0
Age               0
SibSp            0
Parch            0
Ticket           0
Fare             0
Embarked         0
dtype: int64
```

```
In [19]: no.bar(df,color='pink')
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5228f30460>
```



Data Preprocessing

```
In [21]: import yellowbrick
from sklearn.preprocessing import LabelEncoder
label_Enc =LabelEncoder()
```

```
In [22]: df.Sex =label_Enc.fit_transform(df.Sex)
```

```
In [23]: label_Enc.classes_
```

```
Out[23]: array(['female', 'male'], dtype=object)
```

```
In [24]: df.head()
```

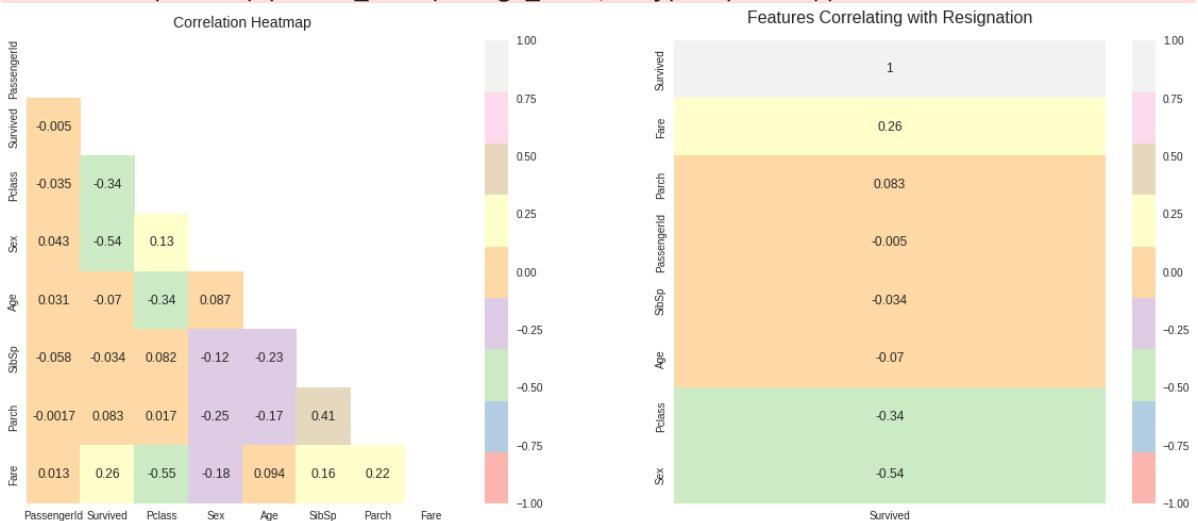
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarke
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	0	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	

```
In [25]: fig,ax=plt.subplots(ncols=2,figsize=(20,8))
resign_corr = df.corr()
mask = np.triu(np.ones_like(resign_corr, dtype=np.bool))
cat_heatmap = sns.heatmap(df.corr(), mask=mask, vmin=-1, vmax=1, annot=True, ax=ax[0])
cat_heatmap.set_title('Correlation Heatmap', fontdict={'fontsize':14}, pad=12);
heatmap = sns.heatmap(resign_corr[['Survived']].sort_values(by='Survived', ascending=False),
                      mask=mask, annot=True, ax=ax[1])
heatmap.set_title('Features Correlating with Resignation', fontdict={'fontsize':16})
```

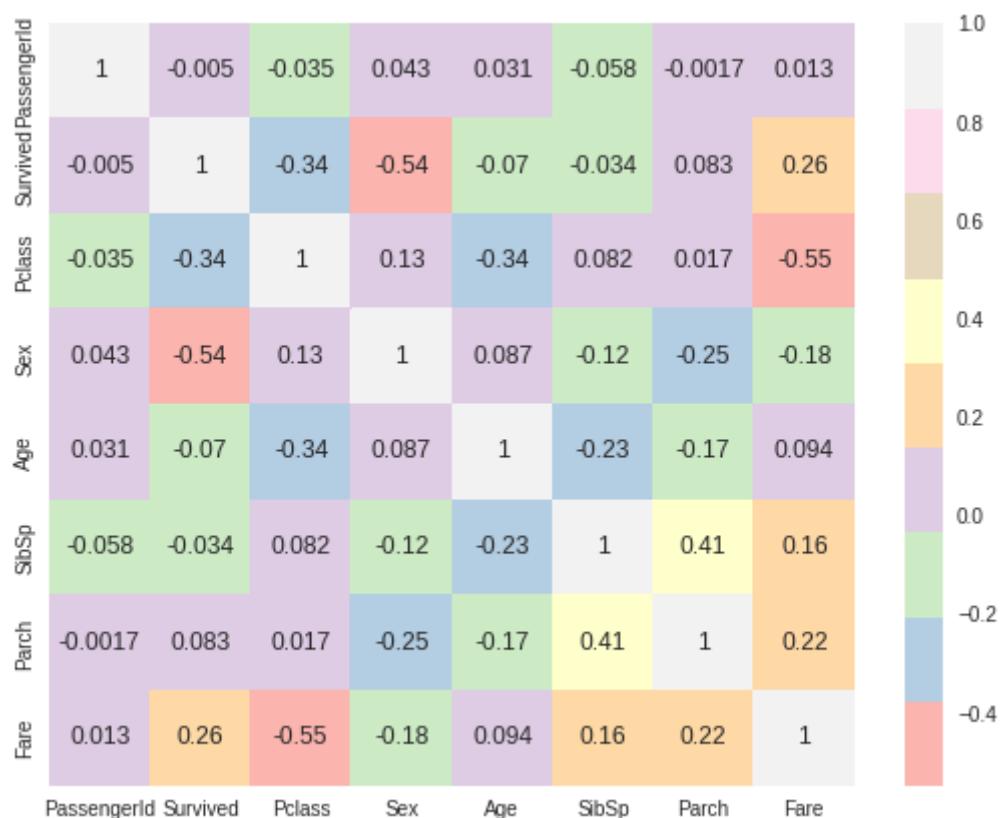
```
<ipython-input-25-0960bdc0efd7>:3: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifically wanted the numpy scalar type, use `np.bool_` here.
```

Deprecated in NumPy 1.20; for more details and guidance: <https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations>

```
mask = np.triu(np.ones_like(resign_corr, dtype=np.bool))
```

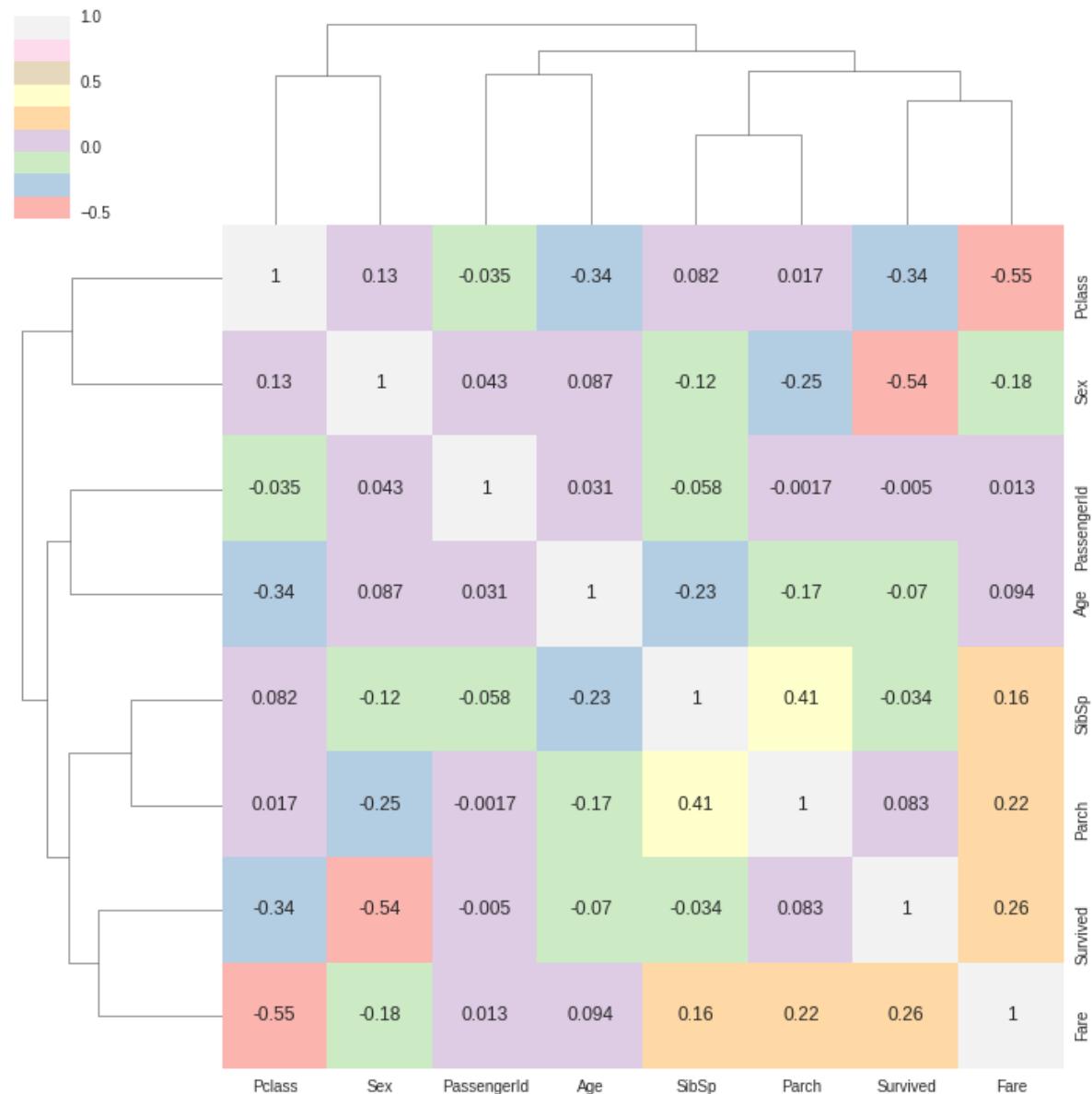


```
In [26]: plt.figure(figsize=(9,7))
sns.heatmap(df.corr(), annot=True, cmap='Pastel1')
plt.show()
```



```
In [27]: sns.clustermap(df.corr(), annot=True, cmap='Pastel1')
```

```
Out[27]: <seaborn.matrix.ClusterGrid at 0x7f5224c77ca0>
```



Dataset Splitting

```
In [28]: X=df.drop(['Survived', 'Name', 'Ticket', 'Embarked'], axis=1)  
y=df['Survived']
```

```
In [29]: X[:3]
```

```
Out[29]:
```

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare
0	1	3	1	22.0	1	0	7.2500
1	2	1	0	38.0	1	0	71.2833
2	3	3	0	26.0	0	0	7.9250

```
In [30]: y[:5]
```

```
Out[30]: 0    0  
1    1  
2    1  
3    1  
4    0  
Name: Survived, dtype: int64
```

```
In [31]: 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,stratify=y,random_state=42)
```

Gaussian Naive Bayes Classifier

```
In [32]: from sklearn.metrics import confusion_matrix, classification_report,accuracy_score  
from sklearn.metrics import recall_score, precision_score, f1_score
```

```
In [33]: from sklearn.naive_bayes import GaussianNB
```

```
In [34]: gnb_clf=GaussianNB()
```

```
In [35]: gnb_clf.fit(X_train, y_train)
```

```
Out[35]: GaussianNB()
```

```
In [36]: y_pred = gnb_clf.predict(X_test)
```

```
In [37]: print("Accuracy Score : ",accuracy_score(y_test,y_pred))
```

```
Accuracy Score : 0.7921348314606742
```

```
In [38]: print("Recall Score",recall_score(y_test,y_pred))
```

```
Recall Score 0.7058823529411765
```

```
In [39]: print("Precision Score : ",precision_score(y_test,y_pred))
```

```
Precision Score : 0.7384615384615385
```

```
In [40]: print("F1 Score : ",f1_score(y_test,y_pred))
```

```
F1 Score : 0.7218045112781954
```

Confusion Matrix

```
In [41]: confusion_matrix(y_test, y_pred)
```

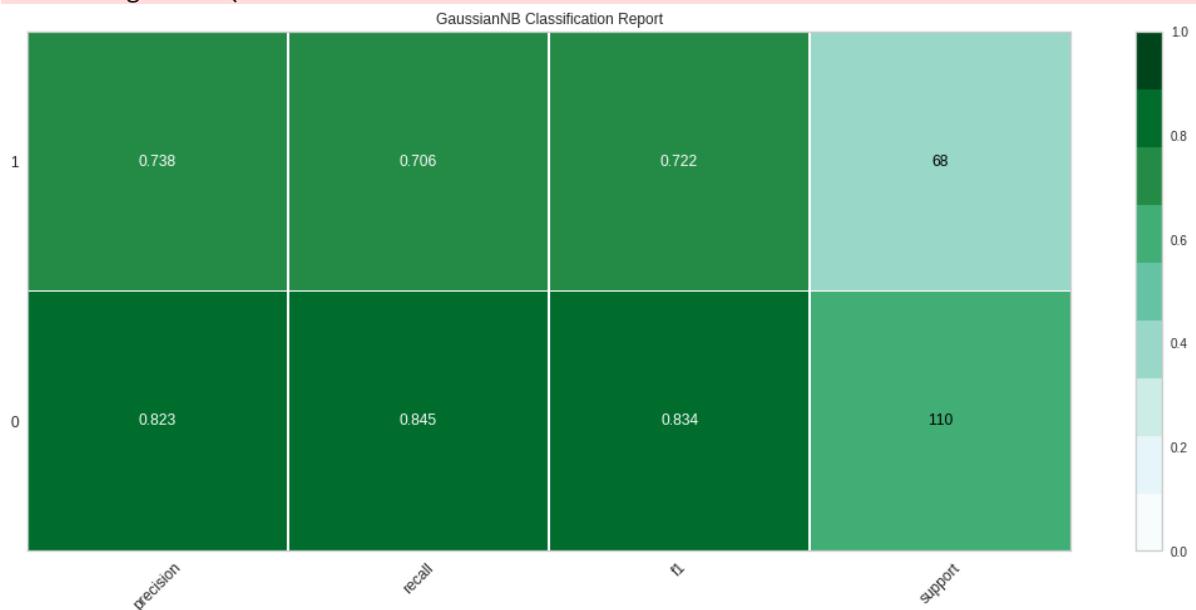
```
Out[41]: array([[93, 17],  
 [20, 48]])
```

```
In [42]: print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.82	0.85	0.83	110
1	0.74	0.71	0.72	68
accuracy			0.79	178
macro avg	0.78	0.78	0.78	178
weighted avg	0.79	0.79	0.79	178

```
In [43]: import yellowbrick as yb
plt.figure(figsize=(15,7))
visualizer = yb.classifier.classification_report(gnb_clf, X_train, y_train, X_test,
visualizer.show()
plt.show()
```

/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but GaussianNB was fitted with feature names
warnings.warn(



```
In [ ]:
```

B. Write a program to implement Decision Tree and Random forest with Prediction, Test Score and Confusion Matrix.

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
```

Data Collection and Processing

```
In [2]: # Loading the csv data to a Pandas DataFrame
gold_data = pd.read_csv('/content/gld_price_data.csv')
```

```
In [3]: # print first 5 rows in the dataframe
gold_data.head()
```

```
Out[3]:
```

	Date	SPX	GLD	USO	SLV	EUR/USD
0	1/2/2008	1447.160034	84.860001	78.470001	15.180	1.471692
1	1/3/2008	1447.160034	85.570000	78.370003	15.285	1.474491
2	1/4/2008	1411.630005	85.129997	77.309998	15.167	1.475492
3	1/7/2008	1416.180054	84.769997	75.500000	15.053	1.468299
4	1/8/2008	1390.189941	86.779999	76.059998	15.590	1.557099

```
In [4]: # print last 5 rows of the dataframe
gold_data.tail()
```

```
Out[4]:
```

	Date	SPX	GLD	USO	SLV	EUR/USD
2285	5/8/2018	2671.919922	124.589996	14.0600	15.5100	1.186789
2286	5/9/2018	2697.790039	124.330002	14.3700	15.5300	1.184722
2287	5/10/2018	2723.070068	125.180000	14.4100	15.7400	1.191753
2288	5/14/2018	2730.129883	124.489998	14.3800	15.5600	1.193118
2289	5/16/2018	2725.780029	122.543800	14.4058	15.4542	1.182033

```
In [5]: # number of rows and columns
gold_data.shape
```

```
Out[5]: (2290, 6)
```

```
In [6]: # getting some basic informations about the data
gold_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2290 entries, 0 to 2289
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
--- 
 0   Date       2290 non-null   object  
 1   SPX        2290 non-null   float64 
 2   GLD        2290 non-null   float64 
 3   USO        2290 non-null   float64 
 4   SLV        2290 non-null   float64 
 5   EUR/USD    2290 non-null   float64 
dtypes: float64(5), object(1)
memory usage: 107.5+ KB
```

```
In [7]: # checking the number of missing values
gold_data.isnull().sum()
```

```
Out[7]: Date      0
SPX       0
GLD       0
USO       0
SLV       0
EUR/USD   0
dtype: int64
```

```
In [8]: # getting the statistical measures of the data
gold_data.describe()
```

```
Out[8]:          SPX      GLD      USO      SLV    EUR/USD
count  2290.000000  2290.000000  2290.000000  2290.000000  2290.000000
mean   1654.315776  122.732875  31.842221  20.084997  1.283653
std    519.111540  23.283346  19.523517  7.092566  0.131547
min    676.530029  70.000000  7.960000  8.850000  1.039047
25%   1239.874969  109.725000  14.380000  15.570000  1.171313
50%   1551.434998  120.580002  33.869999  17.268500  1.303297
75%   2073.010070  132.840004  37.827501  22.882500  1.369971
max   2872.870117  184.589996  117.480003  47.259998  1.598798
```

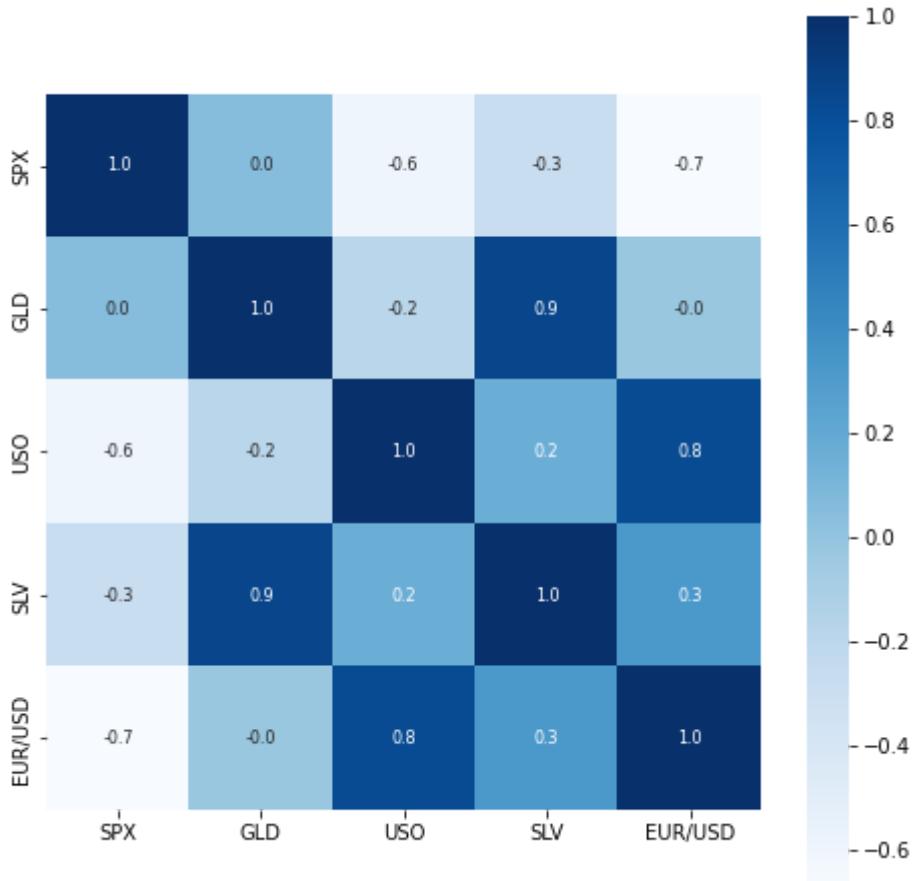
Correlation:

1. Positive Correlation
2. Negative Correlation

```
In [9]: correlation = gold_data.corr()
```

```
In [10]: # constructing a heatmap to understand the correlation
plt.figure(figsize = (8,8))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot=True, annot_kws={
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9593846100>
```



```
In [11]: # correlation values of GLD
print(correlation['GLD'])
```

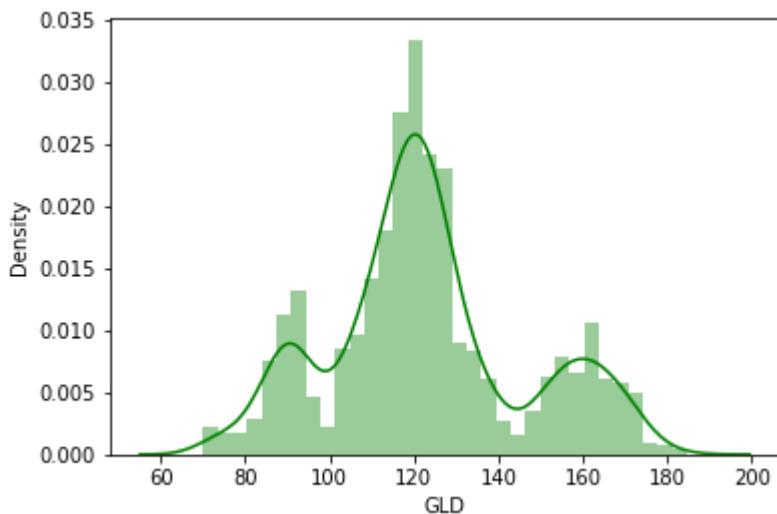
```
SPX      0.049345
GLD      1.000000
USO     -0.186360
SLV      0.866632
EUR/USD   -0.024375
Name: GLD, dtype: float64
```

```
In [12]: # checking the distribution of the GLD Price
sns.distplot(gold_data['GLD'],color='green')
```

/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

```
warnings.warn(msg, FutureWarning)
```

```
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f959388e700>
```



Splitting the Features and Target

```
In [13]: X = gold_data.drop(['Date', 'GLD'], axis=1)
Y = gold_data['GLD']
```

```
In [14]: print(X)
```

	SPX	USO	SLV	EUR/USD
0	1447.160034	78.470001	15.1800	1.471692
1	1447.160034	78.370003	15.2850	1.474491
2	1411.630005	77.309998	15.1670	1.475492
3	1416.180054	75.500000	15.0530	1.468299
4	1390.189941	76.059998	15.5900	1.557099
...
2285	2671.919922	14.060000	15.5100	1.186789
2286	2697.790039	14.370000	15.5300	1.184722
2287	2723.070068	14.410000	15.7400	1.191753
2288	2730.129883	14.380000	15.5600	1.193118
2289	2725.780029	14.405800	15.4542	1.182033

[2290 rows x 4 columns]

```
In [15]: print(Y)
```

0	84.860001
1	85.570000
2	85.129997
3	84.769997
4	86.779999
...	
2285	124.589996
2286	124.330002
2287	125.180000
2288	124.489998
2289	122.543800

Name: GLD, Length: 2290, dtype: float64

Splitting into Training data and Test Data

```
In [16]: X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_s
```

Model Training: Random Forest Regressor

```
In [17]: regressor = RandomForestRegressor(n_estimators=100)
```

```
In [18]: # training the model  
regressor.fit(X_train,Y_train)
```

```
Out[18]: RandomForestRegressor()
```

Model Evaluation

```
In [19]: # prediction on Test Data  
test_data_prediction = regressor.predict(X_test)
```

```
In [20]: print(test_data_prediction)
```

[168.8024997	81.76539994	116.12460019	127.62850073	120.48470156
154.66519818	150.24759836	125.84670103	117.63599858	126.1703004
116.85650075	172.02970087	141.59679814	167.85919873	115.10210022
117.82960026	138.57680361	169.86320092	159.40210318	161.05689992
154.91850014	125.44000014	175.49340004	157.50830401	125.1320003
93.66869953	78.02370047	120.52970031	119.13349971	167.45739924
88.04280098	125.24589994	91.14940071	117.66750035	121.2089994
135.98950078	115.694701	115.04590048	146.44709943	106.99790111
104.82810278	87.04569765	126.48040045	118.10090015	152.68889877
119.61880008	108.28900029	108.03599864	93.27370055	127.08999789
75.02340024	113.68889914	121.17329996	111.22789913	118.92419886
120.5121995	160.02059918	168.33770156	147.09079678	85.83389886
94.36280034	86.79519902	90.50050024	119.04540061	126.49490095
127.72149976	171.15230023	122.4069992	117.51719915	98.86540018
167.9912021	143.15119877	132.31180264	121.23400213	121.05899936
119.94710073	114.55480113	118.34530054	106.96280105	128.01270153
113.95429968	107.48039984	116.68990078	119.65489935	88.99480029
88.26269898	146.4706018	127.41390019	113.40880057	110.5933987
108.00199893	77.71559894	169.88760205	114.04949917	121.74349888
127.43100145	154.88539843	91.87309932	135.033001	159.02880316
125.05030046	125.63790047	130.52790141	114.99450071	119.80359947
92.04919968	110.24699897	167.74819956	156.49949951	114.1226994
106.6976014	79.26499986	113.33470035	125.84690105	107.2521991
119.56700101	155.6990033	159.7409986	119.96249984	134.27220299
101.39339988	117.32829809	119.38200035	112.91350054	102.7760988
160.50779769	99.36290041	147.62240009	125.3281011	169.61439948
125.59379886	127.37199737	127.57160148	113.86039913	113.08560077
123.54299896	102.1041991	88.95689982	124.7889997	101.74339927
107.01119913	113.6225001	117.55810039	99.19359956	121.67850059
163.340599	87.33899855	106.7294999	117.2687009	127.77440158
124.01900058	80.77579923	120.57860031	156.91729846	87.98909965
110.27619958	119.03359907	172.69829885	102.98369909	105.86200059
122.34990023	156.93189786	87.64069835	93.04940034	112.69390038
177.21080023	114.45490017	119.43210034	94.66300117	125.65990073
165.93120159	115.08220066	116.62860131	88.28939885	149.1347014
120.4637994	89.55709972	112.78680013	117.15080065	118.7462013
88.20489995	94.05830011	117.15969981	118.50690153	120.33750005
126.66489866	121.92029971	148.99210006	165.73610109	118.56489973
120.48780117	150.76999997	118.35839864	172.53419942	105.24789919
104.968201	149.43500155	114.01530066	124.81940116	147.6377997
119.51870136	115.36640055	112.68490007	113.46520213	140.84430107
117.9862976	102.99780036	115.891301	104.04320214	98.71730056
117.41470054	90.71890031	91.57620083	153.45039876	102.69129976
154.46740059	114.34300187	138.92410158	90.10079834	115.48689965
114.68779993	123.29140081	121.83640046	165.45400081	92.95649931
135.85150099	121.29709917	120.67990072	104.71530006	141.99780292
121.78439933	116.78350033	113.30420113	126.92689804	122.36689966
125.76879951	121.1925002	86.81459914	132.35280162	145.49390165
92.57609963	157.42719961	159.22610235	126.34419887	165.00649977
108.79369928	110.51460074	103.81319825	94.44030088	127.45920238
106.85330056	160.44280034	121.61150058	132.07840053	130.55650141
160.3821997	90.11689866	175.32800211	127.7070003	126.8019989
86.36999907	124.59159927	150.31859705	89.73030008	106.9375
109.01819982	84.51399903	135.98869973	154.97860217	138.48350371
73.99380007	151.70510101	126.09000092	126.82530024	127.55489919
108.71409963	156.1334008	114.77340088	116.90080167	125.46199968
153.99840094	121.46409979	156.44939912	92.90840041	125.55970153
126.02880037	87.78300066	92.25049951	126.39029925	128.60650335
113.18820028	117.55219758	120.7733002	127.02999853	119.59820125
135.96440047	93.80519931	119.75460029	113.22380113	94.25969941
108.92519934	87.20519881	108.94899918	89.62599972	92.5899002
131.4442032	162.36510047	89.35050028	119.52930064	133.46600222
123.85570026	128.31960197	102.02209867	88.87959882	131.31090094
119.73370041	108.6379002	167.83540179	115.12450037	86.6030987

```
118.81820093 91.06069994 162.07819985 116.63010042 121.49199974
160.18019772 120.12209912 112.79689913 108.42099855 126.74099978
76.11180024 102.97489994 127.08480207 121.75219942 92.58969992
131.91230033 118.03380151 116.19869968 154.46590275 160.02860093
110.24209944 156.64859782 119.32080091 160.58870059 118.61570049
158.23189948 115.24159956 116.4455005 149.64229906 114.91860058
125.655599 165.52139973 117.64900015 124.85989946 153.17650362
153.39210264 132.03060023 114.90310047 121.2380019 125.08720072
89.70290063 123.42919984 154.73330242 111.52600018 106.62440001
162.10980135 118.53599955 165.63710018 134.38040104 115.23589997
152.99509915 168.62709947 114.55610007 114.09480121 158.15769901
85.16059911 127.11650106 127.94040045 129.01919953 124.28710069
123.93290094 90.54930076 153.17020012 97.15369954 136.83959992
89.07429904 107.41179991 114.99410044 112.81100059 124.2679991
91.37529878 125.42780132 162.44889908 120.00079866 165.15160049
126.71939854 112.1334999 127.57619926 95.02019889 91.00459956
103.35769885 120.71540035 83.0787994 126.29580012 160.13240427
117.35450088 118.37429985 119.88980003 122.66529963 120.13190125
121.58870012 118.07590071 106.90810018 148.45260034 126.15509888
115.62890103 73.66800009 127.81950124 153.95840036 121.86300034
125.5668005 88.86170017 104.08609876 125.06970061 120.16880036
73.33340079 152.03700024 121.0907005 104.63079986 86.35239777
115.26989943 172.17669834 119.98030027 160.31919831 113.16560009
121.10420029 118.27880149 95.89909985 118.37110011 125.85890034
118.50799956 96.23690073 154.07480154 122.10479965 147.46699974
159.31840212 113.98780014 122.5353993 148.34009846 127.62480039
165.58640094 135.0120001 119.92969911 167.21709849 108.24659974
121.75059849 138.82410125 107.41269918]
```

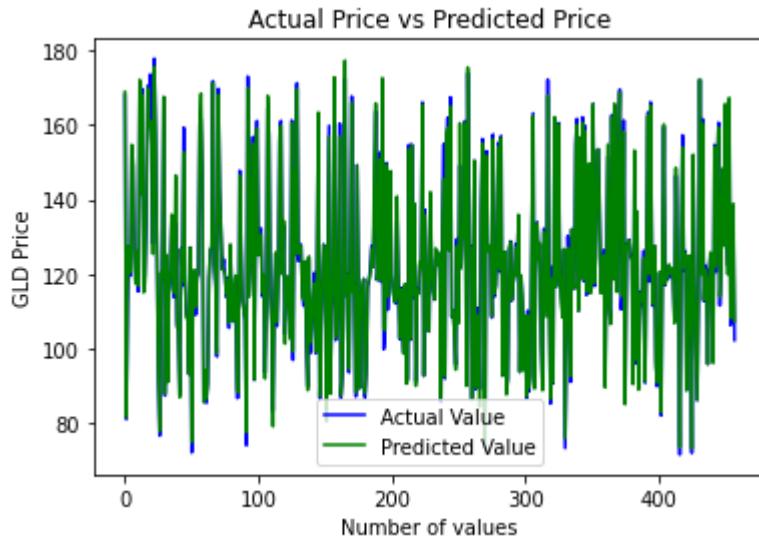
```
In [21]: # R squared error
error_score = metrics.r2_score(Y_test, test_data_prediction)
print("R squared error : ", error_score)
```

R squared error : 0.9897511997931183

Compare the Actual Values and Predicted Values in a Plot

```
In [22]: Y_test = list(Y_test)
```

```
In [23]: plt.plot(Y_test, color='blue', label = 'Actual Value')
plt.plot(test_data_prediction, color='green', label='Predicted Value')
plt.title('Actual Price vs Predicted Price')
plt.xlabel('Number of values')
plt.ylabel('GLD Price')
plt.legend()
plt.show()
```



```
In [24]: # Decision Tree

# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X_train,Y_train)
```

```
Out[24]: DecisionTreeRegressor(random_state=0)
```

```
In [31]: # predicting a new value

# test the output
y_pred = regressor.predict(X_test)
y_pred
```

```
Out[31]: array([168.970001,  86.089996, 114.769997, 127.550003, 121.730003,
 155.669998, 149.149994, 126.809998, 117.389999, 125.620003,
 117.959999, 174.580002, 141.919998, 167.179993, 115.839996,
 116.730003, 134.100006, 168.5      , 159.570007, 138.220001,
 155.360001, 126.610001, 177.210007, 157.339996, 125.459999,
 93.720001, 73.080002, 122.290001, 119.220001, 167.990005,
 87.370003, 124.769997, 91.730003, 117.919998, 121.300003,
 135.410004, 114.57      , 115.800003, 134.119995, 105.720001,
 104.370003, 87.239998, 127.489998, 118.360001, 157.779999,
 119.959999, 108.419998, 107.839996, 93.459999, 128.539993,
 72.510002, 113.260002, 120.110001, 109.860001, 118.919998,
 120.730003, 161.320007, 161.520004, 146.869995, 85.199997,
 93.040001, 86.879997, 90.949997, 119.800003, 126.139999,
 127.400002, 173.490005, 122.970001, 116.209999, 97.550003,
 170.130005, 142.050003, 132.490005, 120.910004, 122.879997,
 119.190002, 113.910004, 118.82      , 106.260002, 127.660004,
 114.769997, 108.470001, 115.57      , 119.699997, 89.910004,
 87.989998, 142.380005, 127.150002, 114.209999, 110.239998,
 108.279999, 73.080002, 173.020004, 113.639999, 121.209999,
 128.5      , 155.139999, 91.769997, 134.410004, 161.320007,
 124.43      , 126.18      , 130.369995, 113.910004, 121.160004,
 92.059998, 111.459999, 173.490005, 160.289993, 113.639999,
 107.040001, 81.989998, 113.669998, 125.540001, 108.089996,
 118.82      , 157.320007, 160.559998, 120.139999, 132.850006,
 101.849998, 116.029999, 118.959999, 112.440002, 102.309998,
 160.559998, 97.43      , 139.350006, 125.290001, 171.509995,
 123.730003, 128.539993, 128.380005, 115.199997, 112.139999,
 124.269997, 101.790001, 89.5      , 125.959999, 99.669998,
 107.040001, 115.849998, 116.730003, 97.699997, 121.650002,
 165.279999, 87.089996, 107.669998, 118.559998, 127.660004,
 125.18      , 79.790001, 121.470001, 160.619995, 89.5      ,
 109.760002, 118.279999, 174.399994, 102.459999, 104.860001,
 122.599998, 160.619995, 86.610001, 92.760002, 111.57      ,
 177.210007, 114.720001, 118.959999, 93.860001, 124.910004,
 166.630005, 113.910004, 117.260002, 88.419998, 150.410004,
 119.75      , 89.440002, 112.650002, 118.360001, 118.860001,
 87.690002, 94.599998, 114.459999, 118.230003, 120.839996,
 126.160004, 122.080002, 138.      , 165.649994, 119.32      ,
 120.730003, 151.440002, 120.559998, 174.399994, 107.43      ,
 105.169998, 150.410004, 114.769997, 124.360001, 146.589996,
 119.730003, 113.639999, 112.650002, 112.220001, 132.690002,
 118.099998, 103.150002, 115.940002, 104.370003, 98.360001,
 118.360001, 89.440002, 91.980003, 154.449997, 102.459999,
 154.470001, 115.050003, 137.660004, 90.589996, 115.139999,
 108.860001, 119.510002, 122.709999, 166.490005, 93.389999,
 137.660004, 121.110001, 120.160004, 105.68      , 135.020004,
 122.169998, 116.470001, 113.5      , 128.789993, 123.32      ,
 125.699997, 121.110001, 87.239998, 132.070007, 135.380005,
 91.5      , 161.520004, 161.320007, 126.949997, 168.110001,
 110.290001, 108.550003, 103.019997, 94.730003, 129.740005,
 107.040001, 161.520004, 121.650002, 131.740005, 130.369995,
 157.210007, 90.610001, 177.210007, 129.860001, 126.160004,
 84.769997, 124.019997, 150.479996, 89.440002, 107.669998,
 108.470001, 88.32      , 134.300003, 156.710007, 135.020004,
 72.5      , 151.440002, 126.809998, 127.169998, 127.970001,
 108.470001, 157.160004, 113.910004, 116.730003, 123.389999,
 154.      , 122.760002, 156.479996, 92.93      , 125.379997,
 124.239998, 88.580002, 91.610001, 125.620003, 129.710007,
 112.589996, 116.839996, 120.779999, 128.789993, 120.730003,
 136.050003, 92.949997, 118.809998, 112.940002, 94.599998,
 109.129997, 88.32      , 108.949997, 90.809998, 93.389999,
 130.369995, 161.419998, 88.32      , 120.139999, 134.25      ,
 124.389999, 129.710007, 101.419998, 91.309998, 130.800003,
 123.32      , 108.220001, 173.199997, 113.290001, 86.510002,
```

```
118.540001, 91.25 , 161.589996, 118.360001, 123.620003,
159.429993, 120.559998, 114.269997, 108.309998, 126.800003,
75.650002, 103.18 , 129.740005, 121.589996, 92.730003,
132.199997, 118.559998, 117.099998, 154.470001, 161.220001,
110.809998, 137.380005, 119.330002, 160.559998, 117.449997,
159.050003, 114.860001, 116.5 , 147.179993, 115.43 ,
127.779999, 167.339996, 117.139999, 124.779999, 152.300003,
153.050003, 132.009995, 115.43 , 121.559998, 129.740005,
90.279999, 125.32 , 157.320007, 110.459999, 107.669998,
162.300003, 118.769997, 165.800003, 134.75 , 113.269997,
152.589996, 169.610001, 117.769997, 114.720001, 161.449997,
84.459999, 127.120003, 128.270004, 128.110001, 124.529999,
124.400002, 90.800003, 151.910004, 96.5 , 136.050003,
86.449997, 108.470001, 115. , 111.75 , 124.959999,
91.610001, 124.589996, 161.419998, 121.239998, 167.580002,
126.160004, 112.970001, 127.599998, 95.449997, 94.349998,
102.279999, 120.739998, 84.279999, 126.589996, 160.460007,
116.610001, 118.68 , 117.610001, 124.389999, 120.720001,
120.940002, 117.650002, 107.040001, 146.869995, 123.730003,
115.32 , 74. , 127.349998, 157.779999, 118. , 125.540001,
89.269997, 102.199997, 123.389999, 121. , 73.790001,
151.050003, 120.110001, 103.93 , 88.949997,
114.769997, 172.289993, 121.470001, 160.649994, 113.669998,
121.790001, 118.559998, 96.5 , 119.169998, 126.07 ,
118.239998, 97.800003, 157.779999, 123.620003, 146. ,
159.300003, 114.730003, 123.239998, 145.729996, 128.669998,
166.130005, 132.690002, 120.980003, 173.020004, 109.199997,
122.019997, 135.410004, 103.419998])
```

```
In [33]: # R squared error
error_score = metrics.r2_score(Y_test, test_data_prediction)

print("R squared error : ", error_score)
```

```
R squared error : 0.9897511997931183
```

```
In [34]: regressor.score(X_test, Y_test)
```

```
Out[34]: 0.9854190298916252
```

```
In [ ]:
```

Practical 4

A. For a given set of training data examples stored in a .CSV file implement Least Square Regression algorithm.

In [1]:

```
# Import Library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
# read data
df=pd.read_csv('headbrain.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Gender	Age Range	Head Size(cm^3)	Brain Weight(grams)
0	1	1	4512	1530
1	1	1	3738	1297
2	1	1	4261	1335
3	1	1	3777	1282
4	1	1	4177	1590

In [4]:

```
# Declare dependent variable(Y) and independent variable(X)
X=df['Head Size(cm^3)'].values
Y = df['Brain Weight(grams)'].values
```

In [5]:

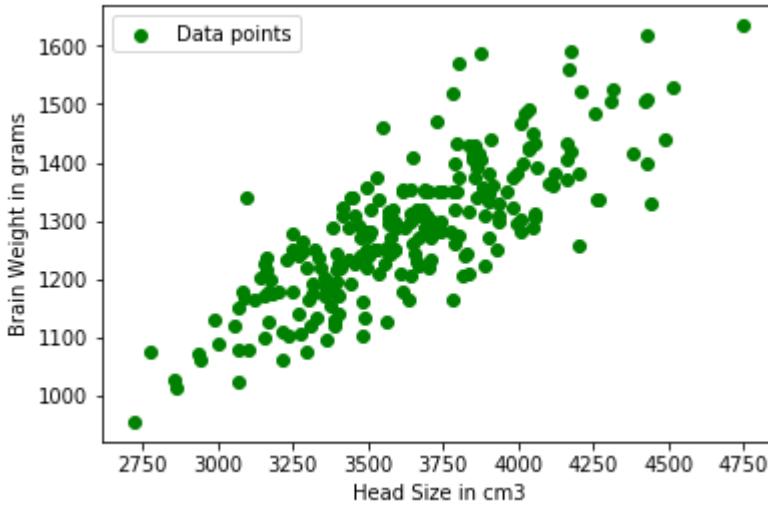
```
np.corrcoef(X, Y)
```

Out[5]:

```
array([[1.          , 0.79956971],
       [0.79956971, 1.        ]])
```

In [6]:

```
# Plot the Input Data
plt.scatter(X, Y, c='green', label='Data points')
plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()
```



```
In [7]: # Calculating coefficient

# Mean X and Y
mean_x = np.mean(X)
mean_y = np.mean(Y)

# Total number of values
n = len(X)

# Using the formula to calculate theta1 and theta2
numer = 0
denom = 0
for i in range(n):
    numer += (X[i] - mean_x) * (Y[i] - mean_y)
    denom += (X[i] - mean_x) ** 2
b1 = numer / denom
b0 = mean_y - (b1 * mean_x)

# Printing coefficients
print("coefficients for regression", b1, b0)
```

```
coefficients for regression 0.26342933948939945 325.57342104944223
```

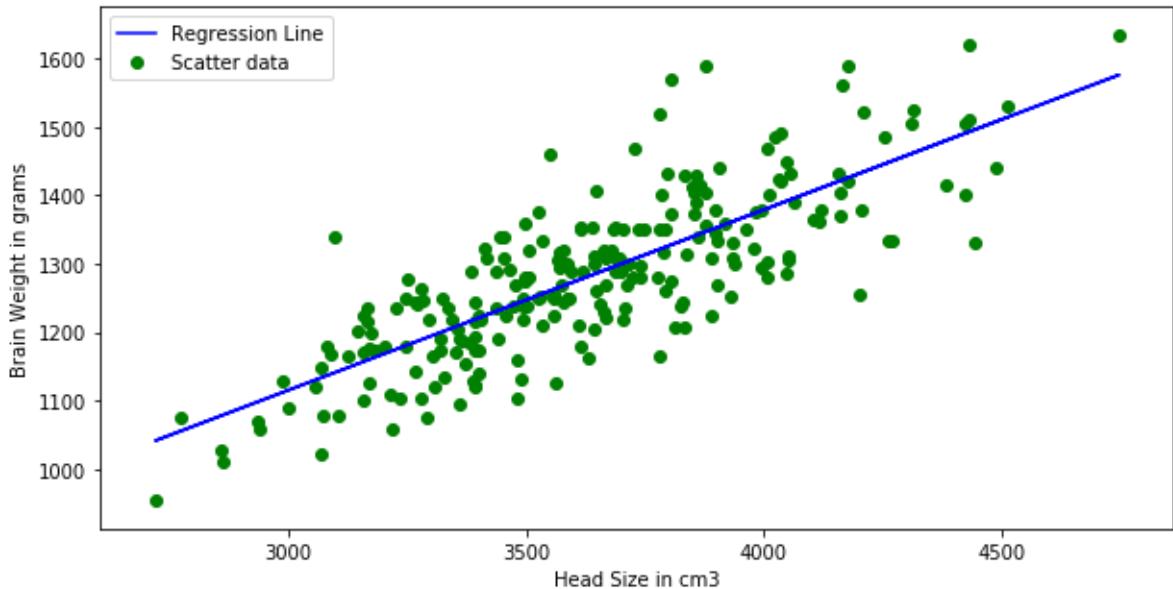
```
In [8]: # Plotting Values and Regression Line
%matplotlib inline

plt.rcParams['figure.figsize'] = (10.0, 5.0)
# max_x = np.max(X) + 100
# min_x = np.min(X) - 100

y = b0 + b1 * X

# Ploting Line
plt.plot(X, y, color='blue', label='Regression Line')
# Ploting Scatter Points
plt.scatter(X, Y, c='green', label='Scatter data')

plt.xlabel('Head Size in cm3')
plt.ylabel('Brain Weight in grams')
plt.legend()
plt.show()
```



```
In [9]: # Calculating Root Mean Squares Error
rmse = 0
for i in range(n):
    y_pred = b0 + b1 * X[i]
    rmse += (Y[i] - y_pred) ** 2

rmse = np.sqrt(rmse/n)
print("Root Mean Square Error is", rmse)
```

Root Mean Square Error is 72.1206213783709

```
In [10]: # Calculating R2 Score
ss_tot = 0
ss_res = 0
for i in range(n):
    y_pred = b0 + b1 * X[i]
    ss_tot += (Y[i] - mean_y) ** 2
    ss_res += (Y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ss_tot)
print("R2 Score", r2)
```

R2 Score 0.6393117199570003

```
In [ ]:
```

B. For a given set of training data examples stored in a .CSV file implement Logistic Regression algorithm.

```
In [12]: #import pandas
import pandas as pd
#col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'diabetes']
# Load dataset
pima = pd.read_csv("diabetes.csv")
```

```
In [13]: pima.head()
```

```
Out[13]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	A
0	6	148	72	35	0	33.6		0.627
1	1	85	66	29	0	26.6		0.351
2	8	183	64	0	0	23.3		0.672
3	1	89	66	23	94	28.1		0.167
4	0	137	40	35	168	43.1		2.288

```
In [15]: #split dataset in features and target variable
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age','Glucose','BloodPressure','SkinThickness', 'DiabetesPedigreeFunction', 'A']
X = pima[feature_cols] # Features
y = pima.Outcome # Target variable
```

```
In [17]: # split X and y into training and testing sets
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

Model Development and Prediction

First, import the Logistic Regression module and create a Logistic Regression classifier object using LogisticRegression() function.

Then, fit your model on the train set using fit() and perform prediction on the test set using predict().

```
In [23]: # import the class
from sklearn.linear_model import LogisticRegression
```

```
In [24]: # instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)
```

```
C:\Users\h103196\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:94
 0: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

    Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
    extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
Out[24]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                           intercept_scaling=1, l1_ratio=None, max_iter=100,
                           multi_class='auto', n_jobs=None, penalty='l2',
                           random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                           warm_start=False)
```

```
In [25]: # predict the model
y_pred=logreg.predict(X_test)
```

```
In [ ]:
```

Model Evaluation using Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. You can also visualize the performance of an algorithm.

The fundamental of a confusion matrix is the number of correct and incorrect predictions are summed up class-wise.

```
In [26]: # import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
Out[26]: array([[117,  13],
   [ 24,  38]], dtype=int64)
```

Here, you can see the confusion matrix in the form of the array object. The dimension of this matrix is 2*2 because this model is binary classification.

You have two classes 0 and 1. Diagonal values represent accurate predictions, while non-diagonal elements are inaccurate predictions.

In the output, 117 and 38 are actual predictions, and 24 and 13 are incorrect predictions.

Visualizing Confusion Matrix using Heatmap

Let's visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn.

```
In [27]: # import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

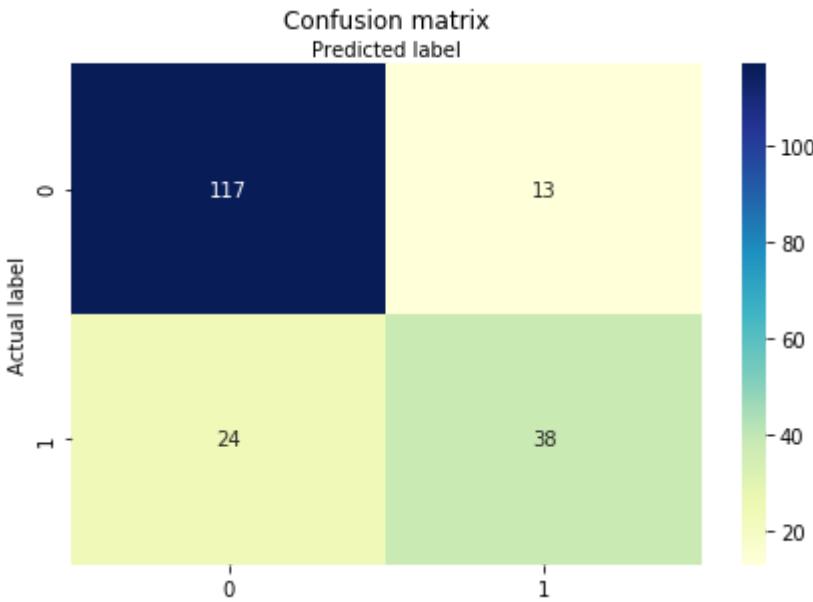
```
In [21]: class_names=[0,1] # name of classes
fig, ax = plt.subplots()
```

```

tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu", fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

```

Out[21]: Text(0.5, 257.44, 'Predicted label')



Confusion Matrix Evaluation Metrics

Let's evaluate the model using model evaluation metrics such as accuracy, precision, and recall.

In [22]:

```

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))

```

Accuracy: 0.8072916666666666
 Precision: 0.7450980392156863
 Recall: 0.6129032258064516

Well, you got a classification rate of 80%, considered as good accuracy.

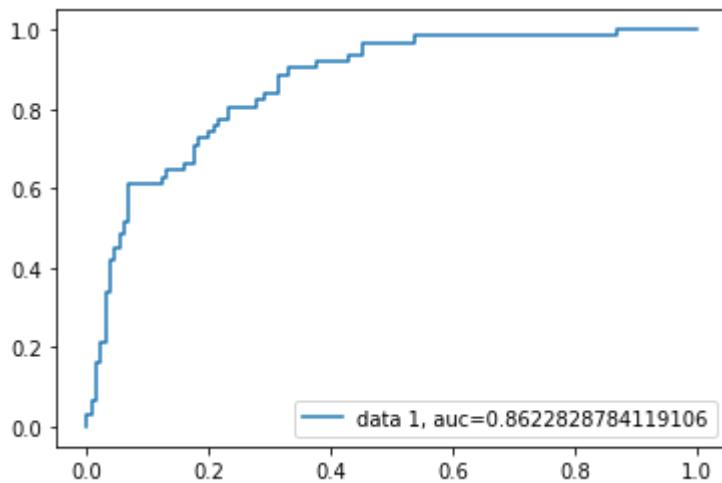
Precision: Precision is about being precise, i.e., how accurate your model is. In other words, you can say, when a model makes a prediction, how often it is correct. In your prediction case, when your Logistic Regression model predicted patients are going to suffer from diabetes, that patients have 74% of the time.

Recall: If there are patients who have diabetes in the test set and your Logistic Regression model can identify it 61% of the time.

ROC Curve

Receiver Operating Characteristic(ROC) curve is a plot of the true positive rate against the false positive rate. It shows the tradeoff between sensitivity and specificity.

```
In [28]: y_pred_proba = logreg.predict_proba(X_test)[:,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



AUC score for the case is 0.86. AUC score 1 represents perfect classifier, and 0.5 represents a worthless classifier.

```
In [ ]:
```

Practical 5

A. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

In [17]: #Importing important Libraries

```
import pandas as pd

from pandas import DataFrame

#Reading Dataset

df_tennis = pd.read_csv('/content/PlayTennis.csv')

print( df_tennis)
```

	Unnamed: 0	PlayTennis	Outlook	Temperature	Humidity	Wind
0	0	No	Sunny	Hot	High	Weak
1	1	No	Sunny	Hot	High	Strong
2	2	Yes	Overcast	Hot	High	Weak
3	3	Yes	Rain	Mild	High	Weak
4	4	Yes	Rain	Cool	Normal	Weak
5	5	No	Rain	Cool	Normal	Strong
6	6	Yes	Overcast	Cool	Normal	Strong
7	7	No	Sunny	Mild	High	Weak
8	8	Yes	Sunny	Cool	Normal	Weak
9	9	Yes	Rain	Mild	Normal	Weak
10	10	Yes	Sunny	Mild	Normal	Strong
11	11	Yes	Overcast	Mild	High	Strong
12	12	Yes	Overcast	Hot	Normal	Weak
13	13	No	Rain	Mild	High	Strong

In [18]: def entropy(probs):

```
    import math
```

```
    return sum( [-prob*math.log(prob, 2) for prob in probs] )
```

#Function to calculate the entropy of the given Data Sets/List with respect to target attribute

```
def entropy_of_list(a_list):
```

```
    #print("A-List",a_list)
```

```
    from collections import Counter
```

```
    cnt = Counter(x for x in a_list) # Counter calculates the proportion of class
    num_instances = len(a_list)*1.0 # = 14
```

```
    print("\n Number of Instances of the Current Sub Class is {0}: ".format(num_inst))
```

```
    probs = [x / num_instances for x in cnt.values()] # x means no of YES/NO
```

```
    print("\n Classes:",min(cnt),max(cnt))
```

```
    print(" \n Probabilities of Class {0} is {1}: ".format(min(cnt),min(probs)))
```

```
    print(" \n Probabilities of Class {0} is {1}: ".format(max(cnt),max(probs)))
```

```
    return entropy(probs) # Call Entropy :
```

The initial entropy of the YES/NO attribute for our dataset.

```
print("\n INPUT DATA SET FOR ENTROPY CALCULATION:\n", df_tennis['PlayTennis'])
```

```
total_entropy = entropy_of_list(df_tennis['PlayTennis'])
```

```
print("\n Total Entropy of PlayTennis Data Set:",total_entropy)
```

```

INPUT DATA SET FOR ENTROPY CALCULATION:
0      No
1      No
2      Yes
3      Yes
4      Yes
5      No
6      Yes
7      No
8      Yes
9      Yes
10     Yes
11     Yes
12     Yes
13     No
Name: PlayTennis, dtype: object

```

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Total Entropy of PlayTennis Data Set: 0.9402859586706309

```

In [19]: def information_gain(df, split_attribute_name, target_attribute_name, trace=0):
    print("Information Gain Calculation of ",split_attribute_name)
    ...
    Takes a DataFrame of attributes, and quantifies the entropy of a target
    attribute after performing a split along the values of another attribute.
    ...
    # Split Data by Possible Vals of Attribute:
    df_split = df.groupby(split_attribute_name)

    # Calculate Entropy for Target Attribute, as well as
    # Proportion of Obs in Each Data-Split
    nobs = len(df.index) * 1.0

    df_agg_ent = df_split.agg({target_attribute_name : [entropy_of_list, lambda x:
        df_agg_ent.columns = ['Entropy', 'PropObservations']
        #if trace: # helps understand what fxn is doing:
        #    print(df_agg_ent)

        # Calculate Information Gain:
        new_entropy = sum( df_agg_ent['Entropy'] * df_agg_ent['PropObservations'] )
        old_entropy = entropy_of_list(df[target_attribute_name])
        return old_entropy - new_entropy

    print('Info-gain for Outlook is :'+str( information_gain(df_tennis, 'Outlook', 'PlayTennis', 1) )
    print('\n Info-gain for Humidity is: ' + str( information_gain(df_tennis, 'Humidity', 'PlayTennis', 1) )
    print('\n Info-gain for Wind is:' + str( information_gain(df_tennis, 'Wind', 'PlayTennis', 1) )
    print('\n Info-gain for Temperature is:' + str( information_gain(df_tennis, 'Temperature', 'PlayTennis', 1) )

```

Information Gain Calculation of Outlook

Number of Instances of the Current Sub Class is 4.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 5.0:

Classes: No Yes

Probabilities of Class No is 0.4:

Probabilities of Class Yes is 0.6:

Number of Instances of the Current Sub Class is 5.0:

Classes: No Yes

Probabilities of Class No is 0.4:

Probabilities of Class Yes is 0.6:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Info-gain for Outlook is :0.2467498197744391

Information Gain Calculation of Humidity

Number of Instances of the Current Sub Class is 7.0:

Classes: No Yes

Probabilities of Class No is 0.42857142857142855:

Probabilities of Class Yes is 0.5714285714285714:

Number of Instances of the Current Sub Class is 7.0:

Classes: No Yes

Probabilities of Class No is 0.14285714285714285:

Probabilities of Class Yes is 0.8571428571428571:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Info-gain for Humidity is: 0.15183550136234136

Information Gain Calculation of Wind

Number of Instances of the Current Sub Class is 6.0:

Classes: No Yes

Probabilities of Class No is 0.5:

Probabilities of Class Yes is 0.5:

Number of Instances of the Current Sub Class is 8.0:

Classes: No Yes

Probabilities of Class No is 0.25:

Probabilities of Class Yes is 0.75:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Info-gain for Wind is:0.04812703040826927

Information Gain Calculation of Temperature

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.25:

Probabilities of Class Yes is 0.75:

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.5:

Probabilities of Class Yes is 0.5:

Number of Instances of the Current Sub Class is 6.0:

Classes: No Yes

Probabilities of Class No is 0.3333333333333333:

Probabilities of Class Yes is 0.6666666666666666:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Info-gain for Temperature is:0.029222565658954647

```
In [20]: def id3(df, target_attribute_name, attribute_names, default_class=None):

    ## Tally target attribute:
    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])# class of YES /NO

    ## First check: Is this split of the dataset homogeneous?
    if len(cnt) == 1:
        return next(iter(cnt)) # next input data set, or raises StopIteration when

    ## Second check: Is this split of the dataset empty?
    # if yes, return a default value
    elif df.empty or (not attribute_names):
        return default_class # Return None for Empty Data Set

    ## Otherwise: This dataset is ready to be devied up!
    else:
        # Get Default Value for next recursive call of this function:
        default_class = max(cnt.keys()) #No of YES and NO Class
        # Compute the Information Gain of the attributes:
        gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names]
        index_of_max = gainz.index(max(gainz)) # Index of Best Attribute
        # Choose Best Attribute to split on:
        best_attr = attribute_names[index_of_max]

        # Create an empty tree, to be populated in a moment
        tree = {best_attr:{}}
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]

        # Split dataset
        # On each split, recursively call this algorithm.
        # populate the empty tree with subtrees, which
        # are the result of the recursive call
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,
                          target_attribute_name,
                          remaining_attribute_names,
                          default_class)
            tree[best_attr][attr_val] = subtree
        return tree
```

Predicting from all Attributes given

```
In [21]: # Get Predictor Names (all but 'class')
attribute_names = list(df_tennis.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('PlayTennis') #Remove the class attribute
print("Predicting Attributes:", attribute_names)

List of Attributes: ['Unnamed: 0', 'PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind']
Predicting Attributes: ['Unnamed: 0', 'Outlook', 'Temperature', 'Humidity', 'Wind']
```

```
In [22]: # Run Algorithm:
from pprint import pprint
tree = id3(df_tennis,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
#print(tree)
pprint(tree)
attribute = next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree Keys:\n",tree[attribute].keys())
```


Information Gain Calculation of Unnamed: 0

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Information Gain Calculation of Outlook

Number of Instances of the Current Sub Class is 4.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 5.0:

Classes: No Yes

Probabilities of Class No is 0.4:

Probabilities of Class Yes is 0.6:

Number of Instances of the Current Sub Class is 5.0:

Classes: No Yes

Probabilities of Class No is 0.4:

Probabilities of Class Yes is 0.6:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Information Gain Calculation of Temperature

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.25:

Probabilities of Class Yes is 0.75:

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.5:

Probabilities of Class Yes is 0.5:

Number of Instances of the Current Sub Class is 6.0:

Classes: No Yes

Probabilities of Class No is 0.3333333333333333:

Probabilities of Class Yes is 0.6666666666666666:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Information Gain Calculation of Humidity

Number of Instances of the Current Sub Class is 7.0:

Classes: No Yes

Probabilities of Class No is 0.42857142857142855:

Probabilities of Class Yes is 0.5714285714285714:

Number of Instances of the Current Sub Class is 7.0:

Classes: No Yes

Probabilities of Class No is 0.14285714285714285:

Probabilities of Class Yes is 0.8571428571428571:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

Information Gain Calculation of Wind

Number of Instances of the Current Sub Class is 6.0:

Classes: No Yes

Probabilities of Class No is 0.5:

Probabilities of Class Yes is 0.5:

Number of Instances of the Current Sub Class is 8.0:

Classes: No Yes

Probabilities of Class No is 0.25:

Probabilities of Class Yes is 0.75:

Number of Instances of the Current Sub Class is 14.0:

Classes: No Yes

Probabilities of Class No is 0.35714285714285715:

Probabilities of Class Yes is 0.6428571428571429:

The Resultant Decision Tree is :

```
{'Unnamed: 0': {0: 'No',
  1: 'No',
  2: 'Yes',
  3: 'Yes',
  4: 'Yes',
  5: 'No',
  6: 'Yes',
  7: 'No',
  8: 'Yes',
  9: 'Yes',
  10: 'Yes',
  11: 'Yes',
  12: 'Yes',
  13: 'No'}}}
```

Best Attribute :

```
Unnamed: 0  
Tree Keys:  
dict_keys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13])
```

Check Accuracy for the same

Classification Accuracy for Training and Testing Set

```
In [23]: training_data = df_tennis.iloc[1:-4] # all but last four instances  
test_data = df_tennis.iloc[-4:] # just the last four  
train_tree = id3(training_data, 'PlayTennis', attribute_names)  
  
test_data['predicted2'] = test_data.apply(classify, # <---- test_data source  
                                         axis=1,  
                                         args=(train_tree,'Yes')) # <---- train_c  
  
print ('\n\n Accuracy is : ' + str( sum(test_data['PlayTennis']==test_data['predicted2']) / len(test_data) ) )
```

Information Gain Calculation of Unnamed: 0

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: No No

Probabilities of Class No is 1.0:

Probabilities of Class No is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 1.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 9.0:

Classes: No Yes

Probabilities of Class No is 0.3333333333333333:

Probabilities of Class Yes is 0.6666666666666666:
Information Gain Calculation of Outlook

Number of Instances of the Current Sub Class is 2.0:

Classes: Yes Yes

Probabilities of Class Yes is 1.0:

Probabilities of Class Yes is 1.0:

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.25:

Probabilities of Class Yes is 0.75:

Number of Instances of the Current Sub Class is 3.0:

Classes: No Yes

Probabilities of Class No is 0.3333333333333333:

Probabilities of Class Yes is 0.6666666666666666:
Information Gain Calculation of Temperature

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.25:

Probabilities of Class Yes is 0.75:

Number of Instances of the Current Sub Class is 2.0:

Classes: No Yes

Probabilities of Class No is 0.5:

Probabilities of Class Yes is 0.5:

Number of Instances of the Current Sub Class is 3.0:

Classes: No Yes

Probabilities of Class No is 0.333333333333333:

Probabilities of Class Yes is 0.666666666666666:

Number of Instances of the Current Sub Class is 9.0:

Classes: No Yes

Probabilities of Class No is 0.333333333333333:

Probabilities of Class Yes is 0.666666666666666:

Information Gain Calculation of Humidity

Number of Instances of the Current Sub Class is 4.0:

Classes: No Yes

Probabilities of Class No is 0.5:

Probabilities of Class Yes is 0.5:

Number of Instances of the Current Sub Class is 5.0:

Classes: No Yes

Probabilities of Class No is 0.2:

Probabilities of Class Yes is 0.8:

Number of Instances of the Current Sub Class is 9.0:

Classes: No Yes

Probabilities of Class No is 0.333333333333333:

Probabilities of Class Yes is 0.666666666666666:

Information Gain Calculation of Wind

Number of Instances of the Current Sub Class is 3.0:

Classes: No Yes

Probabilities of Class No is 0.333333333333333:

Probabilities of Class Yes is 0.666666666666666:

Number of Instances of the Current Sub Class is 6.0:

Classes: No Yes

Probabilities of Class No is 0.166666666666666:

Probabilities of Class Yes is 0.833333333333334:

Number of Instances of the Current Sub Class is 9.0:

```
Classes: No Yes
```

```
Probabilities of Class No is 0.3333333333333333:
```

```
Probabilities of Class Yes is 0.6666666666666666:  
Key: dict_keys(['Unnamed: 0'])  
Attribute: Unnamed: 0  
Key: dict_keys(['Unnamed: 0'])  
Attribute: Unnamed: 0  
Key: dict_keys(['Unnamed: 0'])  
Attribute: Unnamed: 0  
Key: dict_keys(['Unnamed: 0'])  
Attribute: Unnamed: 0
```

```
Accuracy is : 0.75
```

```
<ipython-input-23-8576c8cc24da>:5: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy  
test_data['predicted2'] = test_data.apply(classify, # <---- test_data source
```

```
The Accuracy is 0.75 ie, 75%
```

```
In [23]:
```

B. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set

```
In [38]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [39]: url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

# Assign column names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv(url, names=names)
```

```
In [40]: dataset.head()
```

```
Out[40]:
```

	sepal-length	sepal-width	petal-length	petal-width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [41]: dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   sepal-length    150 non-null    float64 
 1   sepal-width     150 non-null    float64 
 2   petal-length    150 non-null    float64 
 3   petal-width     150 non-null    float64 
 4   Class            150 non-null    object  
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
In [42]: dataset.describe()
```

```
Out[42]:
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
In [43]: dataset.isnull().sum()
```

```
Out[43]:
```

sepal-length	0
sepal-width	0
petal-length	0
petal-width	0
Class	0
dtype:	int64

```
In [44]: X = dataset.iloc[:, :-1].values  
y = dataset.iloc[:, 4].values
```

The X variable contains the first four columns of the dataset (i.e. attributes) while y contains the labels.

Train Test Split

```
In [45]: from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

Feature Scaling

```
In [46]: from sklearn.preprocessing import StandardScaler  
  
scaler = StandardScaler()  
scaler.fit(X_train)  
  
X_train = scaler.transform(X_train)  
X_test = scaler.transform(X_test)
```

```
In [47]: X_train
```

```
Out[47]: array([[-0.39871148,  1.08700609, -1.40772509, -1.30672051],  
[-0.28028233, -0.13664054,  0.17738091,  0.12708753],  
[-0.87242809,  1.82119407, -1.29450323, -1.17637432],  
[ 0.90400919, -0.38136987,  0.46043555,  0.12708753],  
[-1.10928639,  0.10808879, -1.29450323, -1.43706669],  
[-0.04342402, -0.87082852,  0.74349019,  0.90916464],  
[-1.583003 , -1.84974582, -1.40772509, -1.17637432],  
[-0.75399893,  2.55538205, -1.29450323, -1.43706669],  
[ 0.19343428, -0.13664054,  0.57365741,  0.77881846],  
[-1.70143215,  0.35281811, -1.40772509, -1.30672051],  
[-0.28028233, -0.13664054,  0.40382462,  0.3877799 ],  
[-0.99085724, -0.13664054, -1.2378923 , -1.30672051],  
[ 0.19343428, -2.09447515,  0.68687927,  0.3877799 ],  
[ 0.19343428, -0.38136987,  0.40382462,  0.3877799 ],  
[-0.04342402, -0.87082852,  0.17738091, -0.26395103],  
[-1.22771554,  0.10808879, -1.2378923 , -1.30672051],  
[ 1.25929665,  0.10808879,  0.74349019,  1.43054938],  
[-0.87242809,  1.82119407, -1.06805952, -1.04602814],  
[-0.39871148, -1.84974582,  0.12076998,  0.12708753],  
[ 1.49615495, -0.13664054,  1.19637762,  1.16985701],  
[ 1.02243834,  0.10808879,  0.51704648,  0.3877799 ],  
[-0.99085724,  1.08700609, -1.40772509, -1.17637432],  
[-0.99085724, -2.5839338 , -0.16228466, -0.26395103],  
[-0.75399893,  0.84227676, -1.35111416, -1.30672051],  
[ 1.6145841 ,  1.33173542,  1.30959948,  1.69124175],  
[-0.63556978,  1.57646474, -1.29450323, -1.30672051],  
[ 1.02243834,  0.59754744,  1.08315577,  1.69124175],  
[ 0.31186343, -0.38136987,  0.51704648,  0.25743372],  
[-1.22771554, -0.13664054, -1.35111416, -1.43706669],  
[ 2.44358817,  1.82119407,  1.47943226,  1.03951083],  
[-1.46457384,  0.84227676, -1.35111416, -1.17637432],  
[ 0.54872174, -0.87082852,  0.63026834,  0.77881846],  
[-1.10928639, -1.36028717,  0.40382462,  0.64847227],  
[-0.99085724,  1.08700609, -1.2378923 , -0.78533577],  
[ 1.73301326, -0.38136987,  1.42282134,  0.77881846],  
[-0.99085724,  0.84227676, -1.2378923 , -1.04602814],  
[-1.22771554,  0.84227676, -1.2378923 , -1.30672051],  
[-0.16185317, -1.11555785, -0.16228466, -0.26395103],  
[-0.51714063,  2.0659234 , -1.40772509, -1.04602814],  
[-0.87242809,  0.59754744, -1.18128137, -0.91568195],  
[-1.8198613 , -0.13664054, -1.52094695, -1.43706669],  
[ 0.54872174,  0.59754744,  1.25298855,  1.69124175],  
[ 1.02243834, -0.13664054,  0.80010112,  1.43054938],  
[-1.34614469,  0.35281811, -1.40772509, -1.30672051],  
[ 1.6145841 , -0.13664054,  1.13976669,  0.51812609],  
[ 0.43029259,  0.84227676,  0.91332298,  1.43054938],  
[ 1.02243834,  0.59754744,  1.08315577,  1.16985701],  
[-0.75399893, -0.87082852,  0.06415905,  0.25743372],  
[-0.39871148, -1.36028717,  0.12076998,  0.12708753],  
[ 0.78558004, -0.62609919,  0.46043555,  0.3877799 ],  
[-0.51714063, -0.13664054,  0.40382462,  0.3877799 ],  
[ 1.6145841 ,  0.35281811,  1.25298855,  0.77881846],  
[-0.51714063,  1.57646474, -1.29450323, -1.30672051],  
[ 1.85144241, -0.62609919,  1.30959948,  0.90916464],  
[ 0.31186343, -0.62609919,  0.12076998,  0.12708753],  
[-0.16185317, -1.36028717,  0.68687927,  1.03951083],  
[-0.16185317, -0.13664054,  0.23399184, -0.00325865],  
[-1.46457384,  0.35281811, -1.35111416, -1.30672051],  
[-1.10928639,  0.10808879, -1.29450323, -1.43706669],  
[ 1.02243834, -1.36028717,  1.13976669,  0.77881846],  
[ 1.25929665,  0.10808879,  0.91332298,  1.16985701],  
[ 0.78558004, -0.13664054,  0.96993391,  0.77881846],  
[ 0.31186343, -0.13664054,  0.46043555,  0.25743372],  
[-0.04342402, -0.87082852,  0.06415905, -0.00325865],
```

```
[ 0.66715089,  0.35281811,  0.85671205,  1.43054938],  
[-0.16185317, -0.62609919,  0.40382462,  0.12708753],  
[ 0.66715089, -0.87082852,  0.85671205,  0.90916464],  
[-0.39871148, -1.6050165 ,  0.00754812, -0.13360484],  
[ 0.43029259, -0.38136987,  0.29060277,  0.12708753],  
[-0.75399893,  1.08700609, -1.29450323, -1.30672051],  
[ 0.78558004, -0.13664054,  0.80010112,  1.03951083],  
[-0.99085724,  1.33173542, -1.35111416, -1.30672051],  
[ 0.54872174,  0.84227676,  1.02654484,  1.56089557],  
[ 1.1408675 , -0.62609919,  0.57365741,  0.25743372],  
[ 0.66715089, -0.38136987,  0.29060277,  0.12708753],  
[ 0.31186343, -1.11555785,  1.02654484,  0.25743372],  
[ 0.19343428,  0.84227676,  0.40382462,  0.51812609],  
[-0.87242809,  1.57646474, -1.29450323, -1.04602814],  
[ 2.20672986,  1.82119407,  1.64926505,  1.3002032 ],  
[ 0.66715089,  0.10808879,  0.96993391,  0.77881846],  
[-0.99085724, -1.84974582, -0.27550652, -0.26395103],  
[ 2.20672986, -1.11555785,  1.76248691,  1.43054938],  
[ 0.54872174, -1.36028717,  0.63026834,  0.3877799 ],  
[-0.87242809,  1.82119407, -1.2378923 , -1.30672051],  
[ 0.19343428, -0.87082852,  0.74349019,  0.51812609],  
[ 0.43029259, -0.62609919,  0.57365741,  0.77881846],  
[-0.99085724,  0.84227676, -1.29450323, -1.30672051],  
[-1.10928639, -0.13664054, -1.35111416, -1.30672051],  
[-0.28028233, -0.62609919,  0.63026834,  1.03951083],  
[ 1.3777258 ,  0.35281811,  0.51704648,  0.25743372],  
[-1.22771554,  0.84227676, -1.06805952, -1.30672051],  
[-1.10928639,  0.10808879, -1.29450323, -1.43706669],  
[-1.46457384,  0.10808879, -1.29450323, -1.30672051],  
[-1.70143215, -0.38136987, -1.35111416, -1.30672051],  
[ 0.66715089, -0.62609919,  1.02654484,  1.3002032 ],  
[-0.87242809,  0.84227676, -1.29450323, -1.30672051],  
[-0.39871148,  2.80011138, -1.35111416, -1.30672051],  
[-0.87242809, -1.36028717, -0.4453393 , -0.13360484],  
[ 0.54872174,  0.59754744,  0.51704648,  0.51812609],  
[ 0.54872174, -0.38136987,  1.02654484,  0.77881846],  
[ 0.54872174, -0.62609919,  0.74349019,  0.3877799 ],  
[-0.99085724,  0.59754744, -1.35111416, -1.30672051],  
[-0.28028233, -0.38136987, -0.10567373,  0.12708753],  
[ 0.90400919, -0.13664054,  0.34721369,  0.25743372],  
[-0.87242809,  1.08700609, -1.35111416, -1.30672051],  
[ 1.25929665,  0.35281811,  1.08315577,  1.43054938],  
[ 2.08830071, -0.13664054,  1.59265412,  1.16985701],  
[ 0.66715089, -0.62609919,  1.02654484,  1.16985701],  
[ 1.1408675 ,  0.35281811,  1.19637762,  1.43054938],  
[-1.34614469,  0.35281811, -1.2378923 , -1.30672051],  
[ 0.54872174, -1.84974582,  0.34721369,  0.12708753],  
[-0.04342402, -0.87082852,  0.74349019,  0.90916464],  
[ 0.66715089,  0.35281811,  0.40382462,  0.3877799 ],  
[-0.39871148, -1.11555785,  0.34721369, -0.00325865],  
[ 2.20672986, -0.13664054,  1.30959948,  1.43054938],  
[-0.04342402, -1.11555785,  0.12076998, -0.00325865],  
[-0.51714063,  0.84227676, -1.18128137, -1.30672051],  
[ 0.07500513, -0.13664054,  0.23399184,  0.3877799 ],  
[-0.16185317, -0.38136987,  0.23399184,  0.12708753],  
[ 0.07500513,  0.35281811,  0.57365741,  0.77881846]])
```

In [48]: X_test

```
Out[48]: array([[-2.80282326e-01, -8.70828519e-01,  2.33991838e-01,
                  1.27087531e-01],
                 [-1.70143215e+00, -1.36640541e-01, -1.40772509e+00,
                  -1.30672051e+00],
                 [-4.34240223e-02, -6.26099193e-01,  7.43490194e-01,
                  1.56089557e+00],
                 [ 1.02243834e+00,  1.08088786e-01,  3.47213695e-01,
                  2.57433716e-01],
                 [ 3.11863433e-01, -1.36640541e-01,  6.30268337e-01,
                  7.78818457e-01],
                 [-1.61853174e-01,  3.28957003e+00, -1.29450323e+00,
                  -1.04602814e+00],
                 [-1.61853174e-01,  1.82119407e+00, -1.18128137e+00,
                  -1.17637432e+00],
                 [ 1.25929665e+00,  1.08088786e-01,  6.30268337e-01,
                  3.87779901e-01],
                 [ 1.14086750e+00, -1.36640541e-01,  9.69933908e-01,
                  1.16985701e+00],
                 [-1.10928639e+00, -1.60501650e+00, -2.75506519e-01,
                  -2.63951025e-01],
                 [ 3.11863433e-01, -6.26099193e-01,  5.17046480e-01,
                  -3.25865463e-03],
                 [-5.17140630e-01,  2.06592340e+00, -1.18128137e+00,
                  -1.04602814e+00],
                 [-3.98711478e-01, -1.60501650e+00, -4.90628047e-02,
                  -2.63951025e-01],
                 [ 5.48721737e-01, -1.36028717e+00,  6.86879266e-01,
                  9.09164643e-01],
                 [-8.72428085e-01,  1.08700609e+00, -1.35111416e+00,
                  -1.17637432e+00],
                 [ 2.20672986e+00, -6.26099193e-01,  1.64926505e+00,
                  1.03951083e+00],
                 [-4.34240223e-02,  2.31065272e+00, -1.46433602e+00,
                  -1.30672051e+00],
                 [ 1.02243834e+00, -1.36640541e-01,  6.86879266e-01,
                  6.48472272e-01],
                 [ 7.85580041e-01, -1.36640541e-01,  1.13976669e+00,
                  1.30020320e+00],
                 [-5.17140630e-01,  8.42276765e-01, -1.29450323e+00,
                  -1.04602814e+00],
                 [ 1.02243834e+00,  1.08088786e-01,  1.02654484e+00,
                  1.56089557e+00],
                 [-1.61853174e-01, -6.26099193e-01,  1.77380909e-01,
                  1.27087531e-01],
                 [ 1.93434281e-01, -2.09447515e+00,  1.20769981e-01,
                  -2.63951025e-01],
                 [-9.90857237e-01,  3.52818112e-01, -1.46433602e+00,
                  -1.30672051e+00],
                 [ 4.30292585e-01, -2.09447515e+00,  4.03824623e-01,
                  3.87779901e-01],
                 [ 7.50051295e-02, -1.36640541e-01,  7.43490194e-01,
                  7.78818457e-01],
                 [-1.46457384e+00,  1.33173542e+00, -1.57755787e+00,
                  -1.30672051e+00],
                 [ 7.85580041e-01,  3.52818112e-01,  7.43490194e-01,
                  1.03951083e+00],
                 [-2.80282326e-01, -1.36028717e+00,  6.41590523e-02,
                  -1.33604840e-01],
                 [-1.22771554e+00, -1.36640541e-01, -1.35111416e+00,
                  -1.17637432e+00]])
```

Training and Predictions

```
In [49]: from sklearn.neighbors import KNeighborsClassifier  
  
classifier = KNeighborsClassifier(n_neighbors=5)  
classifier.fit(X_train, y_train)
```

```
Out[49]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                               metric_params=None, n_jobs=None, n_neighbors=5, p=2,  
                               weights='uniform')
```

The first step is to import the KNeighborsClassifier class from the sklearn.neighbors library.

In the second line, this class is initialized with one parameter, i.e. n_neighbours.

This is basically the value for the K.

There is no ideal value for K and it is selected after testing and evaluation, however to start out, 5 seems to be the most commonly used value for KNN algorithm.

The final step is to make predictions on our test data.

```
In [50]: y_pred = classifier.predict(X_test)
```

```
In [51]: y_pred
```

```
Out[51]: array(['Iris-versicolor', 'Iris-setosa', 'Iris-virginica',  
               'Iris-versicolor', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',  
               'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',  
               'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',  
               'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',  
               'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',  
               'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',  
               'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa',  
               'Iris-virginica', 'Iris-versicolor', 'Iris-setosa'], dtype=object)
```

Evaluating the Algorithm

```
In [52]: from sklearn.metrics import classification_report, confusion_matrix  
  
print(confusion_matrix(y_test, y_pred))  
print()  
print("*****")  
print()  
print(classification_report(y_test, y_pred))
```



```
[[10  0  0]  
 [ 0 10  1]  
 [ 0  1  8]]
```



```
*****
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	10
Iris-versicolor	0.91	0.91	0.91	11
Iris-virginica	0.89	0.89	0.89	9
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

Comparing Error Rate with the K Value

```
In [53]: error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
```

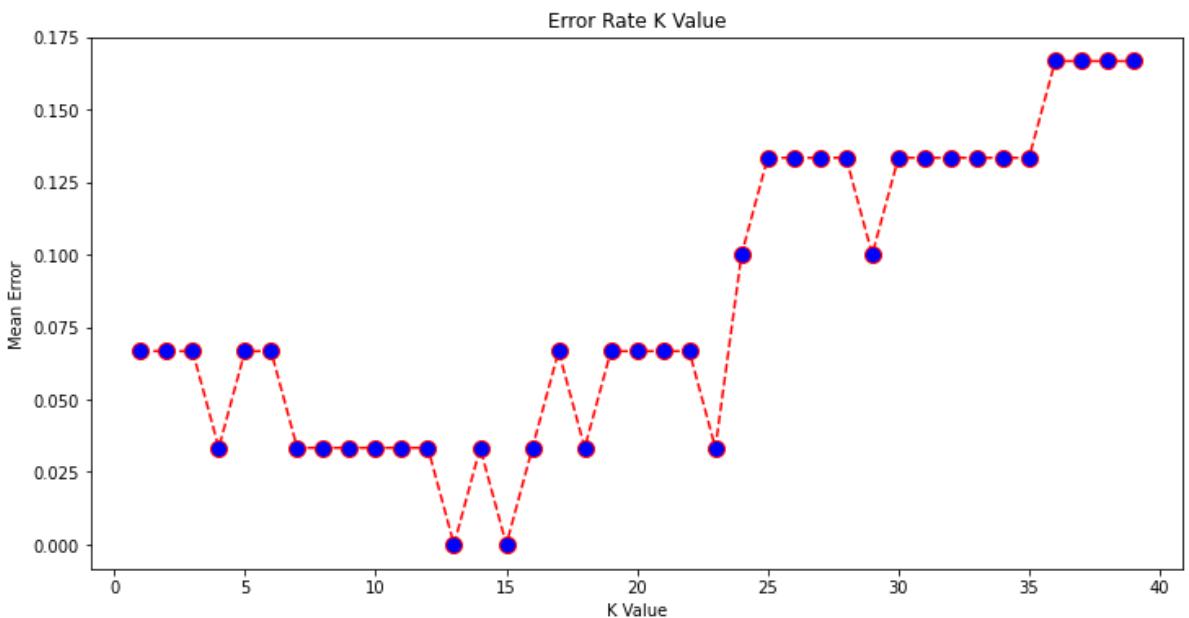
we will plot the mean error for the predicted values of test set for all the K values between 1 and 40.

The above script executes a loop from 1 to 40. In each iteration the mean error for predicted values of test set is calculated and the result is appended to the error list.

The next step is to plot the error values against K values. Execute the following script to create the plot:

```
In [54]: plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
Out[54]: Text(0, 0.5, 'Mean Error')
```



```
In [54]:
```

Practical 6

A. Implement the different Distance methods (Euclidean) with Prediction, Test Score and Confusion Matrix.

```
In [1]: # Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [2]: # Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

```
In [3]: # Splitting the dataset into the 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 = 0.20, random_
```

```
In [4]: # Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
In [5]: # Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

```
Out[5]: KNeighborsClassifier()
```

We are using 3 parameters in the model creation. n_neighbors is setting as 5, which means 5 neighborhood points are required for classifying a given point. The distance metric we are using is Minkowski, the equation for it is given below

$$\left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

As per the equation, we have to select the p-value also.

p = 1 , Manhattan Distance

p = 2 , Euclidean Distance

p = infinity , Chebychev Distance

In our problem, we are choosing the p as 2 (also u can choose the metric as "euclidean") Our Model is created, now we have to predict the output for the test set

```
In [6]: # Predicting the Test set results  
y_pred = classifier.predict(X_test)
```

```
In [10]: y_pred
```

```
In [7]: # Making the Confusion Matrix  
from sklearn.metrics import confusion_matrix, accuracy_score  
cm = confusion_matrix(y_test, y_pred)  
ac = accuracy_score(y_test, y_pred)
```

In [8]: cm

```
Out[8]: array([[55,  3],  
               [ 1, 21]])
```

In [9]: ac

Out[9]: 0.95

In []:

B. Implement the classification model using clustering for the following techniques with K means clustering with Prediction, Test Score and Confusion Matrix.

```
In [2]: import numpy as np # Linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [3]: # Importing the dataset
dataset = pd.read_csv('/content/Mall_Customers.csv')
```

```
In [4]: dataset.head()
```

```
Out[4]:
```

	CustomerID	Genre	Age	Annual_Income_(k\$)	Spending_Score
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

```
In [5]: dataset.tail()
```

```
Out[5]:
```

	CustomerID	Genre	Age	Annual_Income_(k\$)	Spending_Score
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

```
In [6]: dataset.shape
```

```
Out[6]: (200, 5)
```

```
In [7]: dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   CustomerID      200 non-null    int64  
 1   Genre            200 non-null    object  
 2   Age              200 non-null    int64  
 3   Annual_Income_(k$) 200 non-null    int64  
 4   Spending_Score   200 non-null    int64  
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
In [8]: dataset.describe()
```

```
Out[8]:
```

	CustomerID	Age	Annual_Income_(k\$)	Spending_Score
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

```
In [9]: dataset.isnull().sum()
```

```
Out[9]:
```

CustomerID	0
Genre	0
Age	0
Annual_Income_(k\$)	0
Spending_Score	0
dtype: int64	

```
In [10]: dataset.columns
```

```
Out[10]: Index(['CustomerID', 'Genre', 'Age', 'Annual_Income_(k$)', 'Spending_Score'], dtype='object')
```

No Nans found! Great

```
In [11]: data=dataset[['Annual_Income_(k$)', 'Spending_Score']]
```

```
In [12]: data
```

```
Out[12]:
```

	Annual_Income_(k\$)	Spending_Score
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
...
195	120	79
196	126	28
197	126	74
198	137	18
199	137	83

200 rows × 2 columns

```
In [13]:
```

```
data.shape
```

```
Out[13]:
```

```
(200, 2)
```

```
In [14]:
```

```
data.values
```

```
Out[14]: array([[ 15,  39],
   [ 15,  81],
   [ 16,   6],
   [ 16,  77],
   [ 17,  40],
   [ 17,  76],
   [ 18,   6],
   [ 18,  94],
   [ 19,   3],
   [ 19,  72],
   [ 19,  14],
   [ 19,  99],
   [ 20,  15],
   [ 20,  77],
   [ 20,  13],
   [ 20,  79],
   [ 21,  35],
   [ 21,  66],
   [ 23,  29],
   [ 23,  98],
   [ 24,  35],
   [ 24,  73],
   [ 25,   5],
   [ 25,  73],
   [ 28,  14],
   [ 28,  82],
   [ 28,  32],
   [ 28,  61],
   [ 29,  31],
   [ 29,  87],
   [ 30,   4],
   [ 30,  73],
   [ 33,   4],
   [ 33,  92],
   [ 33,  14],
   [ 33,  81],
   [ 34,  17],
   [ 34,  73],
   [ 37,  26],
   [ 37,  75],
   [ 38,  35],
   [ 38,  92],
   [ 39,  36],
   [ 39,  61],
   [ 39,  28],
   [ 39,  65],
   [ 40,  55],
   [ 40,  47],
   [ 40,  42],
   [ 40,  42],
   [ 42,  52],
   [ 42,  60],
   [ 43,  54],
   [ 43,  60],
   [ 43,  45],
   [ 43,  41],
   [ 44,  50],
   [ 44,  46],
   [ 46,  51],
   [ 46,  46],
   [ 46,  56],
   [ 46,  55],
   [ 47,  52],
   [ 47,  59],
```

[48, 51],
[48, 59],
[48, 50],
[48, 48],
[48, 59],
[48, 47],
[49, 55],
[49, 42],
[50, 49],
[50, 56],
[54, 47],
[54, 54],
[54, 53],
[54, 48],
[54, 52],
[54, 42],
[54, 51],
[54, 55],
[54, 41],
[54, 44],
[54, 57],
[54, 46],
[57, 58],
[57, 55],
[58, 60],
[58, 46],
[59, 55],
[59, 41],
[60, 49],
[60, 40],
[60, 42],
[60, 52],
[60, 47],
[60, 50],
[61, 42],
[61, 49],
[62, 41],
[62, 48],
[62, 59],
[62, 55],
[62, 56],
[62, 42],
[63, 50],
[63, 46],
[63, 43],
[63, 48],
[63, 52],
[63, 54],
[64, 42],
[64, 46],
[65, 48],
[65, 50],
[65, 43],
[65, 59],
[67, 43],
[67, 57],
[67, 56],
[67, 40],
[69, 58],
[69, 91],
[70, 29],
[70, 77],
[71, 35],
[71, 95],

[71, 11],
[71, 75],
[71, 9],
[71, 75],
[72, 34],
[72, 71],
[73, 5],
[73, 88],
[73, 7],
[73, 73],
[74, 10],
[74, 72],
[75, 5],
[75, 93],
[76, 40],
[76, 87],
[77, 12],
[77, 97],
[77, 36],
[77, 74],
[78, 22],
[78, 90],
[78, 17],
[78, 88],
[78, 20],
[78, 76],
[78, 16],
[78, 89],
[78, 1],
[78, 78],
[78, 1],
[78, 73],
[79, 35],
[79, 83],
[81, 5],
[81, 93],
[85, 26],
[85, 75],
[86, 20],
[86, 95],
[87, 27],
[87, 63],
[87, 13],
[87, 75],
[87, 10],
[87, 92],
[88, 13],
[88, 86],
[88, 15],
[88, 69],
[93, 14],
[93, 90],
[97, 32],
[97, 86],
[98, 15],
[98, 88],
[99, 39],
[99, 97],
[101, 24],
[101, 68],
[103, 17],
[103, 85],
[103, 23],
[103, 69],

```
[113,    8],
[113,   91],
[120,   16],
[120,   79],
[126,   28],
[126,   74],
[137,   18],
[137,  83]])
```

In [17]:

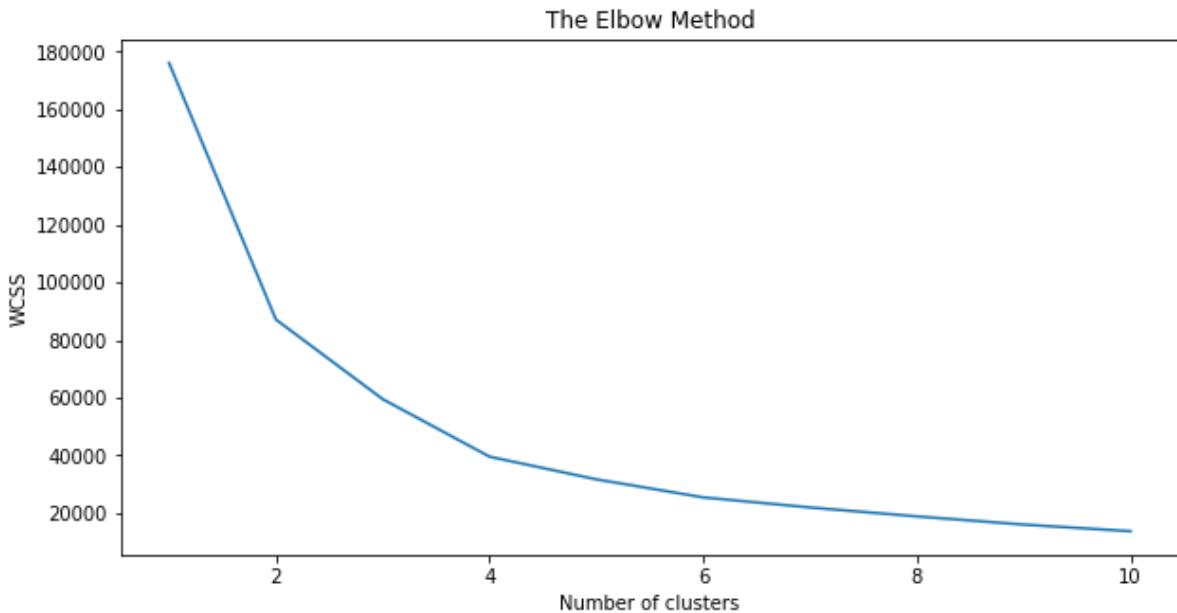
```
#we always assume the max number of cluster would be 10
#you can judge the number of clusters by doing averaging
###Static code to get max no of clusters
X = dataset.iloc[:, [2, 3]].values
```

In [18]:

```
# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wc_ss = []
for i in range(1, 11):
    kmeans_clu = KMeans(n_clusters = i, random_state = 56)
    kmeans_clu.fit(X)
    # inertia method returns wcss for that model
    wc_ss.append(kmeans_clu.inertia_)
```

In [19]:

```
plt.figure(figsize=(10,5))
plt.plot(range(1,11), wc_ss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



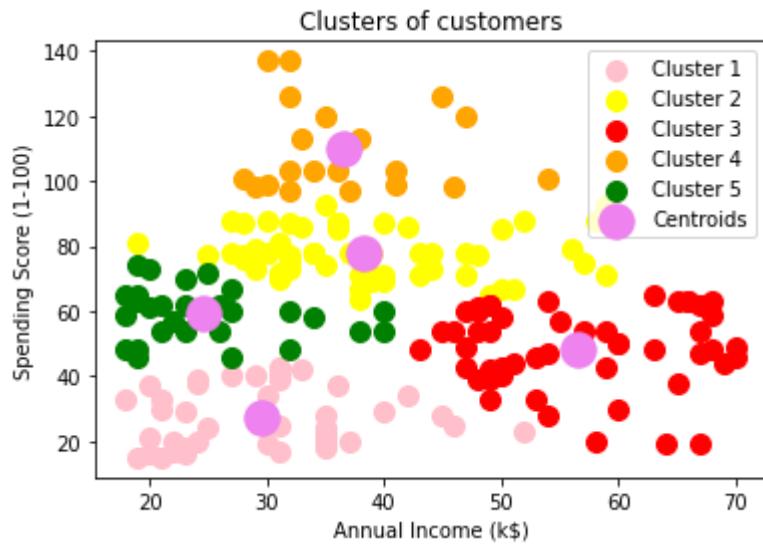
this curve is telling you that last elbow comes at k=5 no matter what range we select ex-(1,21) also i will see the same behaviour but if we chose higher range it is little difficult to visualize the ELBOW that is why we usually prefer range (1,11) Finally we got that k=5

In [20]:

```
# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, random_state = 56)
y_kmeans = kmeans.fit_predict(X)
```

7. Visualisation

```
In [21]: plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'pink', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'yellow', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'orange', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'green', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'purple', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



Practical 7

Implement the classification model using clustering for the following techniques with hierarchical clustering with Prediction, Test Score and Confusion Matrix

```
In [1]: import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd
```

```
In [3]: dataset = pd.read_csv('Mall_Customers.csv')
```

```
In [7]: dataset.head()
```

```
Out[7]:
```

	CustomerID	Genre	Age	Annual_Income_(k\$)	Spending_Score
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

Our goal is to cluster the customers based on their Spending score. That's why CustomerID and Genre are useless. So we remove both columns.

```
In [8]: X = dataset.iloc[:, [3, 4]].values
```

Now, we have only Annual Income and Spending Score Column.

```
In [9]: X
```

```
Out[9]: array([[ 15,  39],
   [ 15,  81],
   [ 16,   6],
   [ 16,  77],
   [ 17,  40],
   [ 17,  76],
   [ 18,   6],
   [ 18,  94],
   [ 19,   3],
   [ 19,  72],
   [ 19,  14],
   [ 19,  99],
   [ 20,  15],
   [ 20,  77],
   [ 20,  13],
   [ 20,  79],
   [ 21,  35],
   [ 21,  66],
   [ 23,  29],
   [ 23,  98],
   [ 24,  35],
   [ 24,  73],
   [ 25,   5],
   [ 25,  73],
   [ 28,  14],
   [ 28,  82],
   [ 28,  32],
   [ 28,  61],
   [ 29,  31],
   [ 29,  87],
   [ 30,   4],
   [ 30,  73],
   [ 33,   4],
   [ 33,  92],
   [ 33,  14],
   [ 33,  81],
   [ 34,  17],
   [ 34,  73],
   [ 37,  26],
   [ 37,  75],
   [ 38,  35],
   [ 38,  92],
   [ 39,  36],
   [ 39,  61],
   [ 39,  28],
   [ 39,  65],
   [ 40,  55],
   [ 40,  47],
   [ 40,  42],
   [ 40,  42],
   [ 42,  52],
   [ 42,  60],
   [ 43,  54],
   [ 43,  60],
   [ 43,  45],
   [ 43,  41],
   [ 44,  50],
   [ 44,  46],
   [ 46,  51],
   [ 46,  46],
   [ 46,  56],
   [ 46,  55],
   [ 47,  52],
   [ 47,  59],
```

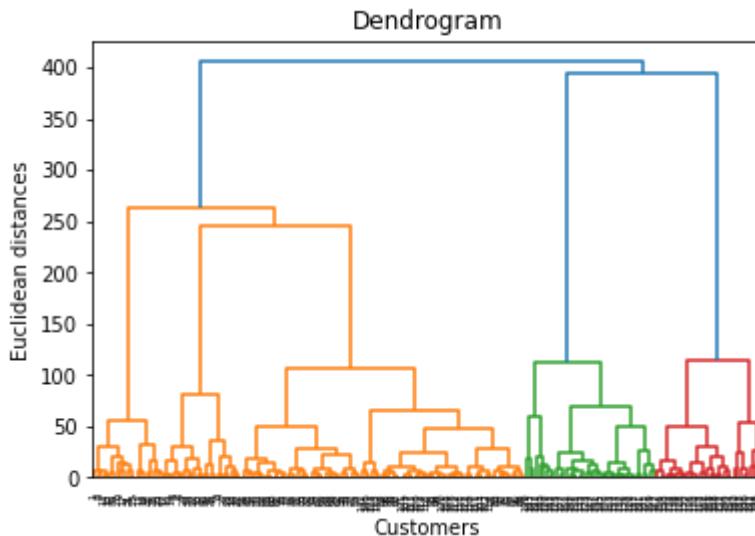
[48, 51],
[48, 59],
[48, 50],
[48, 48],
[48, 59],
[48, 47],
[49, 55],
[49, 42],
[50, 49],
[50, 56],
[54, 47],
[54, 54],
[54, 53],
[54, 48],
[54, 52],
[54, 42],
[54, 51],
[54, 55],
[54, 41],
[54, 44],
[54, 57],
[54, 46],
[57, 58],
[57, 55],
[58, 60],
[58, 46],
[59, 55],
[59, 41],
[60, 49],
[60, 40],
[60, 42],
[60, 52],
[60, 47],
[60, 50],
[61, 42],
[61, 49],
[62, 41],
[62, 48],
[62, 59],
[62, 55],
[62, 56],
[62, 42],
[63, 50],
[63, 46],
[63, 43],
[63, 48],
[63, 52],
[63, 54],
[64, 42],
[64, 46],
[65, 48],
[65, 50],
[65, 43],
[65, 59],
[67, 43],
[67, 57],
[67, 56],
[67, 40],
[69, 58],
[69, 91],
[70, 29],
[70, 77],
[71, 35],
[71, 95],

[71, 11],
[71, 75],
[71, 9],
[71, 75],
[72, 34],
[72, 71],
[73, 5],
[73, 88],
[73, 7],
[73, 73],
[74, 10],
[74, 72],
[75, 5],
[75, 93],
[76, 40],
[76, 87],
[77, 12],
[77, 97],
[77, 36],
[77, 74],
[78, 22],
[78, 90],
[78, 17],
[78, 88],
[78, 20],
[78, 76],
[78, 16],
[78, 89],
[78, 1],
[78, 78],
[78, 1],
[78, 73],
[79, 35],
[79, 83],
[81, 5],
[81, 93],
[85, 26],
[85, 75],
[86, 20],
[86, 95],
[87, 27],
[87, 63],
[87, 13],
[87, 75],
[87, 10],
[87, 92],
[88, 13],
[88, 86],
[88, 15],
[88, 69],
[93, 14],
[93, 90],
[97, 32],
[97, 86],
[98, 15],
[98, 88],
[99, 39],
[99, 97],
[101, 24],
[101, 68],
[103, 17],
[103, 85],
[103, 23],
[103, 69],

```
[113,    8],  
[113,   91],  
[120,   16],  
[120,   79],  
[126,   28],  
[126,   74],  
[137,   18],  
[137,   83]])
```

We have loaded dataset. Now its time to find the optimal number of clusters. And for that we need to create a Dendrogram.

```
In [10]: # Create Dendrogram to find the Optimal Number of Clusters  
  
import scipy.cluster.hierarchy as sch  
dendro = sch.dendrogram(sch.linkage(X, method = 'ward'))  
plt.title('Dendrogram')  
plt.xlabel('Customers')  
plt.ylabel('Euclidean distances')  
plt.show()
```



Here in the code "sch" is the short code for `scipy.cluster.hierarchy`.

"dendro" is the variable name. It may be anything. And "Dendrogram" is the function name.

So, after implementing this code, we will get our Dendrogram.

As I discussed that cut the horizontal line with longest line that traverses maximum distance up and down without intersecting the merging points.

In that dendrogram, the optimal number of clusters are 5.

Now let's fit our Agglomerative model with 5 clusters.

```
In [11]: # Fitting Agglomerative Hierarchical Clustering to the dataset
```

```
from sklearn.cluster import AgglomerativeClustering  
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')  
y_hc = hc.fit_predict(X)
```

Now our model has been trained. If you want to see different clusters, you can do it by simply writing print.

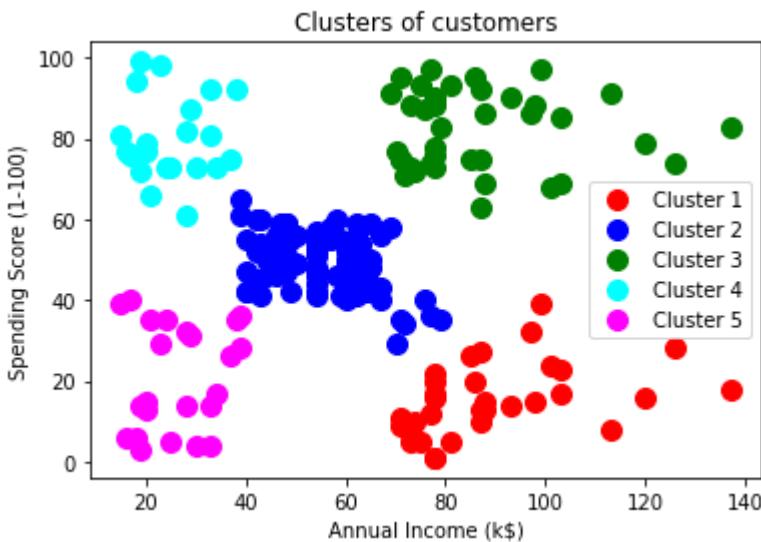
In [12]: `print(y_hc)`

```
[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 1 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 1 2  
0 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0  
2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```

Now, its time to visualize the clusters.

In [13]: `# Visualise the clusters`

```
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')  
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')  
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')  
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')  
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')  
plt.title('Clusters of customers')  
plt.xlabel('Annual Income (k$)')  
plt.ylabel('Spending Score (1-100)')  
plt.legend()  
plt.show()
```



In []:

Practical 8

A. Exploratory Data Analysis (EDA)

Objective to achieve, understand and implement following topics

- Handle Missing value
- Removing duplicates
- Outlier Treatment
- Normalizing and Scaling(Numerical Variables)
- Encoding Categorical variables(Dummy Variables)
- Bivariate Analysis

```
In [61]: # Importing all the necessary libraries
import numpy as np    # numerical python
import pandas as pd   # dataframe working
import matplotlib.pyplot as plt # plotting
import seaborn as sn  # advance plotting
%matplotlib inline
```

```
In [22]: df_car = pd.read_excel("/content/EDA_Cars.xlsx")
df_car.head(30)
```

Out[22]:

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED
0	1	125301	No	F	Bachelors	Blue Collar	45.703013	Commercial	17430.0
1	2	50815.4	No	M	High School		NaN	20.591628	Private
2	3	62977.8	NaN	F	Bachelors	Clerical	33.639949	Private	NaN
3	4	77100	No	F	NaN	Lawyer	15.415676		NaN
4	5	130795	No	M	High School		NaN	NaN	Commercial
5	6	NaN	NaN	F	High School	Home Maker	48.360191		NaN
6	7	87460.1	No	M	High School	Manager	45.000488		NaN
7	8	NaN	Yes	F	High School	Blue Collar	15.665947		NaN
8	9	@@	NaN	F	NaN	Clerical	26.392961		NaN
9	10	@@	Yes	M	High School	Blue Collar	27.490749		NaN
10	11	16988.7	No	M	High School	Blue Collar	20.450016		NaN
11	12	@@	No	F	High School	Blue Collar	61.603208	Commercial	9360.0
12	13	16352	Yes	F	NaN	Home Maker		NaN	Private
13	14	63952.2	Yes	F	Masters	Lawyer	49.576895	Private	32340.0
14	15	24059.7	No	M	Bachelors	Clerical	24.430198	Private	14220.0
15	16	20821.8	Yes	F	High School	Home Maker	45.907713	Private	6880.0
16	17	@@	No	F	High School	Blue Collar	42.500815	Private	NaN
17	18	137103	NaN	F	NaN	Manager	28.800357	Private	19510.0
18	19	102393	No	M	Bachelors	Professional	24.867970		NaN
19	20	40656.4	Yes	F	High School		NaN	57.091375	Private
20	21	@@	Yes	F	Bachelors	Blue Collar		NaN	Commercial
21	22	31773.1	Yes	M	High School	Clerical	14.459323	Commercial	9640.0
22	23	38035.6	Yes	F	High School	Blue Collar	32.180285	Commercial	13140.0
23	24	0	Yes	F	High School	Home Maker	24.520001	Private	11740.0
24	25	0	NaN	M	High School	Student	13.662860	Commercial	15090.0
25	26	NaN	Yes	F	Bachelors	Student	48.143336	Commercial	12850.0
26	27	44705.2	Yes	F	Bachelors	Professional	50.088976	Private	12840.0
27	28	31738.4	Yes	M	High School	Blue Collar	23.286841	Commercial	NaN

INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED
28	29	64013.8	Yes	M High School	Blue Collar	32.717234	Commercial	7900.0
29	30	53244.4	No	M Bachelors	Professional	27.551402	Nan	20920.0

In [23]: `df_car.shape`

Out[23]: `(303, 13)`

In [24]: `df_car.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   INDEX            303 non-null    int64  
 1   INCOME           265 non-null    object  
 2   MARITAL STATUS  275 non-null    object  
 3   SEX              297 non-null    object  
 4   EDUCATION        259 non-null    object  
 5   JOB              257 non-null    object  
 6   TRAVEL TIME     262 non-null    float64 
 7   USE              250 non-null    object  
 8   MILES CLOCKED   278 non-null    float64 
 9   CAR TYPE         293 non-null    object  
 10  CAR AGE          283 non-null    float64 
 11  CITY             297 non-null    object  
 12  POSTAL CODE     300 non-null    float64 
dtypes: float64(4), int64(1), object(8)
memory usage: 30.9+ KB
```

In [16]: `df_car.describe()`

	INDEX	TRAVEL TIME	MILES CLOCKED	CAR AGE	POSTAL CODE
count	303.000000	262.000000	278.000000	283.000000	300.000000
mean	139.640264	34.282098	13591.978417	6.265018	50712.196667
std	85.178422	14.910178	7167.328655	5.111218	24141.029290
min	1.000000	5.000000	1500.000000	1.000000	11435.000000
25%	62.500000	24.449874	7900.000000	1.000000	42420.000000
50%	138.000000	33.564757	12065.000000	6.000000	47150.000000
75%	213.500000	43.907339	18240.000000	10.000000	61701.000000
max	289.000000	83.617643	38000.000000	20.000000	90049.000000

In [25]: `df_car.describe(include=['object', 'int','float'])`

Out[25]:

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED
count	303.000000	265.0	275	297	259	257	262.000000	250	278.000000
unique	Nan	222.0	2	2	4	8	Nan	2	Nan
top	Nan	0.0	No	F	High School	Blue Collar	Nan	Private	Nan
freq	Nan	25.0	151	165	161	96	Nan	133	Nan
mean	139.640264	Nan	Nan	Nan	Nan	NaN	34.282098	Nan	13591.97841
std	85.178422	Nan	Nan	Nan	Nan	NaN	14.910178	Nan	7167.32865
min	1.000000	Nan	Nan	Nan	Nan	NaN	5.000000	Nan	1500.00000
25%	62.500000	Nan	Nan	Nan	Nan	NaN	24.449874	Nan	7900.00000
50%	138.000000	Nan	Nan	Nan	Nan	NaN	33.564757	Nan	12065.00000
75%	213.500000	Nan	Nan	Nan	Nan	NaN	43.907339	Nan	18240.00000
max	289.000000	Nan	Nan	Nan	Nan	NaN	83.617643	Nan	38000.00000

In [26]: df_car.INCOME

Out[26]:

```
0      125301
1      50815.4
2      62977.8
3      77100
4      130795
...
298    15251.5
299    18408.4
300    NaN
301    NaN
302    0
Name: INCOME, Length: 303, dtype: object
```

In [27]: df_car[df_car['INCOME'] == '@@']

Out[27]:

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	CATYI
8	9	@@	Nan	F	Nan	Clerical	26.392961	Nan	10030.0	SL
9	10	@@	Yes	M	High School	Blue Collar	27.490749	Nan	Nan	Pan Tru
11	12	@@	No	F	High School	Blue Collar	61.603208	Commercial	9360.0	Spor C
16	17	@@	No	F	High School	Blue Collar	42.500815	Private	Nan	SL
20	21	@@	Yes	F	Bachelors	Blue Collar	Nan	Commercial	5200.0	SL

In [19]:

In [29]: df_car['INCOME'] = df_car['INCOME'].replace(to_replace = '@@', value= np.nan)

```
df_car['INCOME'] = df_car['INCOME'].astype(float)
```

```
In [30]: df_car[df_car['INCOME'] == '@@']
```

```
Out[30]: INDEX INCOME MARITAL STATUS SEX EDUCATION JOB TRAVEL TIME USE MILES CLOCKED CAR TYPE CAR AGE CITY
```

```
In [31]: #df_car['TRAVEL TIME'] = df_car['TRAVEL TIME'].replace(to_replace = '*****', value
```

```
In [32]: df_car['TRAVEL TIME'] = df_car['TRAVEL TIME'].astype(float)
```

```
In [35]: #df_car['MILES CLOCKED'] = df_car['MILES CLOCKED'].replace(to_replace = 'Na', value  
#df_car['MILES CLOCKED'] = df_car['MILES CLOCKED'].replace(to_replace = '*****', va  
#df_car['MILES CLOCKED'] = df_car['MILES CLOCKED'].astype(float)
```

```
In [34]: #df_car.replace(to_replace='*****', value= np.nan, inplace = True)
```

```
In [33]: df_car.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 303 entries, 0 to 302  
Data columns (total 13 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  --  
 0   INDEX            303 non-null    int64    
 1   INCOME           260 non-null    float64  
 2   MARITAL STATUS  275 non-null    object    
 3   SEX              297 non-null    object    
 4   EDUCATION        259 non-null    object    
 5   JOB              257 non-null    object    
 6   TRAVEL TIME     262 non-null    float64  
 7   USE              250 non-null    object    
 8   MILES CLOCKED   278 non-null    float64  
 9   CAR TYPE         293 non-null    object    
 10  CAR AGE          283 non-null    float64  
 11  CITY             297 non-null    object    
 12  POSTAL CODE     300 non-null    float64  
dtypes: float64(5), int64(1), object(7)  
memory usage: 30.9+ KB
```

```
In [36]: # Check for missing values in any column or Handling missing values in dataset
```

```
df_car.isnull().sum()
```

```
Out[36]: INDEX          0  
INCOME          43  
MARITAL STATUS  28  
SEX             6  
EDUCATION       44  
JOB             46  
TRAVEL TIME    41  
USE             53  
MILES CLOCKED  25  
CAR TYPE        10  
CAR AGE         20  
CITY            6  
POSTAL CODE    3  
dtype: int64
```

We can see that we have various missing values in the respective columns. There are various ways of treating your missing values in the data set. And which technique to use when is actually dependent on the type of data you are dealing with.

- Drop the missing values: In this case, we drop the missing values from those variables. In case there are very few missing values you can drop those values.
- Impute with mean value: For the numerical column, you can replace the missing values with mean values. Before replacing with mean value, it is advisable to check that the variable shouldn't have extreme values .i.e. outliers.
- Impute with median value: For the numerical column, you can also replace the missing values with median values. In case you have extreme values such as outliers it is advisable to use the median approach.
- Impute with mode value: For the categorical column, you can replace the missing values with mode values i.e the frequent ones.

In [37]: *# In this exercise, we will replace the numerical columns with median values and for*

```
#Replacing the NULL Values in numerical columns using MEDIAN
median_income = df_car['INCOME'].median()
median_travel_time = df_car['TRAVEL TIME'].median()
median_miles_clocked = df_car['MILES CLOCKED'].median()
median_car_age = df_car['CAR AGE'].median()
median_postal_code = df_car['POSTAL CODE'].median()

df_car['INCOME'].replace(np.nan, median_income, inplace = True)
df_car['TRAVEL TIME'].replace(np.nan, median_travel_time, inplace = True)
df_car['MILES CLOCKED'].replace(np.nan, median_miles_clocked, inplace = True)
df_car['CAR AGE'].replace(np.nan, median_car_age, inplace = True)
df_car['POSTAL CODE'].replace(np.nan, median_postal_code, inplace = True)
```

In [38]: *# Replacing the NULL values in categorical columns using MODE*

```
mode_sex = df_car['SEX'].mode().values[0]
mode_marital_status = df_car['MARITAL STATUS'].mode().values[0]
mode_education = df_car['EDUCATION'].mode().values[0]
mode_job = df_car['JOB'].mode().values[0]
mode_use = df_car['USE'].mode().values[0]
mode_city = df_car['CITY'].mode().values[0]
mode_car_type = df_car['CAR TYPE'].mode().values[0]

df_car['SEX']= df_car['SEX'].replace(np.nan, mode_sex)
df_car['MARITAL STATUS']= df_car['MARITAL STATUS'].replace(np.nan, mode_marital_status)
df_car['EDUCATION']= df_car['EDUCATION'].replace(np.nan, mode_education)
df_car['JOB']= df_car['JOB'].replace(np.nan, mode_job)
df_car['USE']= df_car['USE'].replace(np.nan, mode_use)
df_car['CITY']= df_car['CITY'].replace(np.nan, mode_city)
df_car['CAR TYPE']= df_car['CAR TYPE'].replace(np.nan, mode_car_type)
```

```
In [39]: df_car.isnull().sum()
```

```
Out[39]: INDEX          0  
INCOME          0  
MARITAL STATUS  0  
SEX             0  
EDUCATION       0  
JOB             0  
TRAVEL TIME    0  
USE             0  
MILES CLOCKED  0  
CAR TYPE        0  
CAR AGE         0  
CITY            0  
POSTAL CODE     0  
dtype: int64
```

```
In [40]: # Handling Duplicate Records  
duplicate = df_car.duplicated()  
print(duplicate.sum())  
df_car[duplicate]
```

14

Out[40]:

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	1
69	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
70	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
71	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
72	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
73	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
74	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
75	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
76	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
77	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
78	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
79	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
80	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
81	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi
82	29	64013.81632	Yes	M	High School	Blue Collar	32.717234	Commercial	7900.0	Pi

Since we have 14 duplicate records in the data, we will remove this from the data set so that we get only distinct records.

Post removing the duplicate, we will check whether the duplicates have been removed from the data set or not.

In [41]: `# code to drop the duplicate
df_car.drop_duplicates(inplace=True)`

In [42]: `dup = df_car.duplicated()
dup.sum()`

Out[42]: 0

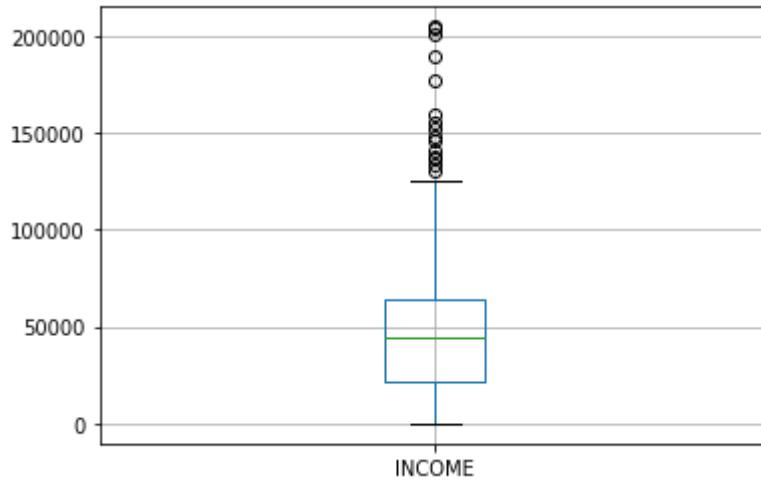
Handling Outlier

Outliers, being the most extreme observations, may include the sample maximum or sample minimum, or both, depending on whether they are extremely high or low.

However, the sample maximum and minimum are not always outliers because they may not be unusually far from other observations.

We generally identify outliers with the help of boxplot, so here box plot shows some of the data points outside the range of the data.

```
In [43]: df_car.boxplot(column=['INCOME'])  
plt.show()
```



Looking at the box plot, it seems that the variables INCOME have outliers present in the variables. These outliers value needs to be treated and there are several ways of treating them:

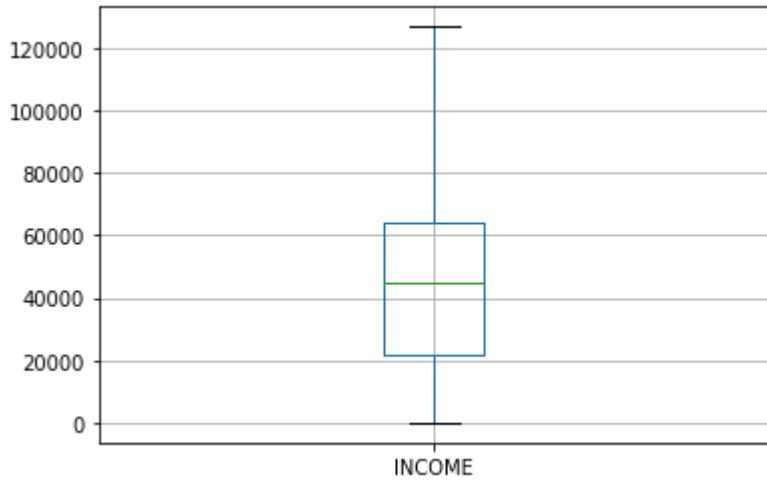
- Drop the outlier value
- Replace the outlier value using the IQR

```
In [44]: # creating a user defined function called remove_outlier for getting the threshold  
  
def remove_outlier(col):  
    sorted(col)  
    Q1, Q3 = col.quantile([0.25, 0.75])  
    IQR = Q3-Q1  
    lower_range = Q1-(1.5 * IQR)  
    upper_range = Q3+(1.5 * IQR)  
    return lower_range, upper_range
```

```
In [45]: lower_income, upper_income = remove_outlier(df_car['INCOME'])  
  
df_car['INCOME'] = np.where(df_car['INCOME'] > upper_income, upper_income, df_car['INCOME'])  
df_car['INCOME'] = np.where(df_car['INCOME'] < lower_income, lower_income, df_car['INCOME'])
```

After removing outliers, let us check it with boxplot

```
In [46]: df_car.boxplot(column=['INCOME'])  
plt.show()
```



Bivariate Analysis

When we talk about bivariate analysis, it means **analyzing 2 variables**. Since we know there are numerical and categorical variables, there is a way of analyzing these variables as shown below:

Numerical vs. Numerical

1. Scatterplot
2. Line plot
3. Heatmap for correlation
4. Joint plot

Categorical vs. Numerical

1. Bar chart
2. Violin plot
3. Categorical box plot
4. Swarm plot

Two Categorical Variables

1. Bar chart
2. Grouped bar chart
3. Point plot

```
In [47]: # if we need to find the correlation  
df_car.corr()
```

Out[47]:

	INDEX	INCOME	TRAVEL TIME	MILES CLOCKED	CAR AGE	POSTAL CODE
INDEX	1.000000	-0.044968	0.016710	0.042880	-0.027206	-0.244783
INCOME	-0.044968	1.000000	0.062594	0.342164	0.267087	0.034204
TRAVEL TIME	0.016710	0.062594	1.000000	0.026915	0.140511	0.021390
MILES CLOCKED	0.042880	0.342164	0.026915	1.000000	0.127137	-0.111283
CAR AGE	-0.027206	0.267087	0.140511	0.127137	1.000000	-0.099449
POSTAL CODE	-0.244783	0.034204	0.021390	-0.111283	-0.099449	1.000000

Normalizing and Scaling

Often the variables of the data set **are of different scales i.e. one variable is in millions and others in only 100.**

For e.g. in our data set Income is having values in thousands and age in just two digits. Since the data in these variables are of different scales, it is tough to compare these variables.

Feature scaling (also known as data normalization) is the method used to standardize the range of features of data.

Since the range of values of data may vary widely, it becomes a necessary step in data preprocessing while using machine learning algorithms.

In this method, we convert variables with different scales of measurements into a single scale.

StandardScaler normalizes the data using the formula $(x - \text{mean})/\text{standard deviation}$.

So we will be doing this only for the numerical variables.

In [48]: `# we use sklearn.preprocessing using the function StandardScaler`

```
from sklearn.preprocessing import StandardScaler
std_scale = StandardScaler()
std_scale
```

Out[48]: `StandardScaler(copy=True, with_mean=True, with_std=True)`

In [49]: `df_car['INCOME'] = std_scale.fit_transform(df_car[['INCOME']])
df_car['TRAVEL TIME'] = std_scale.fit_transform(df_car[['TRAVEL TIME']])
df_car['CAR AGE'] = std_scale.fit_transform(df_car[['CAR AGE']])
df_car['POSTAL CODE'] = std_scale.fit_transform(df_car[['POSTAL CODE']])
df_car['MILES CLOCKED'] = std_scale.fit_transform(df_car[['MILES CLOCKED']])`

In [50]: `df_car.head()`

Out[50]:

	INDEX	INCOME	MARITAL STATUS	SEX	EDUCATION	JOB	TRAVEL TIME	USE	MILES CLOCKED	CITY
0	1	2.330448	No	F	Bachelors	Blue Collar	0.807947	Commercial	0.534077	Sport C
1	2	0.120293	No	M	High School	Blue Collar	-0.964473	Private	0.750925	Minivan
2	3	0.481177	No	F	Bachelors	Clerical	-0.043492	Private	-0.241513	Suburb
3	4	0.900212	No	F	High School	Lawyer	-1.329803	Private	0.659849	Sport C
4	5	2.377524	No	M	High School	Blue Collar	-0.048799	Commercial	2.111280	Panama Tru

ENCODING

One-Hot-Encoding is used to create dummy variables to replace the categories in a categorical variable into features of each category and represent it using 1 or 0 based on the presence or absence of the categorical value in the record.

This is required to do since the machine learning algorithms only work on the numerical data.

That is why there is a need to convert the categorical column into a numerical one.

get_dummies is the method that creates a dummy variable for each categorical variable.

```
In [54]: dummies = pd.get_dummies(df_car[['MARITAL STATUS', 'SEX', 'EDUCATION', 'JOB', 'USE']]
                                columns = ['MARITAL STATUS', 'SEX', 'EDUCATION', 'JOB', 'USE']
                                prefix = ['married', 'sex', 'education', 'job', 'use', 'ca']
```

```
In [55]: dummies.head()
```

Out[55]:

	married_Yes	sex_M	education_High School	education_Masters	education_PhD	job_Clerical	job_Docto
0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0
2	0	0	0	0	0	0	1
3	0	0	1	0	0	0	0
4	0	1	1	0	0	0	0

```
In [59]: columns = ["MARITAL STATUS", "SEX", "EDUCATION", "JOB", "USE", "CAR TYPE", "CITY"]
df_car = pd.concat([df_car, dummies], axis=1)

# drop original column 'fuel type' from df

df_car.drop(columns, axis=1, inplace=True)
```

```
In [60]: df_car.head()
```

Out[60]:

	INDEX	INCOME	TRAVEL TIME	MILES CLOCKED	CAR AGE	POSTAL CODE	married_Yes	sex_M	education_Hig Scho
0	1	2.330448	0.807947	0.534077	0.137267	-0.277291	0.0	0.0	0
1	2	0.120293	-0.964473	0.750925	-1.052842	-0.277291	0.0	1.0	1
2	3	0.481177	-0.043492	-0.241513	-1.052842	-0.277291	0.0	0.0	0
3	4	0.900212	-1.329803	0.659849	0.930674	-0.277291	0.0	0.0	1
4	5	2.377524	-0.048799	2.111280	0.732322	-0.277291	0.0	1.0	1

5 rows × 90 columns

In []:

B.Time Series Analysis

Getting Google Trends data of keywords such as 'diet' and 'gym' and see how they vary over time while learning about trends and seasonality in time series data.

We will go through the code that we put together during the session step by step. You're not going to do much mathematics but you are going to do the following:

- Source your data
- Wrangle your data
- Exploratory Data Analysis
- Trends and seasonality in time series data *Identifying Trends* Seasonal patterns *First Order Differencing* Periodicity and Autocorrelation

In [1]:

```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
```

In [4]:

```
df = pd.read_csv('/content/multiTimeline.csv', skiprows=1)
df.head()
```

Out[4]:

	Month	diet: (Worldwide)	gym: (Worldwide)	finance: (Worldwide)
0	2004-01	100	31	48
1	2004-02	75	26	49
2	2004-03	67	24	47
3	2004-04	70	22	48
4	2004-05	72	22	43

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 168 entries, 0 to 167
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Month            168 non-null    object 
 1   diet: (Worldwide) 168 non-null    int64  
 2   gym: (Worldwide)  168 non-null    int64  
 3   finance: (Worldwide) 168 non-null  int64  
dtypes: int64(3), object(1)
memory usage: 5.4+ KB
```

Wrangle Your Data

The first thing that you want to do is rename the columns of your DataFrame df so that they have no whitespaces in them. There are multiple ways to do this, but for now, you'll reassign

to df.columns a list of what you want the columns to be called.

Double check the result of your reassignment by calling df.head():

```
In [6]: df.columns = ['month', 'diet', 'gym', 'finance']
df.head()
```

```
Out[6]:   month  diet  gym  finance
0  2004-01    100    31     48
1  2004-02     75    26     49
2  2004-03     67    24     47
3  2004-04     70    22     48
4  2004-05     72    22     43
```

Next, you'll turn the '**month**' column into a DateTime data type and make it the index of the DataFrame.

Note that you do this because you saw in the result of the `.info()` method that the '**Month**' **column** was actually an of data type object. Now, that generic data type encapsulates everything from strings to integers, etc. That's not exactly what you want when you want to be looking at time series data. That's why you'll use `.to_datetime()` to convert the '**month**' **column** in your DataFrame to a DateTime.

Be careful! Make sure to include the `inplace` argument when you're setting the index of the DataFrame `df` so that you actually alter the original index and set it to the 'month' column.

```
In [7]: df.month = pd.to_datetime(df.month)
df.set_index('month', inplace=True)
```

```
In [8]: df.head()
```

```
Out[8]:      diet  gym  finance
month
2004-01-01    100    31     48
2004-02-01     75    26     49
2004-03-01     67    24     47
2004-04-01     70    22     48
2004-05-01     72    22     43
```

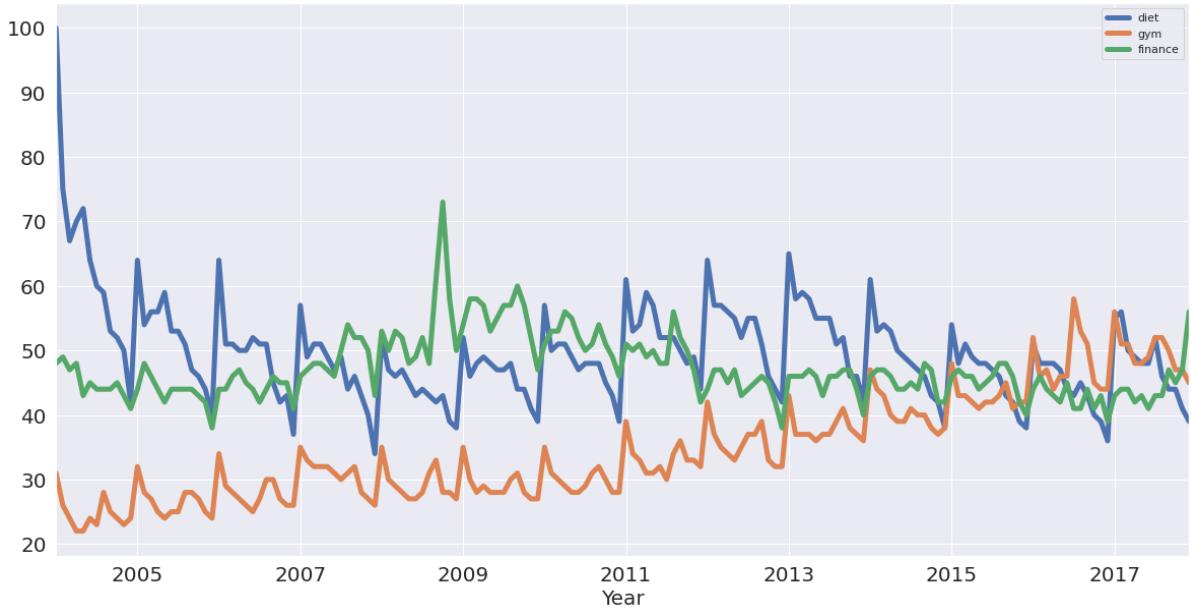
A bit of Exploratory Data Analysis (EDA)

You can use a built-in pandas visualization method `.plot()` to plot your data as 3 line plots on a single figure (one for each column, namely, '**diet**', '**gym**', and '**finance**').

Note that you can also specify some arguments to this method, such as `figsize`, `linewidth` and `fontsize` to set the figure size, line width and font size of the plot, respectively.

Additionally, you'll see that what you see on the x-axis is not the months, as the default label suggests, but the years. To make your plot a bit more accurate, you'll specify the label on the x-axis to 'Year' and also set the font size to 20.

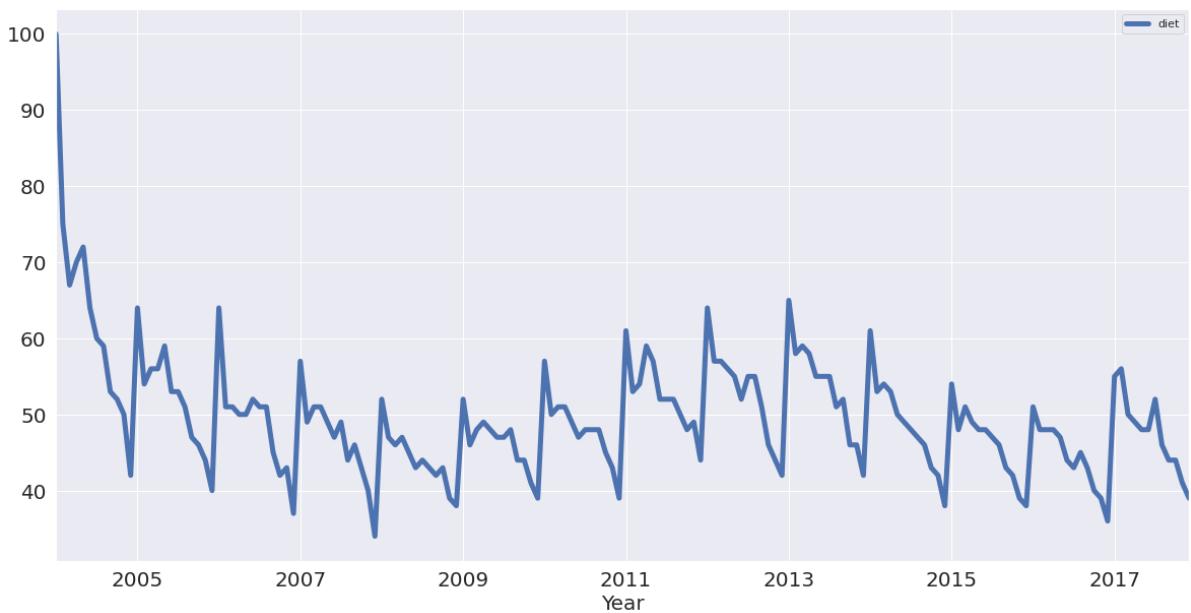
```
In [9]: df.plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20);
```



Numbers represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. Likewise a score of 0 means the term was less than 1% as popular as the peak.

If you want, you can also plot the 'diet' column by itself as a time series:

```
In [10]: df[['diet']].plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20);
```



Note: the first thing to notice is that there is seasonality: each January, there's a big jump. Also, there seems to be a trend: it seems to go slightly up, then down, back up and then

back down. In other words, it looks like there are trends and seasonal components to these time series.

With this in mind, you'll learn how to identify trends in your time series!

Identifying Trends in Time Series

There are several ways to think about identifying trends in time series. One popular way is by taking a rolling average, which means that, for each time point, you take the average of the points on either side of it. Note that the number of points is specified by a window size, which you need to choose.

What happens then because you take the average is it tends to smooth out noise and seasonality. You'll see an example of that right now. Check out this rolling average of 'diet' using the built-in pandas methods.

When it comes to determining the window size, here, it makes sense to first try out one of twelve months, as you're talking about yearly seasonality.

```
In [11]: diet = df[['diet']]
diet.rolling(12).mean().plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20);
```



Note that in the code chunk above you used two sets of squared brackets to extract the 'diet' column as a DataFrame;

If you would have used one set, like `df['diet']`, you would have created a pandas Series.

In the code chunk above, you also chained methods: you called methods on an object one after another. Method chaining is pretty popular and pandas is one of the packages that really allows you to use that style of programming to the max!

Now you have the trend that you're looking for! You have removed most of the seasonality compared to the previous plot.

You can also plot the rolling average of 'gym' using built-in pandas methods with the same window size as you took for the 'diet' data:

```
In [12]: gym = df[['gym']]
gym.rolling(12).mean().plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20);
```



You have successfully removed the seasonality and you see an upward trend for "gym"! But how do these two search terms compare?

You can figure this out by plotting the trends of 'gym' and 'diet' on a single figure:

```
In [13]: df_rm = pd.concat([diet.rolling(12).mean(), gym.rolling(12).mean()], axis=1)
df_rm.plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20);
```



You created a new DataFrame `df_rm` that has two columns with the rolling average of 'diet' and 'gym'. You used the `pd.concat()` function, which takes a list of the columns as a first argument and, since you want to concatenate them as columns, you also added the `axis` argument, which you set to 1.

Next, you plotted the DataFrame with the plot() method, just like you did before! So now, removing the seasonality, you see that diet potentially has some form of seasonality, whereas gym is actually increasing!

With the trends in the data identified, it's time to think about seasonality, which is the repetitive nature of your time series. As you saw in the beginning of this tutorial, it looked like there were trends and seasonal components to the time series of the data.

Seasonal Patterns in Time Series Data

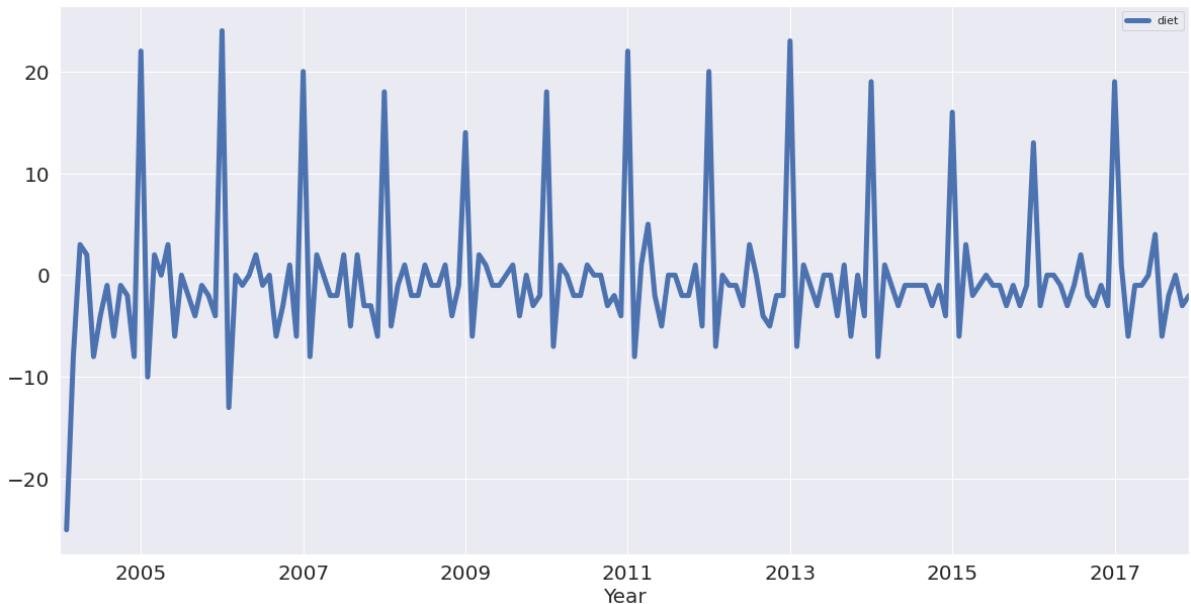
One way to think about the seasonal components to the time series of your data is to remove the trend from a time series, so that you can more easily investigate seasonality. To remove the trend, you can subtract the trend you computed above (rolling mean) from the original signal. This, however, will be dependent on how many data points you averaged over.

Another way to remove the trend is called "differencing", where you look at the difference between successive data points (called "first-order differencing", because you're only looking at the difference between one data point and the one before it).

First-order differencing

You can use pandas and the diff() and plot() methods to compute and plot the first order difference of the 'diet' Series:

```
In [14]: diet.diff().plot(figsize=(20,10), linewidth=5, fontsize=20)
plt.xlabel('Year', fontsize=20);
```



See that you have removed much of the trend and you can really see the peaks in January every year. Each January, there is a huge spike of 20 or more percent on the highest search item you've seen!

Periodicity and Autocorrelation

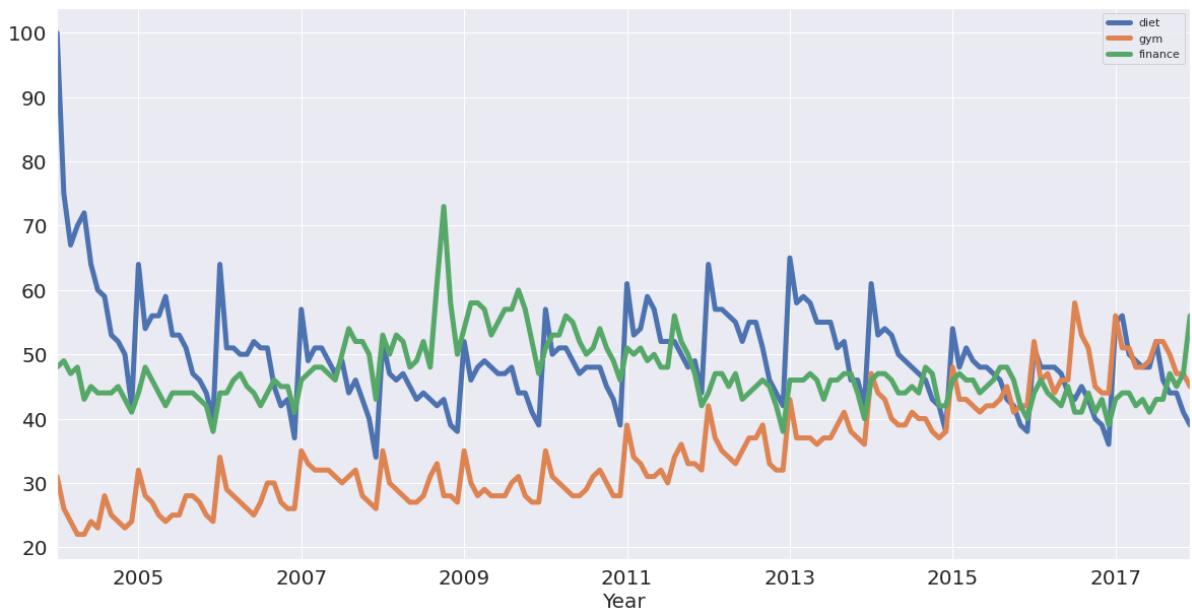
A time series is periodic if it repeats itself at equally spaced intervals, say, every 12 months.

Another way to think of this is that if the time series has a peak somewhere, then it will have a peak 12 months after that and, if it has a trough somewhere, it will also have a trough 12 months after that.

Yet another way of thinking about this is that the time series is correlated with itself shifted by 12 months. That means that, if you took the time series and moved it 12 months backwards or forwards, it would map onto itself in some way.

To start off, plot all your time series again to remind yourself of what they look like:

```
In [15]: df.plot(figsize=(20,10), linewidth=5, fontsize=20)  
plt.xlabel('Year', fontsize=20);
```



```
In [16]: df.corr()
```

```
Out[16]:
```

	diet	gym	finance
diet	1.000000	-0.100764	-0.034639
gym	-0.100764	1.000000	-0.284279
finance	-0.034639	-0.284279	1.000000

Now, what does this above tell you?

Let's focus on 'diet' and 'gym';

They are negatively correlated.

That's very interesting! Remember that you have a seasonal and a trend component.

From the correlation coefficient, 'diet' and 'gym' are negatively correlated.

However, from looking at the times series, it looks as though their seasonal components would be positively correlated and their trends negatively correlated.

The actual correlation coefficient is actually capturing both of those.

What you want to do now is plot the first-order differences of these time series and then compute the correlation of those because that will be the correlation of the seasonal components, approximately.

Remember that removing the trend may reveal correlation in seasonality.

Start off by plotting the first-order differences with the help of `.diff()` and `.plot()`:

```
In [17]: df.diff().plot(figsize=(20,10), linewidth=5, fontsize=20)  
plt.xlabel('Year', fontsize=20);
```



You see that 'diet' and 'gym' are incredibly correlated once you remove the trend. Now, you'll compute the correlation coefficients of the first-order differences of these time series:

```
In [18]: df.diff().corr()
```

```
Out[18]:
```

	diet	gym	finance
diet	1.000000	0.758707	0.373828
gym	0.758707	1.000000	0.301111
finance	0.373828	0.301111	1.000000

Note that once again, there was a slight negative correlation when you were thinking about the trend and the seasonal component. Now, you can see that with the seasonal component, 'diet' and 'gym' are highly correlated, with a coefficient of 0.76.

Autocorrelation

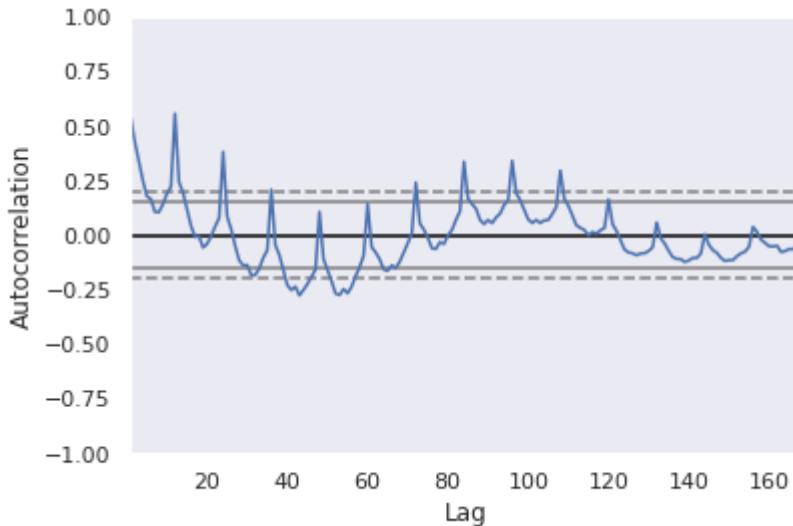
Now you've taken a dive into correlation of variables and correlation of time series, it's time to plot the autocorrelation of the 'diet' series: on the x-axis, you have the lag and on the y-axis, you have how correlated the time series is with itself at that lag.

So, this means that if the original time series repeats itself every two days, you would expect to see a spike in the autocorrelation function at 2 days.

Here, you'll look at the plot and what you should expect to see here is a spike in the autocorrelation function at 12 months: the time series is correlated with itself shifted by twelve months.

Use the plotting interface of pandas, which has the `autocorrelation_plot()` function. You can use this function to plot the time series 'diet':

In [19]: `pd.plotting.autocorrelation_plot(diet);`



If you included more lags in your axes, you'd see that it is 12 months at which you have this huge peak in correlation.

You have another peak at a 24 month interval, where it's also correlated with itself.

You have another peak at 36, but as you move further away, there's less and less of a correlation.

Of course, you have a correlation of itself with itself at a lag of 0.

The dotted lines in the above plot actually tell you about the statistical significance of the correlation.

In this case, you can say that the 'diet' series is genuinely autocorrelated with a lag of twelve months.

You have identified the seasonality of this 12 month repetition!

In []:

Practical 9

Aim: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.

Code

```
In [10]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

In [11]: msg=pd.read_csv('naivetext.csv',names=[ 'message','label'])

In [12]: print('The dimensions of the dataset',msg.shape)
The dimensions of the dataset (18, 2)

In [13]: msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum

In [14]: #splitting the dataset into train and test data
xtrain,xtest,ytrain,ytest=train_test_split(X,y)
print ('\n the total number of Training Data :',ytrain.shape)
print ('\n the total number of Test Data :',ytest.shape)

the total number of Training Data : (13,)

the total number of Test Data : (5,)

In [15]: #output the words or Tokens in the text documents
cv = CountVectorizer()
xtrain_dtm = cv.fit_transform(xtrain)
xtest_dtm=cv.transform(xtest)
print('\n The words or Tokens in the text documents \n')
print(cv.get_feature_names())
df=pd.DataFrame(xtrain_dtm.toarray(),columns=cv.get_feature_names())

The words or Tokens in the text documents

['about', 'am', 'amazing', 'an', 'and', 'awesome', 'bad', 'beers', 'boss', 'can',
'deal', 'enemy', 'feel', 'good', 'great', 'he', 'holiday', 'horrible', 'house', 'is',
'locality', 'love', 'my', 'of', 'place', 'sandwich', 'sick', 'stay', 'stuff',
'sworn', 'that', 'these', 'this', 'tired', 'to', 'today', 'very', 'view', 'went',
'what', 'with']

/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
warnings.warn(msg, category=FutureWarning)
```

```
In [16]: # Training Naive Bayes (NB) classifier on training data.  
clf = MultinomialNB().fit(xtrain_dtm,ytrain)  
predicted = clf.predict(xtest_dtm)
```

```
In [17]: #printing accuracy, Confusion matrix, Precision and Recall  
print('\n Accuracy of the classifier is',metrics.accuracy_score(ytest,predicted))  
print('\n Confusion matrix')  
print(metrics.confusion_matrix(ytest,predicted))  
print('\n The value of Precision', metrics.precision_score(ytest,predicted))  
print('\n The value of Recall', metrics.recall_score(ytest,predicted))
```

Accuracy of the classifier is 0.6

Confusion matrix
[[2 0]
 [2 1]]

The value of Precision 1.0

The value of Recall 0.3333333333333333

```
In [17]:
```

Practical 10

Aim: Perform Text pre-processing, Text clustering, classification with Prediction, Test Score and Confusion Matrix

```
In [7]: import nltk  
nltk.download('wordnet')  
nltk.download('omw-1.4')  
  
[nltk_data] Downloading package wordnet to /root/nltk_data...  
[nltk_data]   Package wordnet is already up-to-date!  
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...  
True  
Out[7]:
```

```
In [1]: import numpy as np  
import re  
import nltk  
from sklearn.datasets import load_files  
nltk.download('stopwords')  
import pickle  
from nltk.corpus import stopwords  
  
[nltk_data] Downloading package stopwords to /root/nltk_data...  
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
In [2]: movie_data = load_files(r"/content/sample_data/txt_sentoken")  
X, y = movie_data.data, movie_data.target
```

```
In [8]: documents = []  
  
from nltk.stem import WordNetLemmatizer  
  
stemmer = WordNetLemmatizer()  
  
for sen in range(0, len(X)):  
    # Remove all the special characters  
    document = re.sub(r'\W', ' ', str(X[sen]))  
  
    # remove all single characters  
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document)  
  
    # Remove single characters from the start  
    document = re.sub(r'^[a-zA-Z]\s+', ' ', document)  
  
    # Substituting multiple spaces with single space  
    document = re.sub(r'\s+', ' ', document, flags=re.I)  
  
    # Removing prefixed 'b'  
    document = re.sub(r'^b\s+', '', document)  
  
    # Converting to Lowercase  
    document = document.lower()  
  
    # Lemmatization  
    document = document.split()
```

```
document = [stemmer.lemmatize(word) for word in document]
document = ' '.join(document)

documents.append(document)
```

```
In [9]: from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features=1500, min_df=5, max_df=0.7, stop_words=stopwords)
X = vectorizer.fit_transform(documents).toarray()
```

```
In [12]: from sklearn.feature_extraction.text import TfidfTransformer
tfidfconverter = TfidfTransformer()
X = tfidfconverter.fit_transform(X).toarray()
```

```
In [13]: from sklearn.feature_extraction.text import TfidfVectorizer
tfidfconverter = TfidfVectorizer(max_features=1500, min_df=5, max_df=0.7, stop_words=stopwords)
X = tfidfconverter.fit_transform(documents).toarray()
```

```
In [14]: 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=0)
```

```
In [16]: from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=1000, random_state=0)
classifier.fit(X_train, y_train)
```

```
Out[16]: RandomForestClassifier(n_estimators=1000, random_state=0)
```

```
In [17]: y_pred = classifier.predict(X_test)
```

```
In [18]: from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

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

[[197 0]		
[76 9]]		
	precision recall f1-score support	
1	0.72 1.00 0.84 197	
2	1.00 0.11 0.19 85	
accuracy		0.73 282
macro avg	0.86 0.55 0.51 282	
weighted avg	0.81 0.73 0.64 282	

```
0.7304964539007093
```

```
In [ ]:
```