Bridge the Gap

Study of Python Programming Language.

About Python Programming Language:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. There are some features of Python Language:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Applications of Python:

1. **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
2. **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
3. **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
4. **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
5. **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
6. **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
7. **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
8. **Databases** − Python provides interfaces to all major commercial databases.
9. **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
10. **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Standard Data Types:

- Numbers

- String
- List
- Tuple
- Dictionary

Python Identifiers:

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Conclusion: Thus we have studied basic methods in Python Programming Language.

Basic modules of Python:

```
print "Hello, Python!"
```

```
Hello, Python!
```

```
a = 33
b = 200
if b > a:
print("b is greater than a")
```

```
b is greater than a
```

```
x = 1
y = 35656222554887711
z = -3255522
print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

```
['apple', 'banana', 'cherry']
```

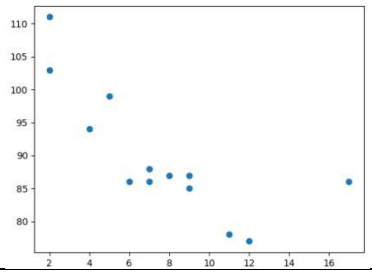
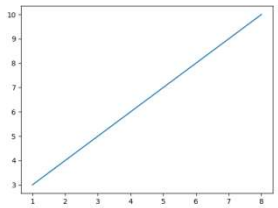
```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

```
3
```

```
import datetime
x = datetime.datetime.now()
print(x)
```

```
2021-11-26 10:31:46.841742
```

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

	John 36
import matplotlib.pyplot as plt import numpy as np x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6]) y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86]) plt.scatter(x, y) plt.show()  	import matplotlib.pyplot as plt import numpy as np xpoints = np.array([1, 8]) ypoints = np.array([3, 10]) plt.plot(xpoints, ypoints) plt.show() 

ASSIGNMENT 1

Title: **Data Preparation**

Aim: Download heart dataset from following link. https://www.kaggle.com/zhaoyingzhu/heartcsv

Perform following operation on given dataset.
a) Find Shape of Data
b) Find Missing Values
c) Find data type of each column
d) Finding out Zero's
e) Find Mean age of patients
f) Now extract only Age, Sex, ChestPain, RestBP, Chol. Randomly divide dataset in training (75%) and testing (25%).

Through the diagnosis test I predicted 100 report as COVID positive, but only 45 of those were actually positive. Total 50 people in my sample were actually COVID positive. I have total 500 samples.
Create confusion matrix based on above data and find

I Accuracy
II Precision
III Recall
IV F-1 score

Software Requirement: Jupyter

Theory:

Data Preparation: It is the process of transforming raw data into a particular form so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions. All projects have the same general steps; they are:

Step 1: Define Problem.
Step 2: Prepare Data.
Step 3: Evaluate Models.
Step 4: Finalize Model.

We are concerned with the data preparation step (step 2), and there are common or standard tasks that you may use or explore during the data preparation step in a machine learning project.

Data Preparation Tasks

1. Data Cleaning: There are many reasons data may have incorrect values, such as being mistyped, corrupted, duplicated, and so on. Domain expertise may allow obviously erroneous observations to

be identified as they are different from what is expected. There are general data cleaning operations that can be performed, such as:

- Using statistics to define normal data and identify outliers.
- Identifying columns that have the same value or no variance and removing them.
- Identifying duplicate rows of data and removing them.
- Marking empty values as missing.
- Imputing missing values using statistics or a learned model.

2. Feature Selection: Feature selection refers to techniques for selecting a subset of input features that are most relevant to the target variable that is being predicted. Feature selection techniques are generally grouped into those that use the target variable (**supervised**) and those that do not (**unsupervised**). Additionally, the supervised techniques can be further divided into models that automatically select features as part of fitting the model (**intrinsic**), those that explicitly choose features that result in the best performing model (**wrapper**) and those that score each input feature and allow a subset to be selected (**filter**).

3. **Data Transforms**: Data transforms are used to change the type or distribution of data variables.

- **Numeric Data Type:** Number values.
- **Integer:** Integers with no fractional part.
- **Real:** Floating point values.
- **Categorical Data Type:** Label values.
- **Ordinal:** Labels with a rank ordering.
- **Nominal:** Labels with no rank ordering.
- **Boolean:** Values True and False.

4. Feature Engineering: Feature engineering refers to the process of creating new input variables from the available data. Engineering new features is highly specific to your data and data types. As such, it often requires the collaboration of a subject matter expert to help identify new features that could be constructed from the data.

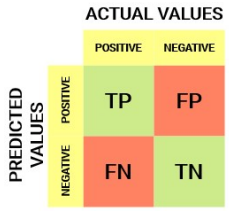
5. Dimensionality Reduction: The number of input features for a dataset may be considered the dimensionality of the data. This motivates feature selection, although an alternative to feature selection is to create a projection of the data into a lower-dimensional space that still preserves the most important properties of the original data. The most common approach to dimensionality reduction is to use a matrix factorization technique:

- Principal Component Analysis (PCA)

- Linear Discriminant Analysis (LDA)

Confusion Matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.



The explanation of the terms associated with confusion matrix are as follows −

- **True Positives (TP)** − It is the case when both actual class & predicted class of data point is 1.
- **True Negatives (TN)** − It is the case when both actual class & predicted class of data point is 0.
- **False Positives (FP)** − It is the case when actual class of data point is 0 & predicted class of data point is 1.
- **False Negatives (FN)** − It is the case when actual class of data point is 1 & predicted class of data point is 0.

We can find the confusion matrix with the help of confusion_matrix() function of sklearn.

```
from sklearn.metrics import accuracy_score,confusion_matrix
```

- **Accuracy**: It may be defined as the number of correct predictions made by our ML model.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- **Precision:**Precision, used in document retrievals, may be defined as the number of correct documents returned by our ML model.

$$Precision = \frac{TP}{TP + FP}$$

- **Recall or Sensitivity:** Recall may be defined as the number of positives returned by our ML model.

$$Recall = \frac{TP}{TP + FN}$$

- **Specificity:** Specificity, in contrast to recall, may be defined as the number of negatives returned by our ML model.

$$Specificity = \frac{TN}{TN + FP}$$

Conclusion: Thus we have studied different data preparation techniques.

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Loading the data
df = pd.read_csv('heart.csv')
print(df.head(3))
print()
#Features of the data set
print('Below are the features of dataset:')
df.info()
#Dimensions of the dataset
print('Below are the diamensions of dataset:')
print('Number of rows in the dataset: ',df.shape[0])
print('Number of columns in the dataset: ',df.shape[1])

#Checking for null values in the dataset
print('Checking for null values in the dataset:')
print(df.isnull().sum())
print(df.describe())
#The describe() features described in the above data set are:
#Count tells us the number of NoN-empty rows, Mean, Standard Deviation, Minimum, and Maximum values &25%, 50%, and 75% are the percentile/quartile of each features.
#Making predictions
```

```
predict = model.predict(X_test)
#Checking accuracy
from sklearn.metrics import accuracy_score,confusion_matrix
print()
print('Accuracy Score: ',accuracy_score(y_test,predict))
print('Using k-NN we get an accuracy score of: ',
      round(accuracy_score(y_test,predict),5)*100,'%')
print()
#Confusion Matrix
class_names = [0,1]
fig,ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks,class_names)
plt.yticks(tick_marks,class_names)
cnf_matrix = confusion_matrix(y_test,predict)
print('Below is the confusion matrix')
print(cnf_matrix)
```

Output

Glimpse of the dataset

```
In [3]: dataset.head()
```

Out[3]:

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	1	63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
1	2	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
2	3	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
3	4	37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
4	5	41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

Features of the dataset

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  303 non-null    int64
 1   Age         303 non-null    int64
 2   Sex         303 non-null    int64
 3   ChestPain   303 non-null    object
 4   RestBP      303 non-null    int64
 5   Chol        303 non-null    int64
 6   Fbs         303 non-null    int64
 7   RestECG     303 non-null    int64
 8   MaxHR       303 non-null    int64
 9   ExAng       303 non-null    int64
 10  Oldpeak     303 non-null    float64
 11  Slope       303 non-null    int64
 12  Ca          299 non-null    float64
 13  Thal        301 non-null    object
 14  AHD         303 non-null    object
dtypes: float64(2), int64(10), object(3)
memory usage: 35.6+ KB
```

Check for null values in the dataset

```
In [6]: dataset.isnull().any()
```

```
Out[6]: Unnamed: 0    False
        Age           False
        Sex           False
        ChestPain     False
        RestBP        False
        Chol          False
        Fbs           False
        RestECG       False
        MaxHR         False
        ExAng         False
        Oldpeak       False
        Slope         False
        Ca             True
        Thal           True
        AHD           False
        dtype: bool
```

Statistical features

```
In [10]: dataset.describe()
```

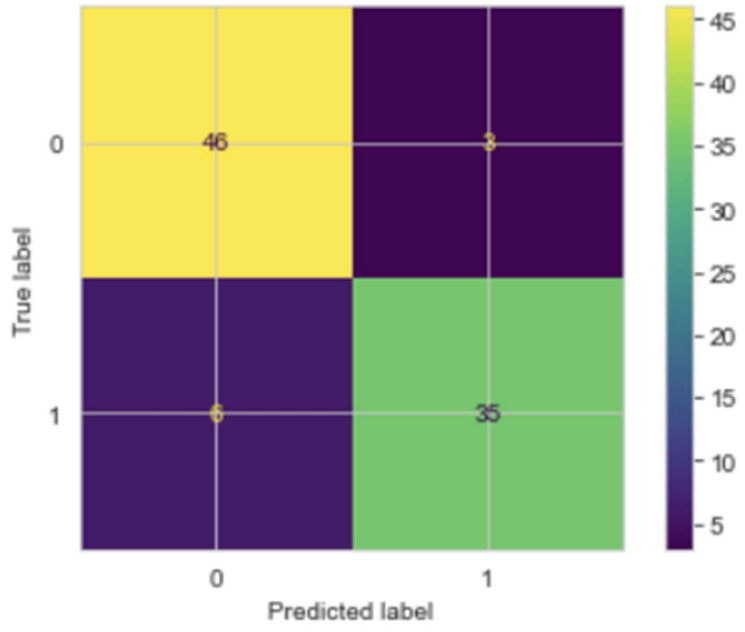
Out[10]:

	Unnamed: 0	Age	Sex	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca
count	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000
mean	150.673401	54.542088	0.676768	131.693603	247.350168	0.144781	0.996633	149.599327	0.326599	1.055556	1.602694	0.676768
std	87.323283	9.049736	0.468500	17.762806	51.997583	0.352474	0.994914	22.941562	0.469761	1.166123	0.618187	0.938965
min	1.000000	29.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	1.000000	0.000000
25%	75.000000	48.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.000000	0.000000	0.000000	1.000000	0.000000
50%	150.000000	56.000000	1.000000	130.000000	243.000000	0.000000	1.000000	153.000000	0.000000	0.800000	2.000000	0.000000
75%	226.000000	61.000000	1.000000	140.000000	276.000000	0.000000	2.000000	166.000000	1.000000	1.600000	2.000000	1.000000
max	302.000000	77.000000	1.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000	3.000000

Classification report

```
In [41]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.94      0.91        49
           1       0.92      0.85      0.89        41

    accuracy                           0.90        90
   macro avg       0.90      0.90      0.90        90
weighted avg       0.90      0.90      0.90        90
```



ASSIGNMENT 2

Title: **Assignment on Classification technique**

Aim: Every year many students give the GRE exam to get admission in foreign Universities. The data set contains GRE Scores (out of 340), TOEFL Scores (out of 120), University Rating (out of 5), Statement of Purpose strength (out of 5), Letter of Recommendation strength (out of 5), Undergraduate GPA (out of 10), Research Experience (0=no, 1=yes), Admitted (0=no, 1=yes). Admitted is the target variable.

Data Set Available on kaggle (The last column of the dataset needs to be changed to 0 or 1)Data Set : https://www.kaggle.com/mohansacharya/graduate-admissions
The counselor of the firm is supposed check whether the student will get an admission or not based on his/her GRE score and Academic Score. So to help the counselor to take appropriate decisions build a machine learning model classifier using Decision tree to predict whether a student will get admission or not.
Apply Data pre-processing (Label Encoding, Data Transformation….) techniques if necessary.
Perform data-preparation (Train-Test Split)
C. Apply Machine Learning Algorithm
D. Evaluate Model.

Software Requirements: Jupyter

Theory:

Classification: Classification may be defined as the process of predicting class or category from observed values or given data points. The categorized output can have the form such as “Black” or “White” or “spam” or “no spam”.Mathematically, classification is the task of approximating a mapping function (f) from input variables (X) to output variables (Y). It is basically belongs to the supervised machine learning in which targets are also provided along with the input data set.

Building a Classifier in Python:

Step1: Importing necessary python package

Step2: Importing dataset

Step3: Organizing data into training & testing sets

Step4: Model evaluation

Step5: Finding accuracy

Classification Algorithms Include:

Naive Bayes

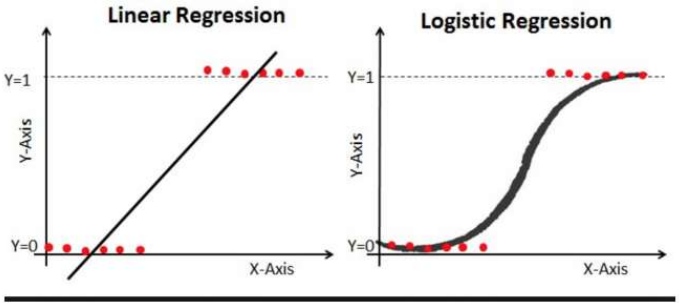
Logistic regression

K-nearest neighbors

(Kernel) SVM

Decision tree

1. **Logistic Regression Algorithm:** It is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). Logistic regression model predicts P(Y=1) as a function of X.



Logistic Regression Algorithm Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

1. Data Pre-processing step
2. Fitting Logistic Regression to the Training set
3. Predicting the test result
4. Test accuracy of the result(Creation of Confusion matrix)
5. Visualizing the test set result.

Example:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
X, y = load_iris()
LR_classifier = LogisticRegression(random_state=0)
LR_classifier.fit(X, y)
LR_classifier.predict(X[:3, :])
```

Output:

```
array([0, 0, 0])
```

It predicted 0 class for all 3 tests given to predict function.

Pros:

1. It is a very simple and efficient algorithm.
2. Low variance.
3. Provides probability score for observations.

Cons:

1. Bad at handling a large number of categorical features.
2. It assumes that the data is free of missing values and predictors are independent of each other.

2. **Decision Tree Algorithm:** Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. Decisions tress is the most powerful algorithms that falls under the category of supervised algorithms.

Decision Tree Algorithm Steps:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Solve decision tree such problems there is a technique which is called as **Attribute selection measure or ASM**. There are two popular techniques for ASM, which are:

1. **Information Gain:** Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class. According to the value of information gain, we split the node and build the decision tree.

   Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

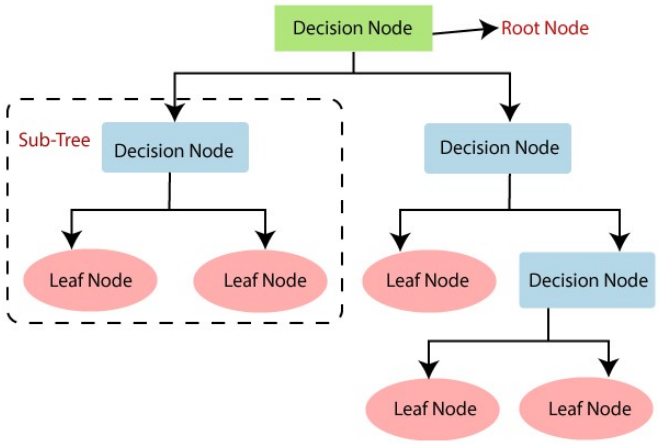
2. **Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

   Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

   Where,S= Total number of samples, P(yes)= probability of yes, P(no)= probability of no

3. **Gini Index:** Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm. An attribute with the low Gini index should be preferred as compared to the high Gini index.

   Gini Index= 1- ∑jPj2



Example:

```
from sklearn import tree
dtc = tree.DecisionTreeClassifier()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=142)
dtc.fit(X_train, y_train)
prediction_results = dtc.predict(X_test[:7,:])
print(prediction_results)
```

Output:array([0, 1, 1, 2, 1, 1, 0])

Pros:

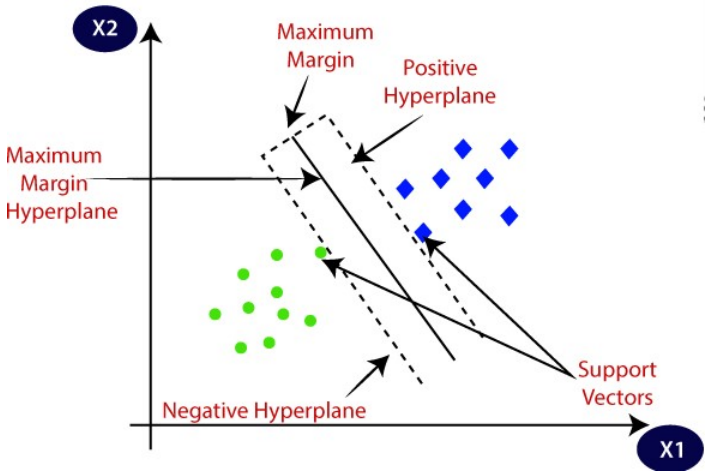
1. When compared to other algorithms, decision trees require less effort for data preparation while pre-processing.
2. They do not require normalization of data and scaling as well.
3. Model made on decision tree is very intuitive and easy to explain to technical teams as well as to stakeholders also.

Cons:

1. If even a small change is done in the data that can lead to a large change in the structure of the decision tree causing instability.
2. Sometimes calculation can go far more complex compared to other algorithms.
3. Decision trees often take higher time to train the model.

3. SVM Algorithm: Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future.

This best decision boundary is called a hyperplane.SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane.



SVM Algorithm Steps:

1. Importing the dataset
2. Splitting the dataset into training and test samples
3. Classifying the predictors and target
4. Initializing Support Vector Machine and fitting the training data
5. Predicting the classes for test set
6. Attaching the predictions to test set for comparing
7. Comparing the actual classes and predictions
8. Calculating the accuracy of the predictions

Pros:

1. SVM works relatively well when there is a clear margin of separation between classes.
2. SVM is more effective in high-dimensional spaces.

Cons:

1. SVM is not suitable for large data sets.
2. SVM does not perform very well when the data set has more noise i.e. when target classes are overlapping. So, it needs to be handled.

Applications of Classifications Algorithms:

1. Sentiment Analysis
2. Email Spam Classification
3. Document Classification
4. Image Classification

Conclusion: Thus we have studied different classification techniques.

Code

```
from sklearn import svm
svm_clf = svm.SVC()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=142)
svm_clf.fit(X_train, y_train)
prediction_results = svm_clf.predict(X_test[:7,:])
print(prediction_results)
```

Output:array([0, 1, 1, 2, 1, 1, 0])

```
"""Data Description
The dataset contains information about a student's:
GRE Score
TOEFL Score
University Ratings
Statement of Purpose Score
Letter of Recomendation Score
CGPA
Whether the Student Has Done Any Research Chance of Admission (What We're Trying to Predict) """
## To load the dataset
import pandas as pd
import matplotlib.pyplot as plt
#seaborn: for data visualization and exploratory data analysis
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
#Read data in csv file store into dataframe
df = pd.read_csv('Admission_Predict.csv')
```

```
print(df.head(5))
#################################################################################
#To drop the irrelevant column and check if there are any null values in the dataset
df = df.drop(['Serial No.'], axis=1)
print(df.isnull().sum())
#################################################################################
#To see the distribution of the variables of graduate applicants.
#distplot() plot distributed data as observations
#KDE: Kerner Density Estimate, probability density function of a continuous random variable Show GRE Score
fig = sns.distplot(df['GRE Score'], kde=False)
plt.title("Distribution of GRE Scores")
plt.show()

#Show TOEFL Score
fig = sns.distplot(df['TOEFL Score'], kde=False)
plt.title("Distribution of TOEFL Scores")
plt.show()

#Show University Ratings
fig = sns.distplot(df['University Rating'], kde=False)
plt.title("Distribution of University Rating")
plt.show()

#Show SOP Ratings
fig = sns.distplot(df['SOP'], kde=False)
plt.title("Distribution of SOP Ratings")
plt.show()
```

```
#Show CGPA

fig = sns.distplot(df['CGPA'], kde=False)

plt.title("Distribution of CGPA")

plt.show()

#It is clear from the distributions, students with varied merit apply  for the university.

#############################################################################

#Understanding the relation between different factors responsible for graduate admissions GRE Score vs TOEFL Score

#regplot() :Plot data and a linear regression model fit.

fig = sns.regplot(x="GRE Score", y="TOEFL Score", data=df)

plt.title("GRE Score vs TOEFL Score")

plt.show()

#People with higher GRE Scores also have higher TOEFL Scores which is justified because both TOEFL and GRE have a verbal section which although not similar are relatable


#GRE Score vs CGPA

fig = sns.regplot(x="GRE Score", y="CGPA", data=df)

plt.title("GRE Score vs CGPA")

plt.show()

#Although there are exceptions, people with higher CGPA usually have higher GRE scores maybe because they are smart or hard working


#LOR vs CGPA show wheather Research 0 or 1

#lmplot():a 2D scatterplot with an optional overlaid regression line.

#hue: Variables that define subsets of the data, which will be drawn on separate facets in the grid.

fig = sns.lmplot(x="CGPA", y="LOR ", data=df, hue="Research")

plt.title("LOR vs CGPA")

plt.show()

#LORs (Letter of Recommendation strength) are not that related with CGPA  so it is clear that a persons LOR is not dependent on that persons academic excellence.
```

#Having research experience is usually related with a good LOR which might be justified by the fact that supervisors have personal interaction with the students performing research which usually results in good LORs

#GRE Score vs LOR SHOW WHEATHER Research 0 or 1

```
fig = sns.lmplot(x="GRE Score", y="LOR ", data=df, hue="Research")

plt.title("GRE Score vs LOR")

plt.show()
```

#GRE scores and LORs are also not that related. People with different kinds of LORs have all kinds of GRE scores

#SOP vs CGPA

```
fig = sns.regplot(x="CGPA", y="SOP", data=df)

plt.title("SOP vs CGPA")

plt.show()
```

#CGPA and SOP are not that related because Statement of Purpose is related to academic performance, but since people with good CGPA tend to be more hard working so they have good things to say in their SOP which might explain the slight move towards higher CGPA as along with good SOPs

#GRE Score vs SOP

```
fig = sns.regplot(x="GRE Score", y="SOP", data=df)

plt.title("GRE Score vs SOP")

plt.show()
```

#Similary, GRE Score and CGPA is only slightly related

#SOP vs TOEFL

```
fig = sns.regplot(x="TOEFL Score", y="SOP", data=df)

plt.title("SOP vs TOEFL")

plt.show()
```

#Applicants with different kinds of SOP have different kinds of TOEFL Score. So the quality of SOP is not always related to the applicants English skills.

```
#############################################################################

#Correlation among variables
import numpy as np
#corr():Find the pairwise correlation of all columns in the dataframe
corr = df.corr()
print(corr)
#plt.subplot:Crate a figure & set sub plots
fig, ax = plt.subplots(figsize=(8, 8))
#Make a diverging palette between two HUSL colors.
#cmap: colour map set
colormap = sns.diverging_palette(220, 10, as_cmap=True)
#zeros_like():Returns an array of given shape and type as given array, with zeros.
dropSelf = np.zeros_like(corr)
#np.triu_indices_from(dropSelf): Return indices of array
dropSelf[np.triu_indices_from(dropSelf)] = True
colormap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, cmap=colormap, linewidths=.5, annot=True, fmt=".2f", mask=dropSelf)
plt.show()

#############################################################################
#Lets split the dataset with training and testing set and prepare the
#inputs and outputs
from sklearn.model_selection import train_test_split
#drop col chances of admission
X = df.drop(['Chance of Admit '], axis=1)
y = df['Chance of Admit ']
#split data for training & tasting
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.20, shuffle=False)


################################################################################
#Lets use a bunch of different algorithms to see which model performs better from sklearn.metrics import accuracy_score
#DecisionTree, Random Forest, K Neighbor, SVR, Linear Regression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

#These methods predict the future applicant's chances of admission.
models = [['DecisionTree :',DecisionTreeRegressor()],
          ['Linear Regression :', LinearRegression()],
          ['SVM :', SVR()]]
print("Results...")

#For loop for generating model results
for name,model in models:
    model = model
#Fit training data of x & y axis
    model.fit(X_train, y_train)
#Pass predicted or test result
    predictions = model.predict(X_test)
#Difference between actual value & predicted value
    print(name, (np.sqrt(mean_squared_error(y_test, predictions))))
#Something as simple as Linear Regression performs the best in this case, which proves that complicated models doesnt always mean better results. There are situations when simple models are much better suited
```

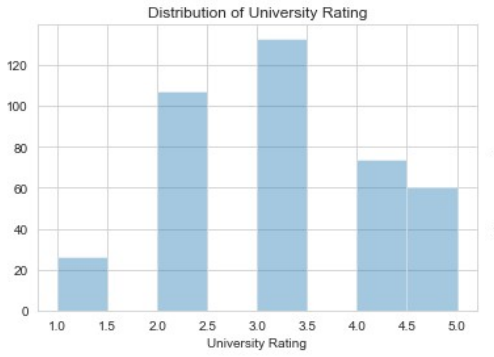
```
###############################################################################
#Generate Feature Importances
classifier = RandomForestRegressor()
classifier.fit(X,y)
#X.columns features in dataset
feature_names = X.columns
print(feature_names)
#Initialize importance_frame[] in 2 dim array.
importance_frame = pd.DataFrame()
#Two Dimensional Array Format column names
importance_frame['Features'] = X.columns
#classifier.feature_importance is decision tree based on correlation value As per importance of admission
importance_frame['Importance'] = classifier.feature_importances_
#Sort the features by high to low bar graph
importance_frame = importance_frame.sort_values(by=['Importance'], ascending=True)

###############################################################################
#Visualize 7 Feature Importances
#bar: plots horizontal rectangles with constant heights.
plt.barh([1,2,3,4,5,6,7], importance_frame['Importance'], align='center', alpha=0.5)
#yticks: set feature lable on y axis
plt.yticks([1,2,3,4,5,6,7], importance_frame['Features'])
plt.xlabel('Importance')

#Clearly, CGPA is the most factor for graduate admissions followed by GRE Score.
plt.title('Feature Importances')
plt.show()
```

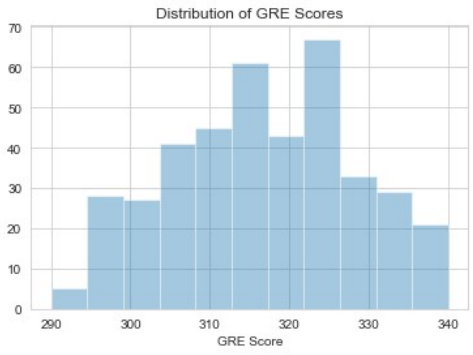
Output

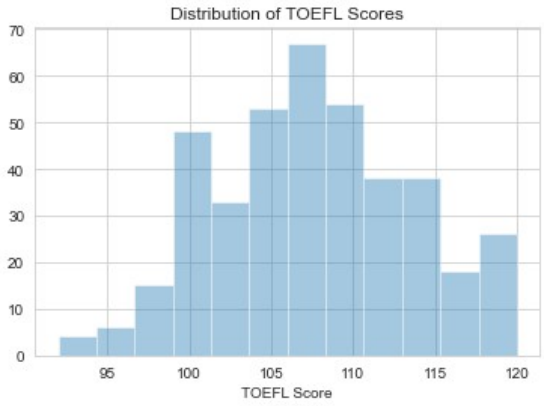
```
In [2]: runfile('C:/Users/Vrushali/Test_Project/Practicle_2.py', wdir='C:/Users/Vrushali/Test_Project')
   Serial No.  GRE Score  TOEFL Score  ...  CGPA  Research  Chance of Admit
0           1        337          118  ...  9.65         1              0.92
1           2        324          107  ...  8.87         1              0.76
2           3        316          104  ...  8.00         1              0.72
3           4        322          110  ...  8.67         1              0.80
4           5        314          103  ...  8.21         0              0.65

[5 rows x 9 columns]
GRE Score           0
TOEFL Score         0
University Rating   0
SOP                 0
LOR                 0
CGPA                0
Research            0
Chance of Admit     0
dtype: int64
```

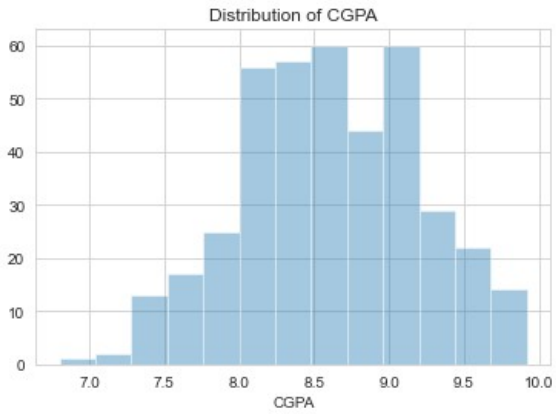
	GRE Score	TOEFL Score	...	Research	Chance of Admit
GRE Score	1.000000	0.835977	...	0.580391	0.802610
TOEFL Score	0.835977	1.000000	...	0.489858	0.791594
University Rating	0.668976	0.695590	...	0.447783	0.711250
SOP	0.612831	0.657981	...	0.444029	0.675732
LOR	0.557555	0.567721	...	0.396859	0.669889
CGPA	0.833060	0.828417	...	0.521654	0.873289
Research	0.580391	0.489858	...	1.000000	0.553202
Chance of Admit	0.802610	0.791594	...	0.553202	1.000000

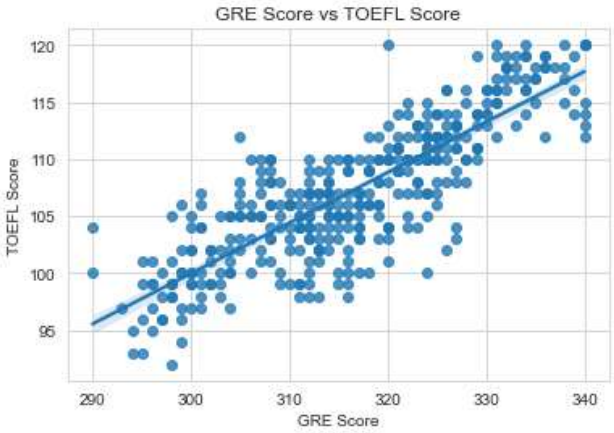
[8 rows x 8 columns]

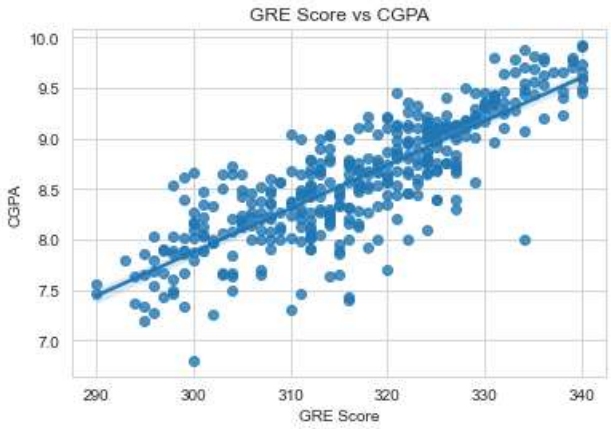


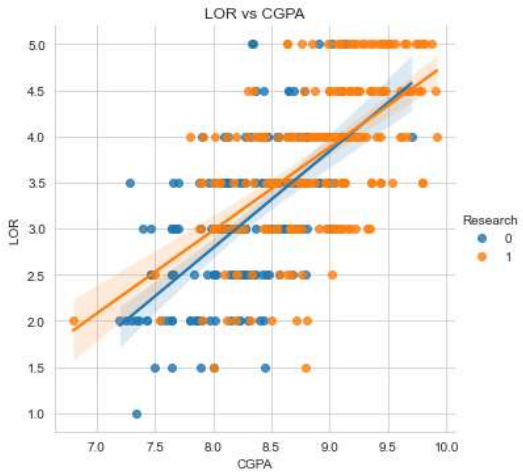


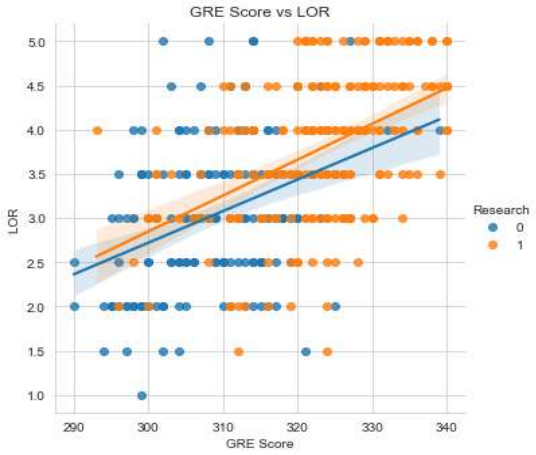


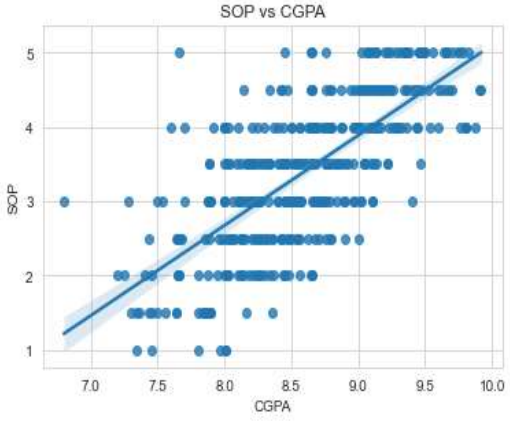

Distribution of CGPA
CGPA

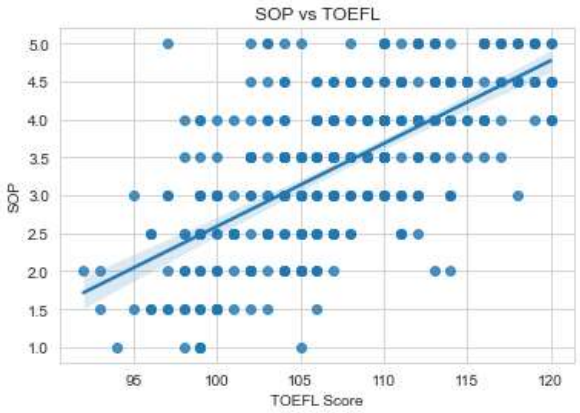

GRE Score vs TOEFL Score
TOEFL Score
GRE Score

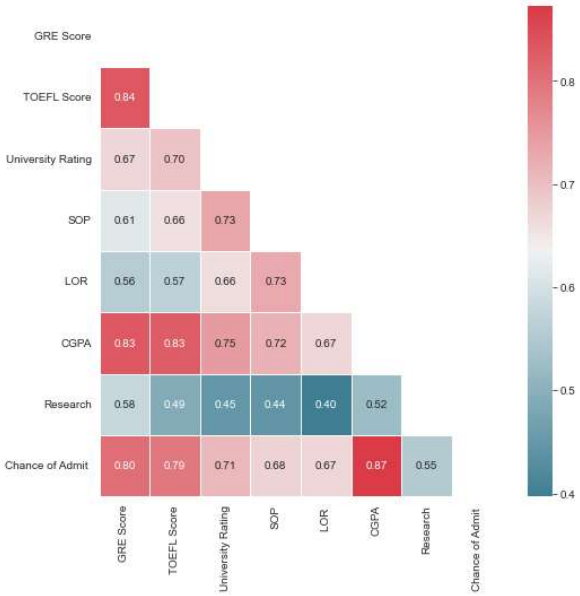

GRE Score vs CGPA
CGPA
GRE Score

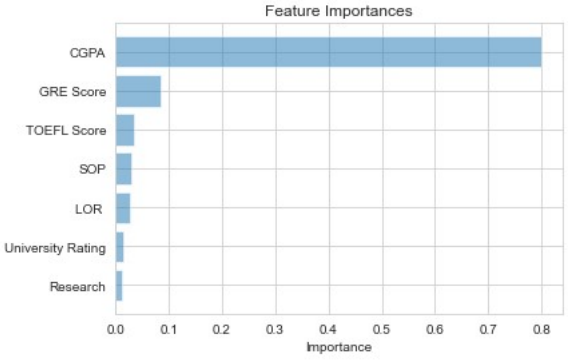

LOR vs CGPA
LOR
CGPA
Research
0
1


GRE Score vs LOR
LOR
GRE Score
Research
0
1


SOP vs CGPA
SOP
CGPA








```
[8 rows x 8 columns]
Results...
DecisionTree : 0.09370832406995656
Linear Regression : 0.06473311695782O9
SVM : 0.08180727044650482
Index(['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA',
       'Research'],
      dtype='object')
```

ASSIGNMENT 3

Title: **Assignment on Regression technique**

Aim:

a. Apply Linear Regression using suitable library function and predict the Month-wise
Download temperature data from below link.
https://www.kaggle.com/venky73/temperatures- of-india?select=temperatures.csv
This data consists of temperatures of INDIA averaging the temperatures of all places month wise. Temperatures values are recorded in CELSIUS temperature.
b. Assess the performance of regression models using MSE, MAE and R-Square metrics
c. Visualize simple regression model.

Software Requirement: Jupyter

Theory:

Regression:

Regression is a supervised learning technique which helps in finding the correlation between variables and enables us to predict the continuous output variable based on the one or more predictor variables. It is mainly used for prediction, forecasting, time series modeling and determining the causal-effect relationship between variables. In Regression, we plot a graph between the variables which best fits the given datapoints, using this plot, the machine learning model can make predictions about the data.
In simple words, "Regression shows a line or curve that passes through all the datapoints on target-predictor graph in such a way that the vertical distance between the datapoints and the regression line is minimum." The distance between datapoints and line tells whether a model has captured a strong relationship or not.

Terminologies Related to the Regression Analysis:

Dependent Variable: The main factor in Regression analysis which we want to predict or understand is called the dependent variable. It is also called target variable.

Independent Variable: The factors which affect the dependent variables or which are used to predict the values of the dependent variables are called independent variable, also called as a predictor.

Outliers: Outlier is an observation which contains either very low value or very high value in comparison to other observed values. An outlier may hamper the result, so it should be avoided.

Multicollinearity: If the independent variables are highly correlated with each other than other variables, then such condition is called Multicollinearity. It should not be present in the dataset, because it creates problem while ranking the most affecting variable.

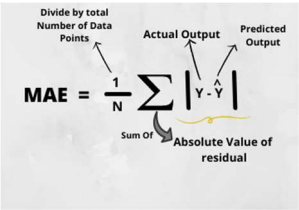
Underfitting and Overfitting: If our algorithm works well with the training dataset but not well with test dataset, then such problem is called Overfitting. And if our algorithm does not perform well even with training dataset, then such problem is called underfitting.

Cost Functions:

It is necessary to obtain the accuracy on training data, but it is also important to get a genuine and approximate result on unseen data otherwise Model is of no use. So to build and deploy a generalized model we require evaluating the model on different metrics which helps us to better optimize the performance, fine-tune it, and obtain a better result.

1. **Mean Absolute Error (MAE**): MAE is a very simple metric which calculates the absolute difference between actual and predicted values.

   ```
   from sklearn.metrics import mean_absolute_error
   print("MAE",mean_absolute_error(y_test,y_pred))
   ```



2. **Mean Squared Error(MSE):** Mean squared error states that finding the squared difference between actual and predicted value. we perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.

```
from sklearn.metrics import mean_squared_error
print("MSE",mean_squared_error(y_test,y_pred))
```

$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)}^{2}$$

The square of the difference between actual and predicted

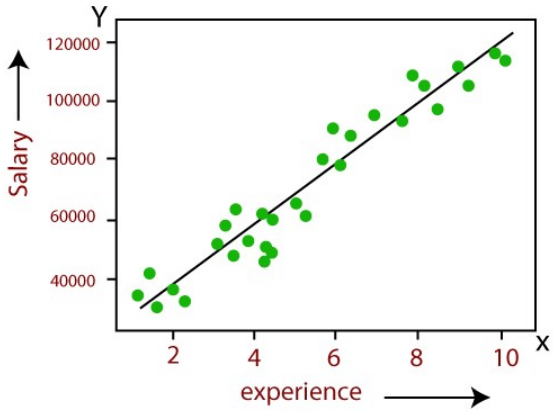
3. **Root Mean Squared Error(RMSE):** As RMSE is clear by the name itself, that it is a simple square root of mean squared error.

```
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
```

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)}$$

Linear Regression

Linear regression is a statistical regression method which is used for predictive analysis. It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables. It shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression. If there is only one input variable (x), then such linear regression is called simple linear regression. And if there is more than one input variable, then such linear regression is called multiple linear regression. The relationship between variables in the linear regression model can be explained using the below image. Here we are predicting the salary of an employee on the basis of the year of experience.



Below is the mathematical equation for Linear regression:

$$Y= aX+b$$

Here, Y= Independent Variable (Target Variable)

X= Dependent Variable (Predictor Variable)

Steps in Linear Regression:

1. Loading the Data
2. Exploring the Data
3. Slicing The Data
4. Train and Split Data
5. Generate The Model
6. Evaluate The accuracy

Applications of linear regression are:

1. Analysing trends and sales estimates
2. Salary forecasting
3. Real estate prediction &Arriving at ETAs in traffic

Pros:

1. Linear Regression is simple to implement.
2. Less complexity compared to other algorithms.
3. Linear Regression may lead to over-fitting but it can be avoided using some dimensionality reduction techniques, regularization techniques, and cross-validation.

Cons:

1. Outliers affect this algorithm badly.
2. It over-simplifies real-world problems by assuming a linear relationship among the variables, hence not recommended for practical use-cases.

Conclusion: Thus, we have studied Regression techniques.

Code

```
import pandas as pd
dataset = pd.read_csv('temperatures.csv')
dataset
print(dataset.keys())
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)

from sklearn import linear_model
regressor = linear_model.LinearRegression()
regressor.fit(x_train,y_train)
y_pred = regressor.predict(x_test)

accuracy = mean_squared_error(y_test,y_predict)
print(accuracy)
weights = regressor.coef_
intercept = regressor.intercept_
print(weights,intercept)

import matplotlib.pyplot as plt
plt.scatter(disease_X_test, disease_Y_test)
plt.plot(disease_X_test,y_predict)
plt.show()
```

Output

```
In [3]: dataset
```

Out[3]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	JAN-FEB	MAR-MAY	JUN-SEP	OCT-DEC
0	1901	22.40	24.14	29.07	31.91	33.41	33.18	31.21	30.39	30.47	29.97	27.31	24.49	28.96	23.27	31.46	31.27	27.25
1	1902	24.93	26.58	29.77	31.78	33.73	32.91	30.92	30.73	29.80	29.12	26.31	24.04	29.22	25.75	31.76	31.09	26.49
2	1903	23.44	25.03	27.83	31.39	32.91	33.00	31.34	29.98	29.85	29.04	26.08	23.65	28.47	24.24	30.71	30.92	26.26
3	1904	22.50	24.73	28.21	32.02	32.64	32.07	30.36	30.09	30.04	29.20	26.36	23.63	28.49	23.62	30.95	30.66	26.40
4	1905	22.00	22.83	26.68	30.01	33.32	33.25	31.44	30.68	30.12	30.67	27.52	23.82	28.30	22.25	30.00	31.33	26.57
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
112	2013	24.56	26.59	30.62	32.66	34.46	32.44	31.07	30.76	31.04	30.27	27.83	25.37	29.81	25.58	32.58	31.33	27.83
113	2014	23.83	25.97	28.95	32.74	33.77	34.15	31.85	31.32	30.68	30.29	28.05	25.08	29.72	24.90	31.82	32.00	27.81
114	2015	24.58	26.89	29.07	31.87	34.09	32.48	31.88	31.52	31.55	31.04	28.10	25.67	29.90	25.74	31.68	31.87	28.27
115	2016	26.94	29.72	32.62	35.38	35.72	34.03	31.64	31.79	31.66	31.98	30.11	28.01	31.63	28.33	34.57	32.28	30.03
116	2017	26.45	29.46	31.60	34.95	35.84	33.82	31.88	31.72	32.22	32.29	29.60	27.18	31.42	27.95	34.13	32.41	29.69

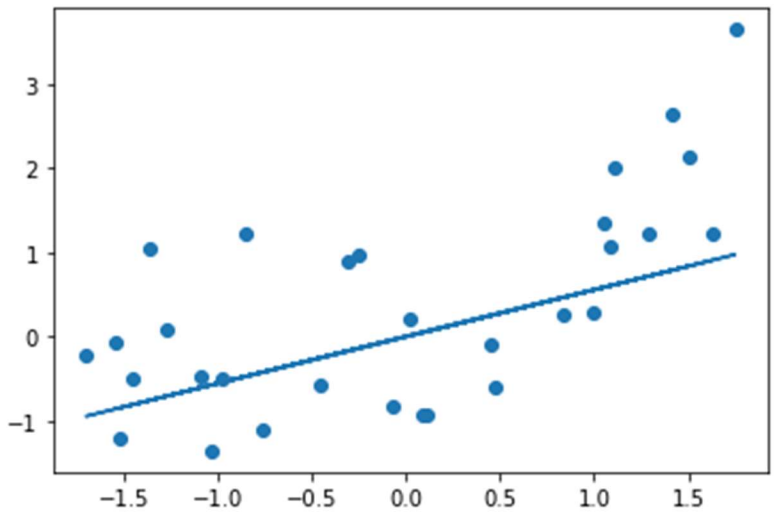
117 rows × 18 columns

In [20]:
```
dataset.keys()
```

Out[20]:
```
Index(['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP',
       'OCT', 'NOV', 'DEC', 'ANNUAL', 'JAN-FEB', 'MAR-MAY', 'JUN-SEP',
       'OCT-DEC'],
      dtype='object')
```

In [23]:
```
accuracy = mean_squared_error(y_test,y_pred)
print(accuracy)
weights = regressor.coef_
intercept = regressor.intercept_
print(weights,intercept)
```

```
1.0168731462240104
[[0.56]] [-1.79e-15]
```



ASSIGNMENT 4

Title: Assignment on Improving Performance of Classifier Models

Aim: A SMS unsolicited mail (every now and then known as cell smartphone junk mail) is any junk message brought to a cellular phone as textual content messaging via the Short Message Service (SMS). Use probabilistic approach (Naive Bayes Classifier / Bayesian Network) to implement SMS Spam Filtering system. SMS messages are categorized as SPAM or HAM using features like length of message, word depend, unique keywords etc.
Download Data -Set from: http://archive.ics.uci.edu/ml/datasets/sms+spam+collection

This dataset is composed by just one text file, where each line has the correct class followed by the raw message.

a. Apply Data pre-processing (Label Encoding, Data Transformation….) techniques if necessary
b. Perform data-preparation (Train-Test Split)
c. Apply at least two Machine Learning Algorithms and Evaluate Models
d. Apply Cross-Validation and Evaluate Models and compare performance.
e. Apply Hyper parameter tuning and evaluate models and compare performance

Software Requirement:

Theory:

Spam Filtering

Naive Bayes classifiers are a popular statistical technique of e-mail filtering. They typically use a bag of words features to identify spam e-mail, an approach commonly used in text classification. Naive Bayes classifiers work by correlating the use of tokens (typically words, or sometimes other things), with a spam and non-spam e-mails and then using Bayes' theorem to calculate a probability that an email is or is not spam. Particular words have particular probabilities of occurring in spam email and in legitimate email. For instance, most email users will frequently encounter the word "Lottery" and "Luck Draw" in spam email, but will seldom see it in other emails. Each word in the email contributes to the email's spam probability or only the most interesting words. This contribution is called the **posterior probability** and is computed using **Bayes' theorem.** Then, the email's spam probability is computed over all words in the email, and if the total exceeds a certain threshold (say 95%), the filter will mark the email as a spam.

Probability:

Probability gives the information about how likely an event can occur. Digging into the terminology of the probability:

Trial or Experiment: The act that leads to a result with certain possibility.
Sample space: The set of all possible outcomes of an experiment.
Event: Non empty subset of sample space is known as event.

Basic Probability Calculation:

$$P(A) = \frac{\text{No.of outcomes in A}}{\text{No. of outcomes in S}}$$

As per the definition, if A is an event of an experiment and it contains n outcomes and S is the sample space then,

$$\mathbf{P(A)=\sum_{i=1}^{n} P(Ei)}$$

Where E1.... En is the outcomes in A. If all the outcomes of the experiment are equally likely then

Bayes Theorem:

The Bayes theorem can be understood as the description of the probability of any event which is obtained by prior knowledge about the event. Bayes' theorem is also known as **Bayes' rule, Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

$$\mathbf{P(A|B)} = \frac{\mathbf{P(B|A)}\,P(A)}{P(B)} \quad \text{....(a)}$$

- P(A|B) is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.
- P(B|A) is called the likelihood, in which we consider that hypothesis is true, then we calculate the probability of evidence.
- P(A) is called the **prior probability**, probability of hypothesis before considering the evidence
- P(B) is called **marginal probability**, pure probability of an evidence.

Applying Bayes' rule:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$P(king|face) = \frac{P(Face|king) * P(King)}{P(Face)} \quad .....(i)$$

P(king): probability that the card is King= 4/52= 1/13

P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$$P(king|face) = \frac{1 * (\frac{1}{13})}{(\frac{3}{13})} = 1/3,$$ **it is a probability that a face card is a king card.**

Naive Bayes Algorithm:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naïve Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Python Implementation of the Naïve Bayes algorithm:

1. Data Pre-processing step
2. Fitting Naive Bayes to the Training set
3. Predicting the test result
4. Test accuracy of the result(Creation of Confusion matrix)
5. Visualizing the test set result.

Conclusion: Thus we have studied Naive Bayes Classifier.

Code

```
Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
# Importing the dataset
dataset = pd.read_csv('user_data.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
# 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 = 0.25, random_state = 0)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
# Predicting the Test set results
y_pred = classifier.predict(x_test)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
# Visualising the Training set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
 1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
```

```
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape)
alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
# Visualising the Test set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() -
 1, stop = x_set[:, 0].max() + 1, step = 0.01),
                nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
          alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
            c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output

dataset - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0
16	15733883	Male	47	25000	1
17	15617482	Male	45	26000	1
18	15704583	Male	46	28000	1
19	15621083	Female	48	29000	1

```
Out[6]: GaussianNB(priors=None, var_smoothing=1e-09)
```

y_pred - NumPy array

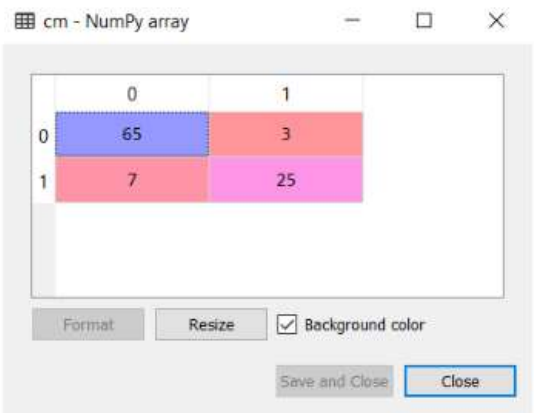
	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	1
10	0
11	0
12	0
13	0

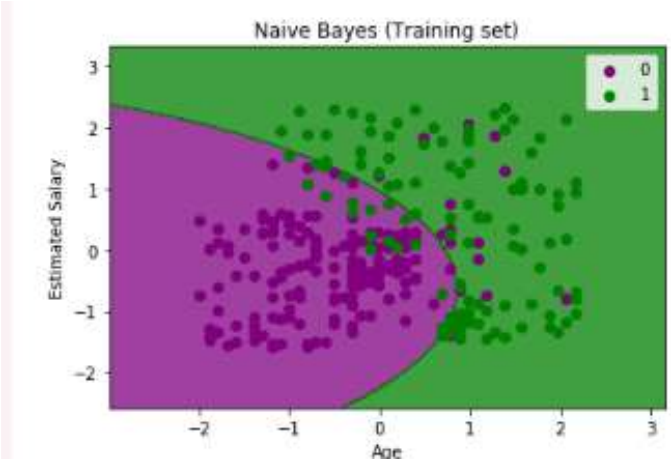
Format Resize Background color Save and Close Close

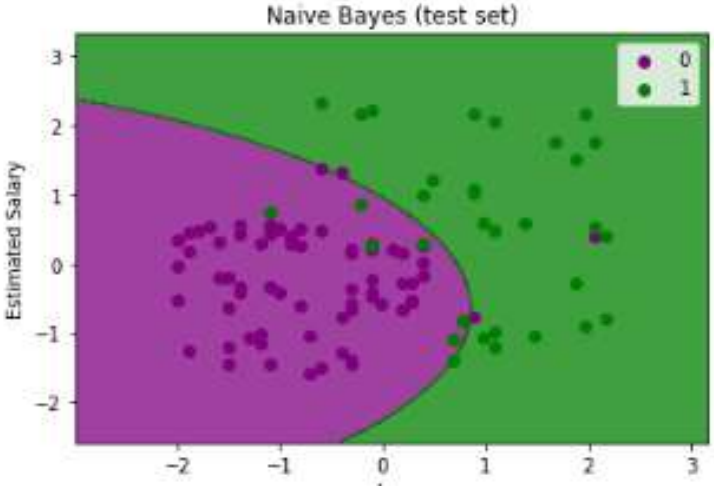
y_test - NumPy array

	0
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	0
12	0
13	0

Format Resize Background color Save and Close Close


cm - NumPy array
0
1
0
65
3
1
7
25
Format
Resize
Background color
Save and Close
Close


Naive Bayes (Training set)
Estimated Salary
Age
0
1


Naive Bayes (test set)
Estimated Salary
0
1

ASSIGNMENT 5

Title: **Assignment on Clustering Techniques**

Aim:

Download the following customer dataset from below link:
Data Set: https://www.kaggle.com/shwetabh123/mall-customers
This dataset gives the data of Income and money spent by the customers visiting a Shopping Mall. The data set contains Customer ID, Gender, Age, Annual Income, Spending Score. Therefore, as a mall owner you need to find the group of people who are the profitable customers for the mall owner. Apply at least two clustering algorithms (based on Spending Score) to find the group of customers.

a. Apply Data pre-processing (Label Encoding, Data Transformation….) techniques if necessary.
b. Perform data-preparation(Train-Test Split)
c. Apply Machine Learning Algorithm
d. Evaluate Model.
e. Apply Cross-Validation and Evaluate Model

Practical Implementation: https://youtu.be/SrY0sTJchHE

Software Requirement: juyter

Theory:

Approach of Clustering

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as ″A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group." It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns. It is an unsupervised learning method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

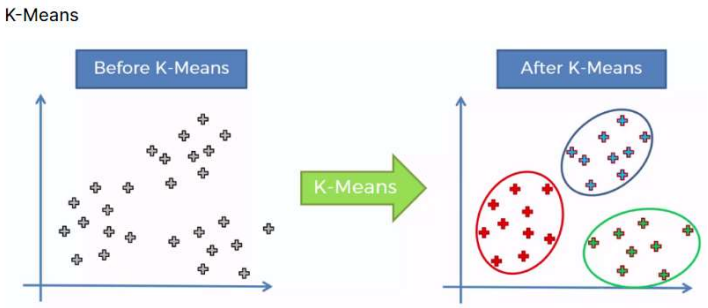
Applications of Clustering:

- Market Segmentation
- Statistical data analysis
- Social network analysis
- Image segmentation

- Anomaly detection, etc.

K-Means Clustering:

K-Means clustering is the most popular unsupervised learning algorithm. It is used when we have unlabelled data which is data without defined categories or groups. The algorithm follows an easy or simple way to classify a given data set through a certain number of clusters, fixed apriori. K-Means algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity



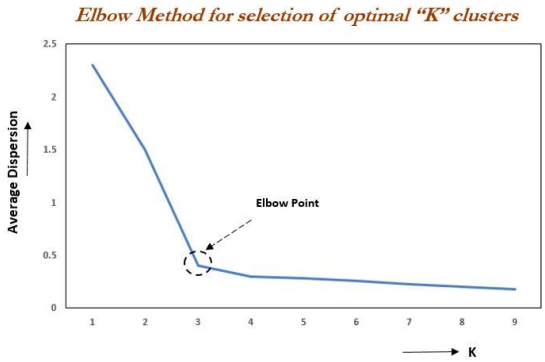
K-Means Algorithm:

- **Step-1:** Select the number K to decide the number of clusters.
- **Step-2:** Select random K points or centroids. (It can be other from the input dataset).
- **Step-3:** Assign each data point to their closest centroid, which will form the predefined K clusters.
- **Step-4:** Calculate the variance and place a new centroid of each cluster.
- **Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- **Step-6:** If any reassignment occurs, then go to step-4 else go to FINISH.
- **Step-7**: The model is ready.

K-Means Clustering Intuition:

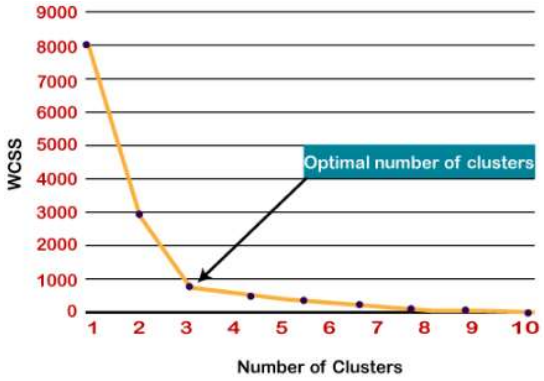
1. **Centroid:** A centroid is a data point at the centre of a cluster. In centroid-based clustering, clusters are represented by a centroid. The algorithm requires number of clusters K and the data set as input. The data set is a collection of features for each data point. The algorithm starts with initial estimates for the K centroids.
2. **Data Assignment Step:** Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, which is based on the squared Euclidean distance. So, if ci is the collection of centroids in set C, then each data point is assigned to a cluster based on minimum Euclidean distance.

3. **Centroid update Step**: In this step, the centroids are recomputed and updated. This is done by taking the mean of all data points assigned to that centroid's cluster.
4. **Choosing the value of K**: The K-Means algorithm depends upon finding the number of clusters and data labels for a pre-defined value of K. We should choose the optimal value of K that gives us best performance. There are different techniques available to find the optimal value of K. The most common technique is the elbow method.
5. **The elbow method:** The elbow method is used to determine the optimal number of clusters in K-means clustering. The elbow method plots the value of the cost function produced by different values of K



6. **WCSS List:** Elbow method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.



Python Implementation of K-means Clustering Algorithm

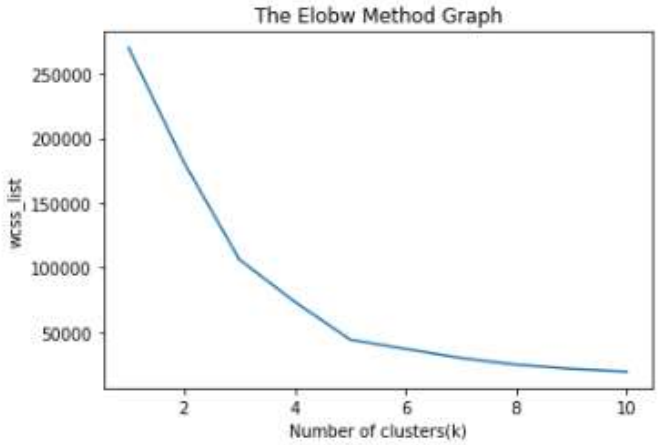
The steps to be followed for the implementation are given below:

1. Data Pre-processing
2. Finding the optimal number of clusters using the elbow method
3. Training the K-means algorithm on the training dataset
4. Visualizing the clusters

Pseudocode of K-Means

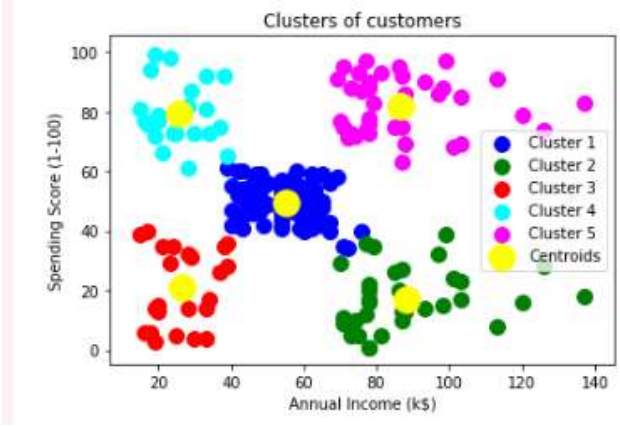
#finding optimal number of clusters using the elbow method

```
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS
#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
kmeans.fit(x)
wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```



#training the K-means model on a dataset

1. kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
2. y_predict= kmeans.fit_predict(x)



Conclusion: Thus, we have studied Clustering techniques.

Code

```
#Import required libraries
import pandas as pd
# Visualization Library
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
# Scaling
from sklearn.preprocessing import StandardScaler
# Dimensional
from sklearn.decomposition import PCA
# Clustering
from sklearn.cluster import KMeans
#import numpy as np
#import seaborn as sns

#Load Data csv file
data = pd.read_csv('Mall_Customers.csv')

#Data Preprocessing Steps
print('For printing sample data:')
print(data.head())
print() #for creating blank space

print('To get total rows and columns:')
print(data.shape)
print()

#Column names, Count, Data Types, Null Values
```

```
print('To get info about columns:')
print(data.info())
print()

#Rename column name
data.rename(columns = {'Genre':'Gender'} , inplace = True)

#Describe Datasets
print(data.describe())
print()

#Drop useless columns
data.drop(labels = 'CustomerID' , axis = 1 , inplace = True)

#Missing values
print(data.isnull().sum())
print()

#Encoding finding data types of data present in csv
print(data.dtypes)
print()

#Find Gender Counts
print(data['Gender'].value_counts())
#Consider Male=1 & Female=0
data['Gender'].replace({'Male':1 , 'Female':0} , inplace = True)
print(data.info())
```

```
#Scaling
#Clustering algorithms such as K-means do need feature scaling before they are fed to the algorithm.
# Since, clustering techniques use Euclidean Distance to form the cohorts,
#it will be wise to scale the variables.
#Data coverted into normalization distribution
sc = StandardScaler()
data_scaled = sc.fit_transform(data)

#Dimensionality reduction
pca = PCA(n_components = 2)
data_pca = pca.fit_transform(data_scaled)
print("data shape after PCA :",data_pca.shape)
print("data_pca is:",data_pca)

# KMeans Clustering
''' Elbow plot Details : Finding optimal value of clusters
K is a hyperparameter in KMeans algorithm.
WCSS : Within Cluster Sum of Squares, in other word it's sum of squared
distance between each point and the centroid in a cluster
Lower WCSS shows a better clustering(because points in a cluster are more similar to each other,
 this is what we want)
Increasing the k value always results in a lower WCSS.
if we put k to be equal to the number of samples(so each point is a special cluster)
then WCSS = 0 , but this is not a wise way.
Here we will use elbow plot to find the best k.
Elbow point will show the best k.
How to find this point?
After this point the speed of WCSS decreasing will be lowered. '''
```

```
#font size
plt_font = {'family':'serif', 'size':16}

"'WCSS: Within Cluster Sum of Squares, in other word it's sum of squared
distance between each point and the centroid in a cluster
Lower WCSS shows a better clustering (because points in a cluster are more similar to each other,
 this is what we want)
Increasing the k value always results in a lower WCSS.'"

#Create blank list
#Minimum no. of clusters & squared distance
wcss_list = []
for i in range(1, 15):
    kmeans = KMeans(n_clusters = i , init = 'k-means++'  , random_state = 1)
    kmeans.fit(data_pca)
    wcss_list.append(kmeans.inertia_)

#Draw Elbow plot
#X & Y axis range
plt.plot(range(1,15) , wcss_list)
plt.plot([4,4] , [0 , 500] , linestyle = '--' , alpha = 0.7)
#Elbow line
plt.text(4.2 , 300 , 'Elbow = 4')
#X & Y axis labels
plt.xlabel('K' , fontdict = plt_font)
plt.ylabel('WCSS' , fontdict = plt_font)
plt.show()
```

```
#KMeans Algorithm
kmeans = KMeans(n_clusters = 4 , init = 'k-means++'  , random_state = 1)
kmeans.fit(data_pca)
cluster_id = kmeans.predict(data_pca)

result_data = pd.DataFrame()
result_data['PC1'] = data_pca[:,0]
result_data['PC2'] = data_pca[:,1]
result_data['ClusterID'] = cluster_id

#KMeans clustered ploting features
#cluster colors & tab details
cluster_colors = {0:'tab:red' , 1:'tab:green' , 2:'tab:blue' , 3:'tab:pink'}
cluster_dict = {'Centroid':'tab:orange','Cluster0':'tab:red' , 'Cluster1':'tab:green'
            , 'Cluster2':'tab:blue' , 'Cluster3':'tab:pink'}

#Scatter data
#X & Y Value, result & cluster colors
plt.scatter(x = result_data['PC1'] , y = result_data['PC2']
            , c = result_data['ClusterID'].map(cluster_colors))

handles = [Line2D([0], [0], marker='o', color='w', markerfacecolor=v, label=k, markersize=8)
        for k, v in cluster_dict.items()]
plt.legend(title='color', handles=handles, bbox_to_anchor=(1.05, 1), loc='upper left')

plt.scatter(x = kmeans.cluster_centers_[:,0] , y = kmeans.cluster_centers_[:,1] ,
          marker = 'o' , c = 'tab:orange', s = 150 , alpha = 1)
```

#Heading details

plt.title("Clustered by KMeans" , fontdict = plt_font)

plt.xlabel("PC1" , fontdict = plt_font)

plt.ylabel("PC2" , fontdict = plt_font)

#Show all data

plt.show()

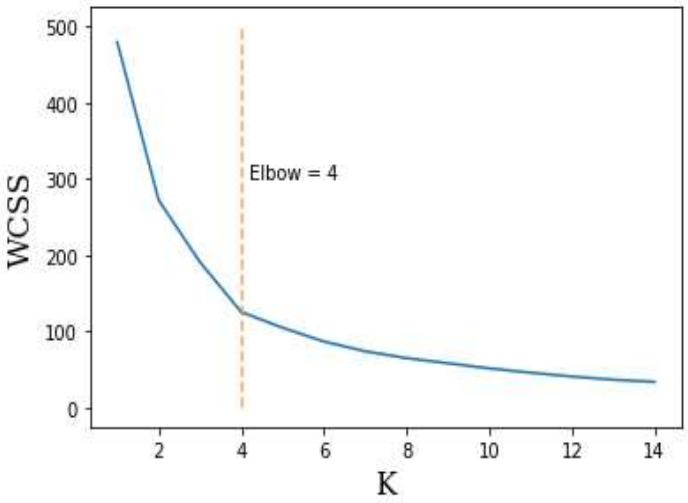
Output:

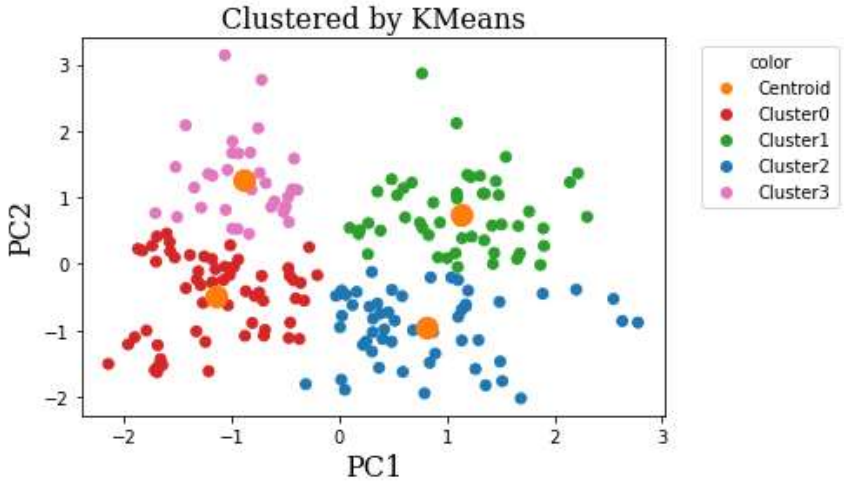
```
For printing sample data:
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16           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

To get total rows and columns:
(200, 5)

To get info about columns:
<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 (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
None
```

```
       CustomerID         Age  Annual Income (k$)  Spending Score (1-100)
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

Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64

Gender                    object
Age                        int64
Annual Income (k$)         int64
Spending Score (1-100)     int64
dtype: object

Female    112
Male       88
Name: Gender, dtype: int64
```

```
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Gender                  200 non-null    int64
 1   Age                     200 non-null    int64
 2   Annual Income (k$)      200 non-null    int64
 3   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4)
memory usage: 6.4 KB
None
data shape after PCA : (200, 2)
data_pca is: [[-4.06382715e-01 -5.20713635e-01]
```

```
data_pca is: [[-4.06382715e-01 -5.20713635e-01]
 [-1.42767287e+00 -3.67310199e-01]
 [ 5.07605706e-02 -1.89406774e+00]
 [-1.69451310e+00 -1.63190805e+00]
 [-3.13108383e-01 -1.81048272e+00]
 [-1.71744627e+00 -1.59926418e+00]
 [ 7.90821124e-01 -1.94727112e+00]
 [-2.14832159e+00 -1.50537369e+00]
 [ 2.77428623e+00 -8.82987672e-01]
 [-1.21629477e+00 -1.61640464e+00]
 [ 2.62905084e+00 -8.61237043e-01]
 [-1.68947038e+00 -1.54542784e+00]
 [ 1.68582253e+00 -2.02394479e+00]
 [-1.64607339e+00 -1.52251259e+00]
 [ 1.17443628e+00 -6.12790961e-01]
 [-1.32613070e+00 -2.36719149e-01]
 [ 1.67728253e-02 -1.74344572e+00]
 [-1.07842454e+00 -2.44715641e-01]
 [ 1.48758780e+00 -5.72676028e-01]
 [-1.66373169e+00 -1.43259774e+00]
 [ 4.88090311e-01 -3.92921145e-01]
 [-1.01895051e+00 -1.66247511e-01]
 [ 1.35891492e+00 -1.82866936e+00]
 [-7.22972722e-01 -1.81687017e-01]
 [ 1.51315931e+00 -1.76451196e+00]
```





ASSIGNMENT 6

Title: **Assignment on Association Rule Learning**

Aim:

Download Market Basket Optimization dataset from below link.
Data Set: https://www.kaggle.com/hemanthkumar05/market-basket-optimization
This dataset comprises the list of transactions of a retail company over the period of one week. It contains a total of 7501 transaction records where each record consists of the list of items sold in one transaction. Using this record of transactions and items in each transaction, find the association rules between items.
There is no header in the dataset and the first row contains the first transaction, so mentioned header = None here while loading dataset.
a. Follow following steps :
b. Data Preprocessing
c. Generate the list of transactions from the dataset
d. Train Apriori algorithm on the dataset
e. Visualize the list of rules
f. Generated rules depend on the values of hyper parameters. By increasing the minimum confidence value and find the rules accordingly

Software Requirement: Jupyter

Theory:

Association Rule Learning:

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database. The association rule learning is one of the very important concepts of machine learning, and it is employed in Market Basket analysis, Web usage mining, continuous production, etc.

How does Association Rule Learning work?

Association rule learning works on the concept of If and Else Statement, such as if A then B.

Here the If element is called antecedent, and then statement is called as Consequent. These types of relationships where we can find out some association or relation between two items is known as single cardinality

Apriori Algorithm in Machine Learning:

The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm was given by the **R.** Agrawal and Srikant in the year 1994. It is mainly used for *market basket analysis* and helps to find those products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

What is Frequent Itemset?

Frequent itemsets are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent itemsets together, then individually A and B should also be the frequent itemset. Suppose there are the two transactions: A= {1,2,3,4,5}, and B= {2,3,7}, in these two transactions, 2 and 3 are the frequent itemsets.

Steps for Apriori Algorithm:

Step-1: Determine the support of itemsets in the transactional database, and select the minimum support and confidence.

Step-2: Take all supports in the transaction with higher support value than the minimum or selected support value.

Step-3: Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.

Step-4: Sort the rules as the decreasing order of lift.

Apriori Algorithm Working:

Example: Suppose we have the following dataset that has various transactions, and from this dataset, we need to find the frequent itemsets and generate the association rules using the Apriori algorithm:

TID	ITEMSETS
T1	A, B
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

Given: Minimum Support= 2, Minimum Confidence= 50%

Solution:

Step-1: Calculating C1 and L1: In the first step, we will create a table that contains support count (The frequency of each itemset individually in the dataset) of each itemset in the given dataset. This table is called the Candidate set or C1**.**

Itemset	Support_Count
A	6
B	7
C	5
D	2
E	1

Now, we will take out all the itemsets that have the greater support count that the Minimum Support (2). It will give us the table for the **frequent itemset L1.** Since all the itemsets have greater or equal support count than the minimum support, except the E, so E itemset will be removed.

Itemset	Support_Count
A	6
B	7
C	5
D	2

Step-2: Candidate Generation C2, and L2:

In this step, we will generate C2 with the help of L1. In C2, we will create the pair of the itemsets of L1 in the form of subsets.After creating the subsets, we will again find the support count from the main transaction table of datasets, i.e., how many times these pairs have occurred together in the given dataset. So, we will get the below table for C2

Itemset	Support_Count
{A, B}	4
{A,C}	4
{A, D}	1
{B, C}	4
{B, D}	2
{C, D}	0

Again, we need to compare the C2 Support count with the minimum support count, and after comparing, the itemset with less support count will be eliminated from the table C2. It will give us the below table for L2

Itemset	Support_Count
{A, B}	4
{A, C}	4
{B, C}	4
{B, D}	2

A, B, C, D

Step-3: Candidate generation C3, and L3:

For C3, we will repeat the same two processes, but now we will form the C3 table with subsets of three itemsets together, and will calculate the support count from the dataset. It will give the below table:

Itemset	Support_Count
{A, B, C}	2
{B, C, D}	1
{A, C, D}	0
{A, B, D}	0

Now we will create the L3 table. As we can see from the above C3 table, there is only one combination of itemset that has support count equal to the minimum support count. So, the L3 will have only one combination, i.e., **{A, B, C}.**

Step-4: Finding the association rules for the subsets:

To generate the association rules, first, we will create a new table with the possible rules from the occurred combination {A, B.C}. For all the rules, we will calculate the Confidence using formula **sup(A ^B)/A.** After calculating the confidence value for all rules, we will exclude the rules that have less confidence than the minimum threshold(50%).Consider the below table:

Rules	Support	Confidence
A ^B → C	2	Sup{(A ^B) ^C}/sup(A ^B)= 2/4=0.5=50%
B^C → A	2	Sup{(B^C) ^A}/sup(B ^C)= 2/4=0.5=50%
A^C → B	2	Sup{(A ^C) ^B}/sup(A ^C)= 2/4=0.5=50%
C→ A ^B	2	Sup{(C^(A ^B)}/sup(C)= 2/5=0.4=40%
A→ B^C	2	Sup{(A^(B ^C)}/sup(A)= 2/6=0.33=33.33%
B→ B^C	2	Sup{(B^(B ^C)}/sup(B)= 2/7=0.28=28%

As the given threshold or minimum confidence is 50%, so the first three rules **A ^B → C, B^C → A, and A^C → B** can be considered as the strong association rules for the given problem.

Advantages of Apriori Algorithm

1. This is easy to understand algorithm
2. The join and prune steps of the algorithm can be easily implemented on large datasets.

Disadvantages of Apriori Algorithm

1. The apriori algorithm works slow compared to other algorithms.
2. The overall performance can be reduced as it scans the database for multiple times.
3. The time complexity and space complexity of the apriori algorithm is O(2D), which is very high. Here D represents the horizontal width present in the database.

Conclusion: Thus we have studied Aprori algorithm & Association Rule.

Code

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None)

transactions = []

for i in range(0, 7501):

  transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])

from apyori import apriori

rules = apriori(transactions = transactions, min_support = 0.003, min_confidence = 0.2, min_lift = 3, min_length = 2, max_length = 2)

results = list(rules)

def inspect(results):

    lhs         = [tuple(result[2][0][0])[0] for result in results]

    rhs         = [tuple(result[2][0][1])[0] for result in results]

    supports    = [result[1] for result in results]

    confidences = [result[2][0][2] for result in results]

    lifts       = [result[2][0][3] for result in results]

    return list(zip(lhs, rhs, supports, confidences, lifts))

resultsinDataFrame = pd.DataFrame(inspect(results), columns = ['Left Hand Side', 'Right Hand Side', 'Support', 'Confidence', 'Lift'])

resultsinDataFrame

resultsinDataFrame.nlargest(n = 10, columns = 'Lift')
```

Output

Displaying the results sorted by descending lifts

```
In [0]: resultsinDataFrame.nlargest(n = 10, columns = 'Lift')
```

Out[9]:

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
3	fromage blanc	honey	0.003333	0.245098	5.164271
0	light cream	chicken	0.004533	0.290598	4.843951
2	pasta	escalope	0.005866	0.372881	4.700812
8	pasta	shrimp	0.005066	0.322034	4.506672
7	whole wheat pasta	olive oil	0.007999	0.271493	4.122410
5	tomato sauce	ground beef	0.005333	0.377358	3.840659
1	mushroom cream sauce	escalope	0.005733	0.300699	3.790833
4	herb & pepper	ground beef	0.015998	0.323450	3.291994
6	light cream	olive oil	0.003200	0.205128	3.114710

ASSIGNMENT 7

Title: **Assignment on Multilayer Neural Network Model**

Aim:

a. Load the dataset in the program. Define the ANN Model with Keras. Define at least two hidden layers. Specify the ReLU function as activation function for the hidden layer and Sigmoid for the output layer.
b. Compile the model with necessary parameters. Set the number of epochs and batch size and fit the model.
c. Evaluate the performance of the model for different values of epochs and batch sizes.
d. Evaluate model performance using different activation functions Visualize the model using ANN Visualizer

Download the dataset of National Institute of Diabetes and Digestive and Kidney Diseases from below link :
Data Set: https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv

The dataset is has total 9 attributes where the last attribute is "Class attribute" having values 0 and 1. (1="Positive for Diabetes", 0="Negative")

Software Requirement: Spyder (64 bit)

Theory:

Artificial neural network: Artificial neural networks, usually simply called neural networks, are computing systems inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain.

Elements of a Neural Network :-
Input Layer :- This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.
Hidden Layer :- Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.
Output Layer :- This layer bring up the information learned by the network to the outer world.

Activation Functions:

Activation function defines the output of input or set of inputs or in other terms defines node of the output of node that is given in inputs. They basically decide to activate or deactivate neurons to get the desired output. It also performs a nonlinear transformation on the input to get better results on a complex **neural network**. Activation function also helps to normalize the output of any input in the range between **1 to -1**. Activation function must be efficient and it should reduce the computation time because the neural network sometimes trained on millions of data points. Activation function basically decides in any neural network that given input or receiving information is relevant or it is irrelevant. Let's take an example to understand better what is a neuron and how activation function bounds the output value to some limit. The neuron is basically is a weighted average of input, then this sum is passed through an activation function to get an output.

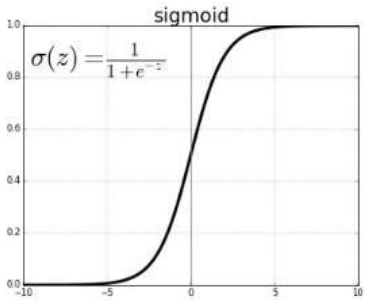
$$\mathbf{Y = \sum (weights*input + bias)}$$

Here Y can be anything for a neuron between range **-infinity to +infinity**. So, we have to bound our output to get the desired prediction or generalized results.

$$\mathbf{Y = Activation\ function(\sum (weights*input + bias))}$$

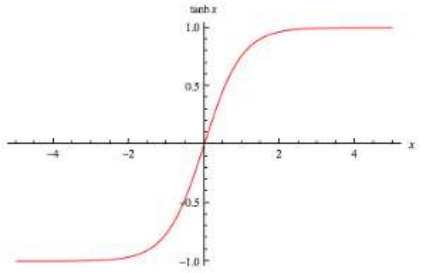
So, we pass that neuron to activation function to bound output values.

1. **Sigmoid Function:** The biggest advantage that it has over other steps and linear functions is its non-linearity. The function ranges from 0 to 1 having an S shape. Also known by the name of the logistic or squashing function in some literature. The sigmoid function is used in output layers of the DNN and is used for probability-based output



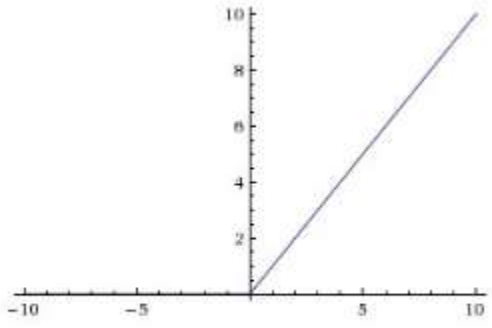
f(x)=1/(1+e^(-x)

2. **TanH Function:** The hyperbolic tangent function is a zero-centered function and its range lies between -1 to 1. As this function is zero centered, this makes it easier to model inputs that have strongly negative, neutral, and strongly positive values. It is advised to use tanh function instead of sigmoid function if your output is other than 0 and 1. The tanh functions have been used mostly in RNN for natural language processing and speech recognition tasks.



f(x)=(e^x-e^(-x))/(e^x+e^(-x)

3. **Rectified Linear Unit(ReLU):** ReLU has been the most widely used activation function for DL applications with state-of-the-art results It provides the upper hand in performance and generalization compared to the Sigmoid and Tanh activation functions. Along with the overall speed of computation enhanced, ReLU provides faster computation since it does not compute exponentials and divisions. It easily overfits compared to the sigmoid function and is one of the main limitations. Some techniques like dropout are used to reduce the overfitting



f(x)=x if x≥0 and 0 if x<0

Deep Learning Libraries:

1. **TensorFlow** was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization. The system is designed to facilitate research in machine learning, and to make it quick and easy to transition from research prototype to production system..
2. **Keras** is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano..

3. **Caffe** is a deep learning framework made with expression, speed, and modularity in mind. It is developed by the Berkeley Vision and Learning Center (BVLC) and community contributors..
4. **Microsoft Cognitive Toolkit (Previously CNTK)** is a unified deep learning toolkit that describes neural networks as a series of computational steps via a directed graph. **PyTorch**, Tensors and Dynamic neural networks in Python with strong GPU acceleration. **Apache MXnet** is a deep learning framework designed for both efficiency and flexibility. It allows you to mix symbolic and imperative programming to maximize efficiency and productivity.
5. **DeepLearning4J** is part of the Skymind Intelligence Layer, along with ND4J, DataVec, Arbiter and RL4J. It is an Apache 2.0-licensed, open-source, distributed neural net library written in Java and Scala..
6. **Theano** allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. However, in September 2017, Theano announced that any further major developments would cease after the 1.0 release. Don't let this put you off though, it is still an extremely powerful library that you can carry out deep learning research with it at anytime.

7. Torch is the main package in Torch7 where data structures for multi-dimensional tensors and mathematical operations over these are defined. Additionally, it provides many utilities for accessing files, serializing objects of arbitrary types and other useful utilities.
8. Caffe2 is a lightweight, modular, and scalable deep learning framework. Building on the original Caffe, Caffe2 is designed with expression, speed, and modularity in mind.

Batch:

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

Epoch:

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm.

Conclusion: Thus we have studied multilayer neural network model.

Code

```
import numpy as np

import pandas as pd

import tensorflow as tf

dataset = pd.read_csv('pima-indians-diabetes.csv')

dataset

dataset.info()

dataset.isnull().any()

x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

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 = 10)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)

x_test = sc.transform(x_test)

ann = tf.keras.models.Sequential()

ann.add(tf.keras.layers.Dense(units = 12, activation = 'relu'))

ann.add(tf.keras.layers.Dense(units = 12, activation = 'tanh'))

ann.add(tf.keras.layers.Dense(units = 12, activation = 'tanh')
```

```
ann.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))


ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

ann.fit(x_train, y_train, batch_size = 32, epochs = 100)

y_pred = ann.predict(x_test)

y_pred = (y_pred > 0.5)

np.concatenate((y_pred.reshape(len(y_pred),1),y_test.reshape(len(y_test),1)),1)

from sklearn.metrics import confusion_matrix, accuracy_score

cm = confusion_matrix(y_test, y_pred)

print(cm)

print(f'Accuracy Score = {round(accuracy_score(y_test, y_pred)*100,2)}%')
```

Output

```
In [11]: ann = tf.keras.models.Sequential()

In [12]: ann.add(tf.keras.layers.Dense(units = 8, activation = 'relu'))

In [13]: ann.add(tf.keras.layers.Dense(units = 8, activation = 'tanh'))

In [14]: ann.add(tf.keras.layers.Dense(units = 8, activation = 'tanh'))

In [15]: ann.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

In [16]: ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
         #  updates weights^
```

```
In [21]: from sklearn.metrics import confusion_matrix, accuracy_score
         cm = confusion_matrix(y_test, y_pred)
         print(cm)
         print(f'Accuracy Score = {round(accuracy_score(y_test, y_pred)*100,2)}%')

[[92  9]
 [23 30]]
Accuracy Score = 79.22%
```

ASSIGNMENT 8

Content Beyond Syllabus

Title: **Assignment of exploring Machine learning libraries.**

Aim: Demonstrate multiple methods of Machine Learning libraries like NumPy & Pandas. Perform multiple operations on given dataset. Below is the link of dataset
https://olympus.greatlearning.in/courses/10899/files/1753546?module_item_id=903335

Theory:

➢ NumPy:

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Operations using NumPy:

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

Methods of Numpy:

1. **np.array:** This method useful for creating one & multidimensional array.

```
import numpy as np
n1=np.array([10,20,30,40])
n1
```

```
import numpy as np
n2=np.array([[10,20,30,40],[40,30,20,10]])
n2
```

2. **np.zeros:** Returns a new array of specified size, filled with zeros.

```
import numpy as np
n1=np.zeros((1,2))
n1
```

```
import numpy as np
n1=np.zeros((5,5))
n1
```

3. **np.arrange:** Return records in given order.

```
import numpy as np
n1=np.arange(10,20) n1
```

4. **np.full:** Return full array according given dimensions.

```
 import numpy as np
n1=np.full((2,2),10)
n1
```

5. **np.random.randint:** Return random values in given range.

```
import numpy as np
n1=np.random.randint(1,100,5)
n1
```

6. **n1.shape:** Return number of rows & columns in given array .

```
 n1.shape
```

7. **np.sum:** Return sum of the array.

```
import numpy as np
n1=np.array([10,20])
n2=np.array([30,40])
np.sum([n1,n2])
```

8. **np.equal:** Check similarities of values in given array.

```
import numpy as np
n1=np.array([10,20,30])
n2=np.array([10,30,20])
 np.equal(n1,n2)
```

9. **np.vstack/ np.hstack/np.column_stack**: Return multiple arrays in vertical, horizontal & column format.

```
import numpy as np
n1=np.array([10,20,30])
n2=np.array([40,50,60])

np.vstack((n1,n2))
```

```
import numpy as np
n1=np.array([10,20,30])
n2=np.array([40,50,60])

np.hstack((n1,n2))
```

```
import numpy as np
n1=np.array([10,20,30])
n2=np.array([40,50,60])

np.column_stack((n1,n2)
)
```

10. NumPy Manipulations:

```
#Division:
import numpy as np
n1=np.array([10,20,30])
n1=n1/2
n1
```

```
#Intersection:
import numpy as np
n1=np.array([10,20,30,4
0,50,60])
```

```
#Difference:
import numpy as np
n1=np.array([10,20,30,40,
50,60])
```

```
n2=np.array([50,60,70,80,90])
np.intersect1d(n1,n2)
```

```
n2=np.array([50,60,70,80,90])
np.setdiff1d(n1,n2)
```

➢ PANDAS:

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

A panda allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant.

Methods of Pandas:

1. **pd.Series:** A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type. With the index argument, you can name your own labels.

```
import pandas as pd
s1=pd.Series([1,2,3,4,5])
s1
```

```
import pandas as pd
s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
s1
```

```
import pandas as pd
pd.Series({'a':10,'b':20,'c':30})
```

2. **pd.DataFrame:** A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

```
import pandas as pd
pd.DataFrame({"Name":['Bob','Sam','Anne'],"Marks":[76,25,92]})
```

3. **pd.read_csv:** A simple way to store big data sets is to use CSV files (comma separated files). CSV files contain plain text and are a well know format that can be read by everyone including Pandas.

```
import pandas as pd
iris=pd.read_csv('iris.csv')
```

4. **head():** Returns the headers and a specified number of rows, starting from the top.

```
iris.head()
```

5. **tail():** Returns the headers and a specified number of rows, starting from the bottom.

```
iris.tail()
```

6. **shape():** Return number of rows & colums count.

```
iris.shape
```

7. **describe():** Return more information about the data set

```
iris.describe()
```

8. **min():** Return minimum value in each columns.

```
iris.min()
```

9. **max():** Return maximum value in each columns.

```
iris.max()
```

10. **iloc[]:** Return one or more specified row(s)

```
iris.iloc[0:3,0:2]
```

Conclusion: Thus we have studied different methods in machine learning libraries.

Output:

```
[8] import numpy as np

    n1=np.array([10,20,30,40])
    n1

    array([10, 20, 30, 40])
```

```
[9] import numpy as np

    n2=np.array([[10,20,30,40],[40,30,20,10]])
    n2

    array([[10, 20, 30, 40],
           [40, 30, 20, 10]])
```

```
[10] type(n2)

     numpy.ndarray
```

```
[11] n1[0]

     10
```

```
[13] import numpy as np
     n1=np.zeros((1,2))
     n1

     array([[0., 0.]])
```

```
[14] import numpy as np
     n1=np.zeros((5,5))
     n1

     array([[0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.]])
```

```
[15] import numpy as np
     n1=np.arange(10,20)
     n1

     array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
[16] import numpy as np
     n1=np.full((2,2),10)
     n1

     array([[10, 10],
            [10, 10]])
```

```
[18] import numpy as np
     n1=np.random.randint(1,100,5)
     n1

     array([36, 87,  1, 33, 11])
```

```
[19] n1.shape

     (5,)
```

```
[20] import numpy as np
     n1=np.array([10,20,30])
     n2=np.array([40,50,60])

     np.vstack((n1,n2))

     array([[10, 20, 30],
            [40, 50, 60]])
```

```
[21] import numpy as np
     n1=np.array([10,20,30])
     n2=np.array([40,50,60])

     np.hstack((n1,n2))

     array([10, 20, 30, 40, 50, 60])
```

```
[22] import numpy as np
     n1=np.array([10,20,30])
     n2=np.array([40,50,60])

     np.column_stack((n1,n2))

     array([[10, 40],
            [20, 50],
            [30, 60]])
```

```
[23] import numpy as np
     n1=np.array([10,20,30])
     n1=n1/2
     n1

     array([ 5., 10., 15.])
```

```
import numpy as np
n1=np.array([10,20,30,40,50,60])
n2=np.array([50,60,70,80,90])
np.intersect1d(n1,n2)

array([50, 60])
```

```
[25] np.setdiff1d(n1,n2)

     array([10, 20, 30, 40])
```

```
[ ] n1.shape
```

```
(5,)
```

```
[2] import numpy as np
    n1=np.array([10,20])
    n2=np.array([30,40])

    np.sum([n1,n2])
```

```
100
```

```
import numpy as np
n1=np.array([10,20,30])
n2=np.array([10,30,20])

np.equal(n1,n2)
```

```
array([ True, False, False])
```

```
[1] import pandas as pd
    s1=pd.Series([1,2,3,4,5])
    s1
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

```
[2] import pandas as pd
    s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
    s1
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

```
[3] import pandas as pd
    pd.Series({'a':10,'b':20,'c':30})
```

```
a    10
b    20
c    30
dtype: int64
```

```
[4] import pandas as pd

    pd.DataFrame({"Name":['Bob','Sam','Anne'],"Marks":[76,25,92]})
```

	Name	Marks
0	Bob	76
1	Sam	25
2	Anne	92

```
[5] from google.colab import files
    uploaded = files.upload()
```

```
Choose Files iris.csv
• iris.csv(application/vnd.ms-excel) - 3867 bytes, last modified: 11/24/2021 - 100% done
Saving iris.csv to iris.csv
```

```
[8] import pandas as pd

    iris=pd.read_csv('iris.csv')
```

```
[9] iris.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
[10] iris.tail()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

```
[11] iris.shape

(150, 5)
```

```
[12] iris.describe()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
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

```
[13] iris.min()

Sepal.Length       4.3
Sepal.Width          2
Petal.Length         1
Petal.Width        0.1
Species         setosa
dtype: object
```

```
[14] iris.max()

Sepal.Length          7.9
Sepal.Width           4.4
Petal.Length          6.9
Petal.Width           2.5
Species         virginica
dtype: object
```

```
[16] iris.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
iris.iloc[0:3,0:2]
```

	Sepal.Length	Sepal.Width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2