Practical Lab Manual

Machine Learning(410242)

Fourth Year Computer Engineering (2019 Course)

Student Name: _____

Seat No / Roll No:_____ **Class:**_____

Branch: _____

MMANTC
ASPIRE | INSPIRE

DEPARTMENT OF COMPUTER ENGINEERING

Al Jamia Mohammadiyah Education Society's

MAULANA MUKHTAR AHMAD NADVI TECHNICAL CAMPUS

Mansoora Campus, Malegaon (Nashik)

Academic Year 2025-26

Certificate

This is to certify that……..

Roll No: ……Exam Seat No……..…… Class: ……

Program..... of

Institute .....has

Successfully Completed the Term Work / Assignments Satisfactory in the Course

……..……Course Code……..…… for the

Academic Year 2025-26 as per Prescribed in Curriculum **Fourth Year Computer Engineering (2019 Course) Savitribai Phule Pune University.**

Subject Teacher **HOD** **Principal**

Institute Seal

VISION OF THE INSTITUTE

Empowering society through quality education and research for the socio-economic development of the region.

MISSION OF THE INSTITUTE

- Inspire students to achieve excellence in science and engineering.
- Commit to making quality education accessible and affordable to serve society.
- Provide transformative, holistic, and value-based immersive learning experiences for students.
- Transform into an institution of global standards that contributes to nation-building.
- Develop sustainable, cost-effective solutions through innovation and research.
- Promote quality education in rural areas.

VISION OF THE DEPARTMENT

To build strong research and learning environment producing globally competent professionals and innovators who will contribute to the betterment of the society.

MISSION OF THE DEPARTMENT

- To create and sustain an academic environment conducive to the highest level of research and teaching.
- To provide state-of-the-art laboratories which will be up to date with the new developments in the area of computer engineering.
- To organize competitive event, industry interactions and global collaborations in view of providing a nurturing environment for students to prepare for a successful career and the ability to tackle lifelong challenges in global industrial needs.
- To educate students to be socially and ethically responsible citizens in view of national and global development.

DEPARTMENT OF COMPUTER ENGINEERING

Program Outcomes (POs)

Learners are expected to know and be able to

PO1	Engineering knowledge	Apply the knowledge of mathematics, science, Engineering fundamentals, and an Engineering specialization to the solution of complex Engineering problems.
PO2	Problem analysis	Identify, formulate, review research literature and analyze complex Engineeringproblems reaching substantiated conclusions using first principles of mathematics, natural sciences and Engineering sciences.
PO3	Design / Development of Solutions	Design solutionsfor complex Engineeringproblems and design system componentsor processes thatmeet the specifiedneeds with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	Conduct Investigations of Complex Problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	Modern Tool Usage	Create, select, and apply appropriate techniques, resources, and modern Engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	The Engineer and Society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	Environment and sustainability	Understand the impact of the professional Engineering solutions in societal and Environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.
PO9	Individual and Team Work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
P10	Communication Skills	Communicate effectively on complex Engineering activities with the Engineering community and with society at large, such as, being able to comprehendand write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
P11	Project Management and Finance	Demonstrate knowledge and understanding of Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary Environments.
P12	Life-long Learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DEPARTMENT OF COMPUTER ENGINEERING

Program Specific Outcomes (PSOs)

A graduate of the Computer Engineering Program will demonstrate

PSO1

Professional Skills- The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexities.

PSO2

Problem-Solving Skills- The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

PSO3

Successful Career and Entrepreneurship- The ability to employ modern computer languages, environments and platforms in creating innovative career paths to be an entrepreneur and to have a zest for higher studies

SAVITRIBAI PHULE PUNE UNIVERSITY

FOURTH YEAR OF COMPUTER ENGINEERING (2019 COURSE)

410246: Laboratory Practice III

Course Objectives

● Learn effect of data preprocessing on the performance of machine learning algorithms

● Develop in depth understanding for implementation of the regression models.

● Implement and evaluate supervised and unsupervised machine learning algorithms.

● Analyze performance of an algorithm.

● Learn how to implement algorithms that follow algorithm design strategies namely divide and conquer, greedy, dynamic programming, backtracking, branch and bound.

● Understand and explore the working of Blockchain technology and its applications

Course Outcomes

After completion of the course, students will be able to

CO1: Apply preprocessing techniques on datasets.
CO2: Implement and evaluate linear regression and random forest regression models.
CO3: Apply and evaluate classification and clustering techniques.
CO4: Analyze performance of an algorithm.
CO5: Implement an algorithm that follows one of the following algorithm design strategies: divide and conquer, greedy, dynamic programming, backtracking, branch and bound.
CO6: Interpret the basic concepts in Blockchain technology and its applications

410246: Machine Learning	
Any 5 assignments and 1 Mini project are mandatory.	
Group B	
01	Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and random forest regression models. 5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset.
02	Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv
03	Given abank customer, build aneuralnetwork-based classifier that can determine whether they will leave or not in the next 6 months. Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps: 1. Read the dataset. 2. Distinguish thefeature and target set and divide the data set into training and test sets. 3. Normalize the train and test data. 4. Initialize and build themodel. Identify thepointsof improvement and implement the same. 5. Print the accuracy score and confusion matrix (5 points).
04	Implement Gradient Descent Algorithm to find the local minima of a function. For example, find the local minima of the function $y=(x+3)^2$ starting from the point x=2.
05	Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset. Dataset link : https://www.kaggle.com/datasets/abdallamahgoub/diabetes
06	Implement K-Meansclustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method. Dataset link : https://www.kaggle.com/datasets/kyanyoga/sample-sales-data
Mini Project	
07	**Mini Project -** Use the following dataset to analyze ups and downs in the market and predict future stock price returns based on Indian Market data from 2000 to 2020. Dataset Link: https://www.kaggle.com/datasets/sagara9595/stock-data

08	**Mini Project -** Build a machine learning model that predicts the type of people who survived the Titanic shipwreck using passenger data (i.e. name, age, gender, socio-economic class, etc.). Dataset Link: https://www.kaggle.com/competitions/titanic/data
09	**Mini Project -** Develop a application for signature identification by creating your own dataset of your college student

@The CO-PO Mapping Matrix

CO\PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	-	2	-	3	-	-	2	2	2	1	2
CO2	1	-	2	2	3	2	-	2	2	2	1	2
CO3	1	-	2	2	3	2	-	2	2	2	2	2
CO4	1	-	2	-	3	-	-	2	2	2	2	2
CO5	1	-	2	-	3	-	-	2	2	2	2	2
CO6	1	-	2	-	3	-	-	2	2	2	2	2

INDEX

Sr. No.	Title	Date Performance	Date Completion	Marks	Faculty Dated Sign.
01	Predict the price of the Uber ride from a given pickup point to the agreed drop-off location				
02	Classify the email using the binary classification method				
03	Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.				
04	Implement Gradient Descent Algorithm to find the local minima of a function				
05	Implement K-Nearest Neighbors algorithm on diabetes.csv dataset.				
06	Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset.				
07	Mini Project:				

Practical No.1

Title:

Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.

Aim:

Perform following tasks:

1. Pre-process the dataset.
2. Identify outliers.
3. Check the correlation.
4. Implement linear regression and random forest regression models.
5. Evaluate the models and compare their respective scores like R2, RMSE, etc.

Theory:

Uber rides prediction is a Machine Learning model which predict the uber rides that how much uber rides can do an Uber boy per week based on the different types of features. This model works good with a good accuracy (93.47 %). This model trained with a Linear Regression Algorithm with a best accuracy out of other models.

Linear regression is a basic and commonly used type model of predictive analysis. These regression estimates are used to explain the relationship between one dependent variable (Feature) and one or more independent variables (Features).

This model is useful for the safety of the passengers that no uber boy can't do a ride out of limit. If they pick up a passenger out of his limit so company will put a charge on him.

1. Pre-process the Dataset:

Data Pre-processing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

The main agenda for a model to be accurate and precise in predictions is that the algorithm should be able to easily interpret the data's features.

- Getting the dataset: the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the dataset.
- Importing libraries: In order to perform data pre-processing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs.

- Importing datasets: Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory.
- Finding Missing Data: he next step of data pre-processing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model.
- Encoding Categorical Data: Categorical data is data which has some categories such as, in our dataset; there are two categorical variables, Country, and Purchased.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So, it is necessary to encode these categorical variables into numbers.

- Splitting dataset into training and test set: n machine learning data pre-processing, we divide our dataset into a training set and test set. This is one of the crucial steps of data pre-processing as by doing this, we can enhance the performance of our machine learning model.
- Feature scaling: Feature scaling is the final step of data pre-processing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominates the other variable.

2. Identify Outliers:

Identifying outliers is an important step in the data preprocessing phase of machine learning. Outliers are data points that significantly deviate from the rest of the data, and they can have a substantial impact on the performance of some machine learning algorithms. Here are some common methods to identify outliers:

Visualizations:

Box Plots: Box plots can help identify outliers by showing the distribution of a dataset. Points outside the whiskers of the box plot are often considered outliers.

Scatter Plots: Scatter plots can reveal data points that are far away from the bulk of the data.

Summary Statistics:

Z-Score: The Z-score measures how many standard deviations a data point is from the mean. Points with Z-scores above a certain threshold (e.g., 3) are often considered outliers.

IQR (Interquartile Range): This method defines outliers as values that fall below Q1 - 1.5xIQR or above Q3 + 1.5xIQR, where Q1 is the 25th percentile and Q3 is the 75th percentile.

Visualization Tools:

Tools like scatter plot matrices, pair plots, or 3D plots can help in visualizing data and spotting outliers.

Ensemble Methods:

Techniques like bagging or boosting can sometimes help in reducing the influence of outliers.

Reasons for outliers in data

1. Errors during data entry or a faulty measuring device (a faulty sensor may result in extreme readings).
2. Natural occurrence (salaries of junior level employee's vs C-level employees).

Problems caused by outliers:

1. Outliers in the data may causes problems during model fitting (esp. linear models).
2. Outliers may inflate the error metrics which give higher weights to large errors (example, mean squared error, RMSE).

Check the correlation:

Correlation explains how one or more variables are related to each other. These variables can be input data features which have been used to forecast our target variable.

A correlation could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variables' values decrease. Correlation can also be neutral or zero, meaning that the variables are unrelated.

- Positive Correlation: both variables change in the same direction.
- Neutral Correlation: No relationship in the change of the variables.
- Negative Correlation: variables change in opposite directions

Implement linear regression and random forest regression models:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data, and hence the output doesn't depend on one decision tree but on multiple decision trees. In the case of a classification problem, the final output is taken by using the majority

voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is called Aggregation.

Code:

```
!unzip "/content/archive.zip"

import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
df = pd.read_csv('/content/drive/MyDrive/uber.csv')

 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #  Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   Unnamed: 0         200000 non-null   int64
 1   key                200000 non-null   object
 2   fare_amount        200000 non-null   float64
 3   pickup_datetime    200000 non-null   object
 4   pickup_longitude   200000 non-null   float64
 5   pickup_latitude    200000 non-null   float64
 6   dropoff_longitude  199999 non-null   float64
 7   dropoff_latitude   199999 non-null   float64
 8   passenger_count    200000 non-null   int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

```
df.shape
```

(200000, 9)

```
df.head()
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	

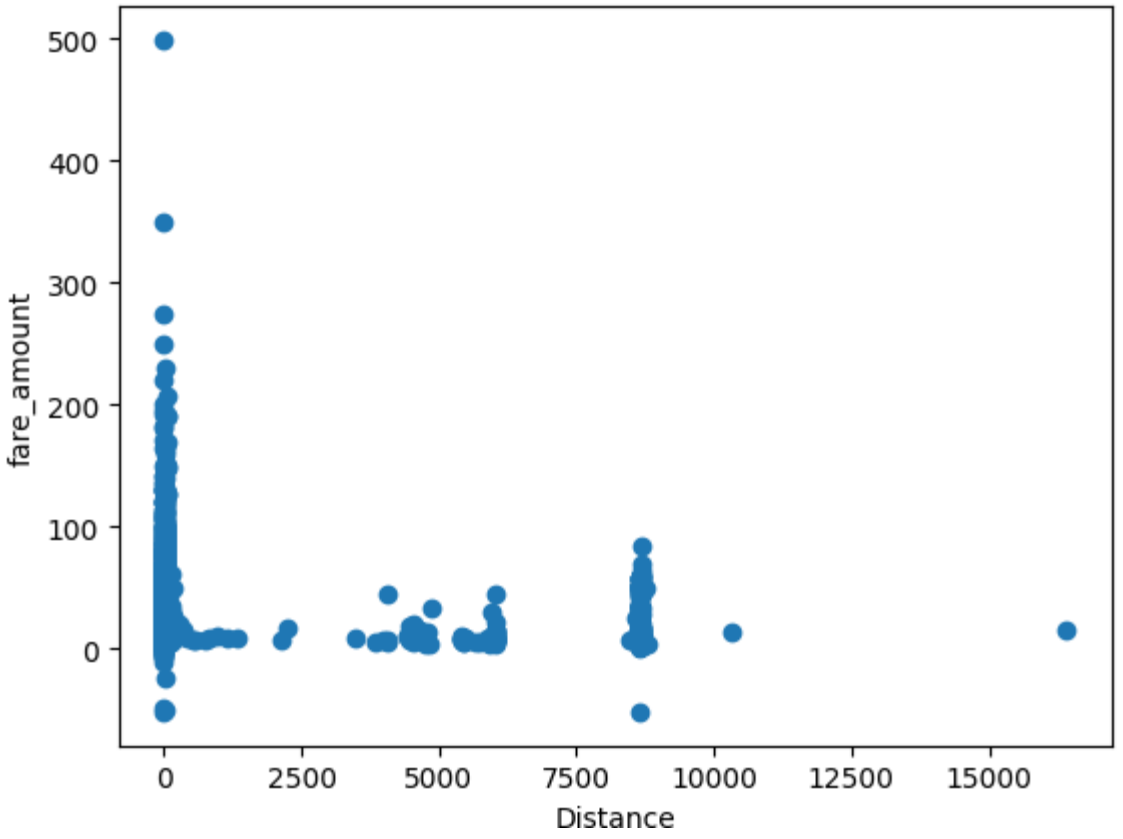
```
df.isnull().sum()
Unnamed: 0 0

key 0

fare_amount        0

pickup_datetime    0

pickup_longitude   0

pickup_latitude    0

dropoff_longitude  1

dropoff_latitude   1

passenger_count    0

dtype: int64
```

```
from numpy.lib.function_base import  diff
def harversine (lon_1, lon_2, lat_1, lat_2):
  lon_1, lat_1, lon_2, lat_2 = map(np.radians, [lon_1, lon_2, lat_1,
lat_2])

  diff_lon = lon_2 - lon_1
  diff_lat = lat_2 - lat_1

  km = 2 * 6371 * np.arcsin(np.sqrt(np.sin(diff_lat/2.0)**2 +
                                    np.cos(lat_1) * np.cos(lat_2) *
np.sin(diff_lon/2.0)**2))

  return km
df['Distance'] =
harversine(df['pickup_longitude'],df['pickup_latitude'],
                                  df['dropoff_longitude'],df['dropoff_lati
tude'])
df.head()
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	

```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

```
Text(0, 0.5, 'fare_amount')
```




```
df.drop(df[df['Distance'] > 60].index, inplace = True)
df.drop(df[df['fare_amount'] == 0].index, inplace = True)
df.drop(df[df['fare_amount'] == 0].index, inplace = True)
df.drop(df[df['fare_amount'] < 0].index, inplace = True)
df.shape
```

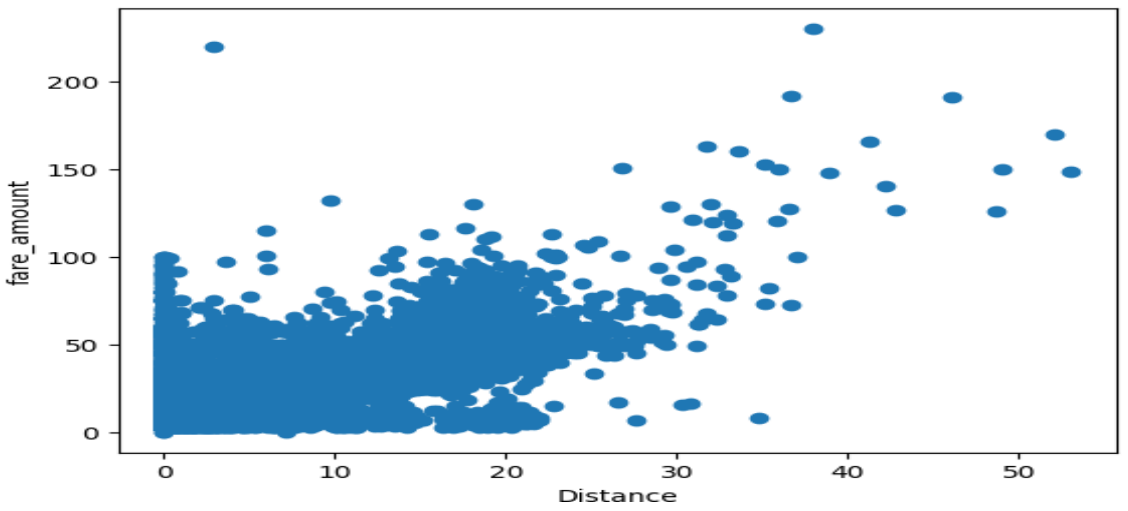
```
(199447, 10)
```

```
df.drop(df[(df['fare_amount']>100) & (df['Distance']<1)].index, inplace
= True )
df.drop(df[(df['fare_amount']<100) & (df['Distance']>100)].index,
inplace = True )
df.shape
```

```
(199447, 10)
```

```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

```
Text(0, 0.5, 'fare_amount')
```




```
df2 = pd.DataFrame().assign(fare=df['fare_amount'],
Distance=df['Distance'])
df2.info()
```

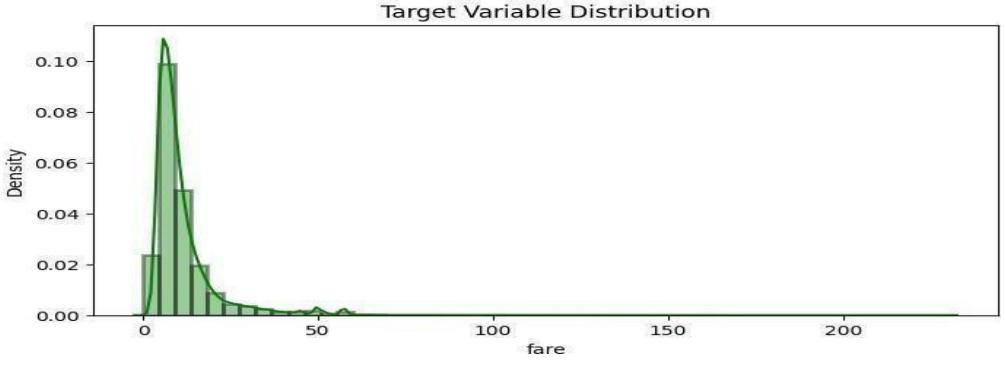
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 199447 entries, 0 to
199999 Data columns (total 2 columns):
 # Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0  fare       199447 non-null  float64
 1  Distance 199447 non-null float64
dtypes: float64(2)
memory usage: 4.6 MB
```

```
df2.shape
```

(199447, 2)

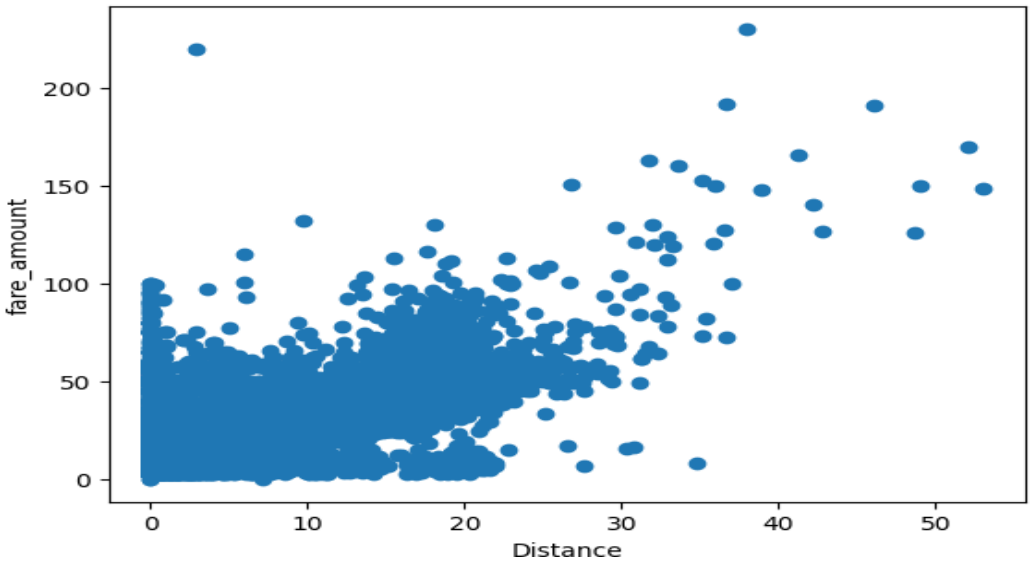
```
plt.figure(figsize=[8,4])
sns.distplot(df2['fare'], color='g',hist_kws=dict(edgecolor="black",
linewidth=2,))
plt.title('Target Variable Distribution')
plt.show()
```

```
<ipython-input-22-7306073c8fc5>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df2['fare'], color='g',hist_kws=dict(edgecolor="black", linewidth=2,))
```



```
plt.scatter(df['Distance'], df['fare_amount'])
plt.xlabel("Distance")
plt.ylabel("fare_amount")
```

```
Text(0, 0.5, 'fare_amount')
```




```
X = df2['fare']
y = df2['Distance']
X = df2['Distance'].values.reshape(-1, 1)
y = df2['fare'].values.reshape(-1, 1)
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
y_std = std.fit_transform(y)
x_std = std.fit_transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_std, y_std,
test_size=0.2, random_state=0)

from sklearn.linear_model import LinearRegression
l_reg = LinearRegression()
l_reg.fit(X_train, y_train)

print("Training set score: {:.2f}".format(l_reg.score(X_train,
y_train)))
print("Test set score: {:.7f}".format(l_reg.score(X_test, y_test)))
```

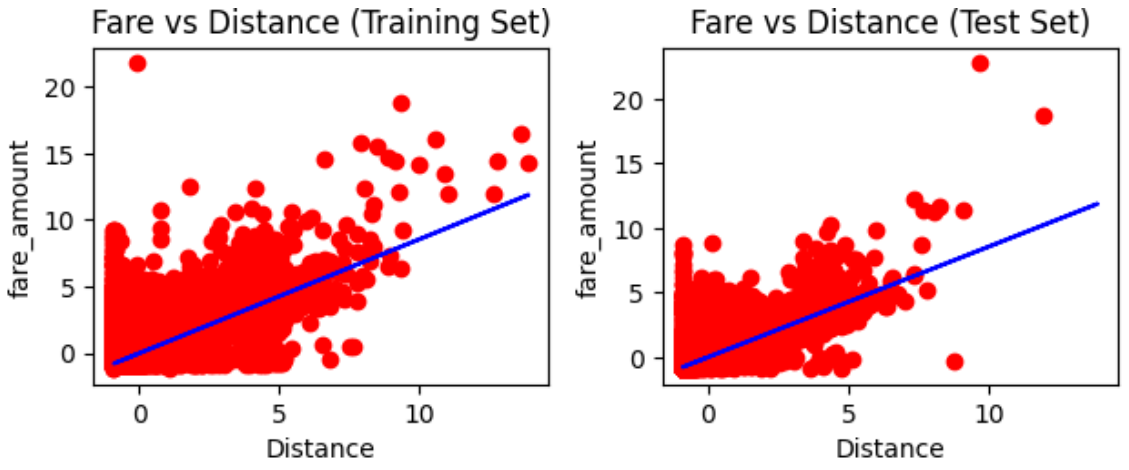
```
Training set score: 0.74
Test set score: 0.7340468
```

```
y_pred = l_reg.predict(X_test)
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test,
y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test,
y_pred))
```

```
print('Root Mean Squared Error:',
np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error:
0.26621298757938955 Mean Squared
Error:  0.2705243510778542
Root Mean Squared Error: 0.5201195546005305
```

```
plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.title("Fare vs Distance (Training Set)")
plt.ylabel("fare_amount")
plt.xlabel("Distance")

plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red')
plt.plot(X_train, l_reg.predict(X_train), color ="blue")
plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")

plt.tight_layout()
plt.show()
```



Conclusion:

In the practical we worked on uber dataset to predict the prize of uber ride from a given pick up point to the agreed drop off location.

Practical No.2

Title:

Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam.

Aim:

Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

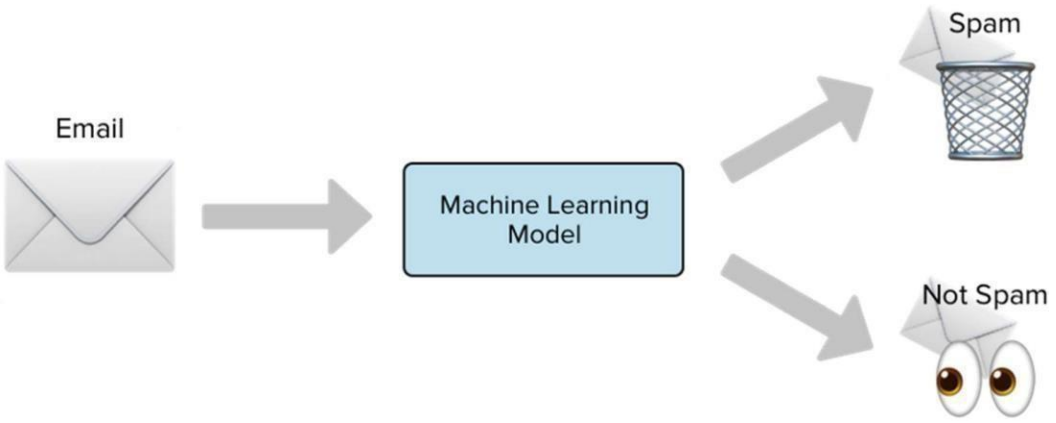
Theory:

Spam Classification

Many email services today provide spam filters that are able to classify emails into spam and non-spam email with high accuracy. SVMs will be used to build a spam filter.

A SVM classifier will be trained to classify whether a given email, xx, is spam (y=1)(y=1) or non-spam (y=0)(y=0). In particular, each email should be converted into a feature vector x∈Rnx∈Rn.

The dataset is based on a subset of the SpamAssassin Public Corpus and only the body of the email will be used (excluding the email headers).



Here in the dataset, you can see there are two features.

1. Label — Ham or Spam
2. Email Text — Actual Email

So basically, our model will recognize the pattern and will predict whether the mail is spam or genuine.

Algorithm used — SVM

SVM:

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n- dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper- plane that differentiates the two classes very well.

Code:

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

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

```
 ! pip install -q kaggle
 from google.colab import files


files.upload()
```

```
Upload widget is only available when the cell has been executed in the cuíient bíowseí session.
Please íeíun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json':
b'{"username":"pawankumarp","key":"a95cda35683153 3d7480492b7f1dd25f"}'}
```

```
! cp /content/kaggle.json kaggle.json ~/.kaggle/
```

```
cp: warning: source file 'kaggle.json' specified more than once
```

```
! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets download balaka18/email-spam-classification-dataset-
csv
```

```
Downloading email-spam-classification-dataset-csv.zip to
  /content 0% 0.00/1.66M [00:00<?, ?B/s]
100% 1.66M/1.66M [00:00<00:00, 174MB/s]
```

```
! unzip '/content/email-spam-classification-dataset-csv.zip'
```

```
Archive: /content/email-spam-classification-dataset-csv.zip
  inflating: emails.csv
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn import preprocessing
df = pd.read csv('emails.csv')
```

```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5172 entries, 0 to 5171
Columns: 3002 entries, Email No. to Prediction
dtypes: int64(3001), object(1)
memory usage: 118.5+ MB
```

```
df.head()
```

	Email No.	the	to	ect	and	for	of	a	you	hou	...	connevey	jay	valued	lay	infrastructure	military	allowing	ff	dry	Prediction
0	Email 1	0	0	1	0	0	0	2	0	0	...	0	0	0	0	0	0	0	0	0	0
1	Email 2	8	13	24	6	6	2	102	1	27	...	0	0	0	0	0	0	0	1	0	0
2	Email 3	0	0	1	0	0	0	8	0	0	...	0	0	0	0	0	0	0	0	0	0
3	Email 4	0	5	22	0	5	1	51	2	10	...	0	0	0	0	0	0	0	0	0	0
4	Email 5	7	6	17	1	5	2	57	0	9	...	0	0	0	0	0	0	0	1	0	0

5 rows × 3002 columns

```
df.dtypes
Email No. object
the       int64
to        int64
ect       int64
and       int64 ...
military int64
allowing  int64
ff        int64
dry       int64
Prediction int64
Length: 3002, dtype: object
```

```
df.drop(columns=['Email No.'], inplace=True)
df.isna().sum()
the       0
to        0
ect       0
and       0
for       0 ..
military  0
allowing  0
ff        0
dry       0
Prediction 0
Length: 3001, dtype: int64
```

```
df.describe()
```

	the	to	ect	and	for	of	a	you	hou	in	...	connevey	jay	valued	lay	infrastructure	military	allowing	
count	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	...	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.000000	5172.00
mean	6.640565	6.188128	5.143852	3.075599	3.124710	2.627030	55.517401	2.466551	2.024362	10.600155	...	0.005027	0.012568	0.010634	0.098028	0.004254	0.006574	0.004060	0.91
std	11.745009	9.534576	14.101142	6.045970	4.680522	6.229845	87.574172	4.314444	6.967878	19.281892	...	0.105788	0.199682	0.116693	0.569532	0.096252	0.138908	0.072145	2.78
min	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	12.000000	0.000000	0.000000	1.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
50%	3.000000	3.000000	1.000000	1.000000	2.000000	1.000000	28.000000	1.000000	0.000000	5.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
75%	8.000000	7.000000	4.000000	3.000000	4.000000	2.000000	62.250000	3.000000	1.000000	12.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.00
max	210.000000	132.000000	344.000000	89.000000	47.000000	77.000000	1898.000000	70.000000	167.000000	223.000000	...	4.000000	7.000000	2.000000	12.000000	3.000000	4.000000	3.000000	114.00

8 rows × 3001 columns

```
X=df.iloc[:, :df.shape[1]-1]
y=df.iloc[:, -1]
X.shape, y.shape
((5172, 3000), (5172,)) X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.15, random_state=8)

models = {
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=2),
    "Linear SVM":LinearSVC(random_state=8, max_iter=900000),
    "Polynomical SVM":SVC(kernel="poly", degree=5, random_state=8)

}

for model_name, model in models.items():
    y_pred=model.fit(X_train, y_train).predict(X_test)
    print(f"Accuracy for {model_name} model \t:
{metrics.accuracy_score(y_test, y_pred)}")
  Accuracy for K-Nearest Neighbors model  :
  0.8878865979381443 Accuracy for Linear SVM model   :
  0.9755154639175257 Accuracy for Polynomical SVM model
                                    : 0.759020618556701
```

Conclusion:

We successfully classified email into normal state (Not spam), and abnormal state (Spam)and used SVM for classification and analysed their performance.

Practical No.3

Title:

Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months. Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc.

Aim:

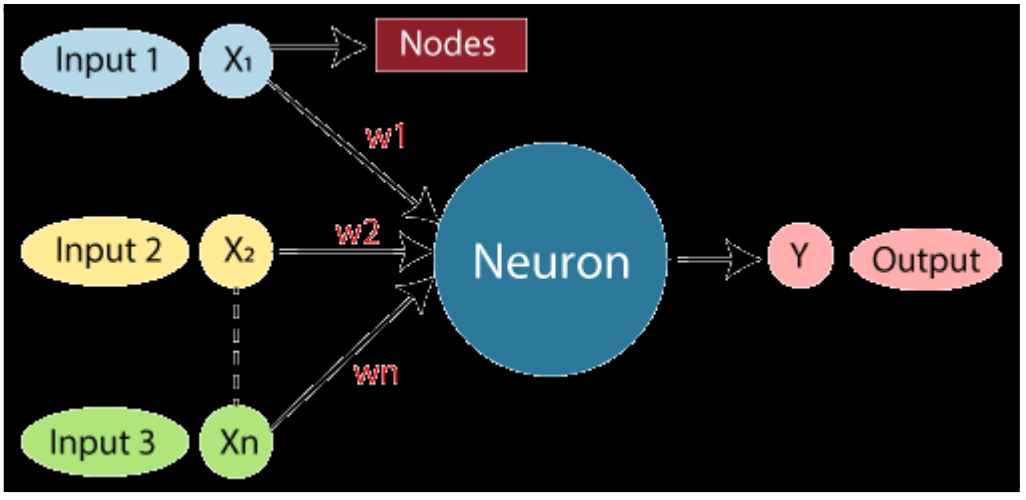
Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix (5 points).

Theory:

Artificial Neural Network:

The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



The typical Artificial Neural Network looks something like the given figure.

Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

An Artificial Neural Network in the field of Artificial intelligence where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should or not. Only those who are red make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

Normalization:

Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

Mathematically, we can calculate normalization with the below formula: Xn = (X - Xminimum) / (Xmaximum - Xminimum)

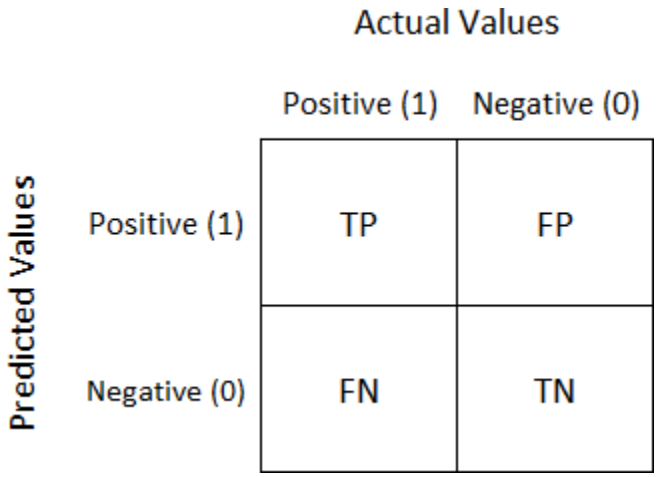
Where,

- Xn = Value of Normalization
- Xmaximum = Maximum value of a feature
- Xminimum = Minimum value of a feature

Confusion Matrix:

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classi ers, the matrix is of 2*2 table, for 3 classes, it is 3*3 table, and so on.
- The matrix is divided into two dimensions, that arepredicted valuesandactual valuesalong with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.
- It looks like the below table:



The above table has the following cases:

- True Negative:Model has given prediction No, and the real or actual value was also No.

- True Positive:The model has predicted yes, and the actual value was also true.

- False Negative:The model has predicted no, but the actual value was Yes, it is also called asType-II error.
- False Positive:The model has predicted Yes, but the actual value was No. It is also called aType-I error.

Need for Confusion Matrix in Machine learning

- It evaluates the performance of the classi cation models, when they make predictions on test data, and tells how good our classi cation model is.
- It not only tells the error made by the classi ers but also the type of errors such as it is either type-I or type-II error.
- With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

CODE:

```
      ! pip install -q kaggle

! mkdir ~/.kaggle

! cp kaggle.json ~/.kaggle/
 ! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets download 'barelydedicated/bank-customer-churn-
modeling'
  Downloading bank-customer-churn-modeling.zip to
    /content 0% 0.00/262k [00:00<?, ?B/s]
  100% 262k/262k [00:00<00:00, 94.9MB/s]

  ! unzip '/content/bank-customer-churn-modeling.zip'
  Archive: /content/bank-customer-churn-modeling.zip
    inflating: Churn_Modelling.csv
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
StandardScaler
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn import metrics
from sklearn import preprocessing
df = pd.read_csv('/content/Churn_Modelling.csv')
df.info()
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to
9999 Data columns (total 14
columns):
 # Column             Non-Null Count  Dtype
---------------------------------------------------------------
 0 RowNumber          10000 non-null  int64
 1 CustomerId         10000 non-null  int64
 2 Surname            10000 non-null  object
 3 CreditScore        10000 non-null  int64
 4 Geography          10000 non-null  object
 5 Gender             10000 non-null  object
 6 Age                10000 non-null  int64
 7 Tenure             10000 non-null  int64
 8 Balance            10000 non-null  float64
 9 NumOfProducts      10000 non-null  int64
 10HasCrCard          10000 non-null  int64
 11IsActiveMember     10000 non-null  int64
 12EstimatedSalary    10000 non-null  float64
 13Exited             10000 non-null int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
df.drop(columns=['RowNumber', 'CustomerId', 'Surname'], inplace=True)

df.isna().sum()
```

```
CreditScore      0
Geography        0
Gender           0
Age              0
Tenure           0
Balance          0
NumOfProducts    0
HasCrCard        0
```

IsActiveMember 0

EstimatedSalary 0

Exited 0

dtype: int64

```
df.describe()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

```
X=df.iloc[:, :df.shape[1]-1].values
y=df.iloc[:, -1].values
X.shape, y.shape
```

((10000, 10), (10000,))

```
print(X[:8,1], '... will now become: ')

label_X_country_encoder = LabelEncoder()
X[:,1] = label_X_country_encoder.fit_transform(X[:,1])
print(X[:8,1])
```

```
['France' 'Spain' 'France' 'France' 'Spain' 'Spain' 'France' 'Germany']
... will now become:
[0 2 0 0 2 2 0 1]
```

```
print(X[:6,2], '... will now become: ')

label_X_gender_encoder = LabelEncoder()
X[:,2] = label_X_gender_encoder.fit_transform(X[:,2])
print(X[:6,2])
```

```
['Female' 'Female' 'Female' 'Female' 'Female' 'Male'] ... will now
become:
[0 0 0 0 0 1]
```

```
transform = ColumnTransformer([("countries", OneHotEncoder(), [1])],
remainder="passthrough") # 1 is the country column
X = transform.fit_transform(X)
X
```

```
array([[1.0, 0.0, 0.0, ..., 1, 1, 101348.88], [0.0, 0.0, 1.0, ..., 0,
1, 112542.58], [1.0, 0.0, 0.0, ..., 1, 0, 113931.57], ..., [1.0, 0.0,
0.0, ..., 0, 1, 42085.58], [0.0, 1.0, 0.0, ..., 1, 0, 92888.52], [1.0,
0.0, 0.0, ..., 1, 0, 38190.78]], dtype=object)
```

```
X = X[:,1:]
X.shape
```

```
(10000, 11)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

sc=StandardScaler()
X_train[:,np.array([2,4,5,6,7,10])] =
sc.fit transform(X train[:,np.array([2,4,5,6,7,10])])
X_test[:,np.array([2,4,5,6,7,10])] =
sc.transform(X_test[:,np.array([2,4,5,6,7,10])])
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

```
array([[-0.5698444 , 1.74309049, 0.16958176, ..., 0.64259497, -
1.03227043, 1.10643166], [ 1.75486502, -0.57369368, -2.30455945, ...,
0.64259497, 0.9687384 , -0.74866447], [-0.5698444 , -0.57369368, -
1.19119591, ..., 0.64259497, -1.03227043, 1.48533467], ..., [-0.5698444
, -0.57369368, 0.9015152 , ..., 0.64259497, -1.03227043, 1.41231994],
[-0.5698444 , 1.74309049, -0.62420521, ..., 0.64259497, 0.9687384 ,
0.84432121], [ 1.75486502, -0.57369368, -0.28401079, ..., 0.64259497, -
1.03227043, 0.32472465]]) from tensorflow.keras.models import
Sequential


# Initializing the ANN
classifier = Sequential()
from tensorflow.keras.layers import Dense

classifier.add(Dense(activation = 'relu', input_dim = 11, units=256,
kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=512,
kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=256,
kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=128,
kernel_initializer='uniform'))
classifier.add(Dense(activation = 'sigmoid', units=1,
kernel_initializer='uniform'))
classifier.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               3072
```

```
 dense_1  (Dense)        (None,         512)                131584

 dense_2  (Dense)        (None,         256)                131328

 dense_3  (Dense)        (None,         128)                32896

 dense_4  (Dense)        (None,         1)                  129

=================================================================
Total params: 299009 (1.14 MB)
Trainable params: 299009 (1.14 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
classifier.fit(
    X_train, y_train,
    validation_data=(X_test,y_test),
    epochs=20,
    batch_size=32
)
```

```
Epoch 1/20
250/250 [==============================] - 5s 10ms/step - loss: 0.4268 - accuracy: 0.8216 - val_loss: 0.3792 - val_accuracy: 0.8390
Epoch 2/20
250/250 [==============================] - 2s 9ms/step - loss: 0.3601 - accuracy: 0.8537 - val_loss: 0.3570 - val_accuracy: 0.8585
Epoch 3/20
250/250 [==============================] - 3s 10ms/step - loss: 0.3505 - accuracy: 0.8572 - val_loss: 0.3460 - val_accuracy: 0.8630
Epoch 4/20
250/250 [==============================] - 4s 16ms/step - loss: 0.3442 - accuracy: 0.8601 - val_loss: 0.3422 - val_accuracy: 0.8640
Epoch 5/20
250/250 [==============================] - 3s 12ms/step - loss: 0.3384 - accuracy: 0.8666 - val_loss: 0.3454 - val_accuracy: 0.8630
Epoch 6/20
250/250 [==============================] - 2s 9ms/step - loss: 0.3340 - accuracy: 0.8633 - val_loss: 0.3418 - val_accuracy: 0.8615
Epoch 7/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3307 - accuracy: 0.8650 - val_loss: 0.3368 - val_accuracy: 0.8615
Epoch 8/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3301 - accuracy: 0.8626 - val_loss: 0.3336 - val_accuracy: 0.8655
Epoch 9/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3233 - accuracy: 0.8691 - val_loss: 0.3425 - val_accuracy: 0.8555
Epoch 10/20
250/250 [==============================] - 3s 12ms/step - loss: 0.3197 - accuracy: 0.8685 - val_loss: 0.3428 - val_accuracy: 0.8535
Epoch 11/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3190 - accuracy: 0.8706 - val_loss: 0.3507 - val_accuracy: 0.8590
Epoch 12/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3151 - accuracy: 0.8706 - val_loss: 0.3350 - val_accuracy: 0.8590
Epoch 13/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3098 - accuracy: 0.8760 - val_loss: 0.3481 - val_accuracy: 0.8520
Epoch 14/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3074 - accuracy: 0.8748 - val_loss: 0.3337 - val_accuracy: 0.8675
Epoch 15/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3028 - accuracy: 0.8756 - val_loss: 0.3391 - val_accuracy: 0.8615
```

```
y_pred = classifier.predict(X_test)
y pred
```

63/63 [==============================] - 0s 5ms/step
array([[0.32362953],
[0.26068547],
[0.15331283],
…,
[0.00363243],
[0.18768916],
[0.15182865]], dtype=float32)

```
y_pred = (y_pred > 0.5)
y_pred
```

```
array([[False], [False], [False], ..., [False], [False], [False]]) from
sklearn.metrics import confusion_matrix,classification_report


cm1 = confusion_matrix(y_test, y_pred)
cm1
array([[1503,  92],
       [ 191, 214]])
```

```
print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.89      0.94      0.91      1595
           1       0.70      0.53      0.60       405

    accuracy                           0.86      2000
   macro avg       0.79      0.74      0.76      2000
weighted avg       0.85      0.86      0.85      2000
```

Conclusion:

In this way we build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

Practical No.5

Title:

Implement K-Nearest Neighbors algorithm on diabetes.csv dataset.

Aim:

Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

Theory:

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Example:

Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

Step-1: Select the number K of the neighbours

Step-2: Calculate the Euclidean distance of K number of neighbours

Step-3: Take the K nearest neighbours as per the calculated Euclidean distance.

Step-4: Among these k neighbours, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbour is maximum.

Step-6: Our model is ready.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

ModelAccuracy:

Model accuracy is a performance metric used in machine learning to evaluate the performance of a classification model. It is defined as the ratio of correctly predicted instances to the total instances in the dataset. Mathematically, it can be expressed as:

Accuracy =Number of Correct Predictions/Total Number of Predictions
Accuracy= Total Number of Predictions/Number of Correct Predictions

For example, if a model correctly predicts 90 out of 100 instances, the accuracy would be 90/100 =0.9 or 90%.

While accuracy is a commonly used metric, it's important to note that it may not always be the most appropriate measure of performance, especially in scenarios where the classes are imbalanced or where misclassifying certain instances is more critical than others.

CODE:

```
 ! pip install -q kaggle
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

```
from google.colab import files


files.upload()
```

```
Upload widget is only available when the cell has been executed in the cuíient bíowseí session.
Please íeíun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json':
b'{"username":"pawankumarp","key":"aaedcf6ca09958745470edb12a44c06f"}'}
```

```
 ! mkdir ~/.kaggle


! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets download abdallamahgoub/diabetes
```

```
Downloading diabetes.zip to
  /content 0% 0.00/8.89k [00:00<?,
  ?B/s]
100% 8.89k/8.89k [00:00<00:00, 23.1MB/s]
```

```
! unzip '/content/diabetes.zip'
```

```
Archive: /content/diabetes.zip
  inflating: diabetes.csv
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
from sklearn import preprocessing
df = pd.read_csv('/content/diabetes.csv')
df.info()
```

```
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 # Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
0    Pregnancies             768  non-null   int64
1    Glucose                 768  non-null   int64
2    BloodPressure           768  non-null   int64
3    SkinThickness           768  non-null   int64
4    Insulin                 768  non-null   int64
5    BMI                     768  non-null   float64
6    Pedigree                768  non-null   float64
7    Age                     768  non-null   int64
8    Outcome                 768  non-null   int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.corr().style.background_gradient(cmap='BuGn')
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
Pedigree	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

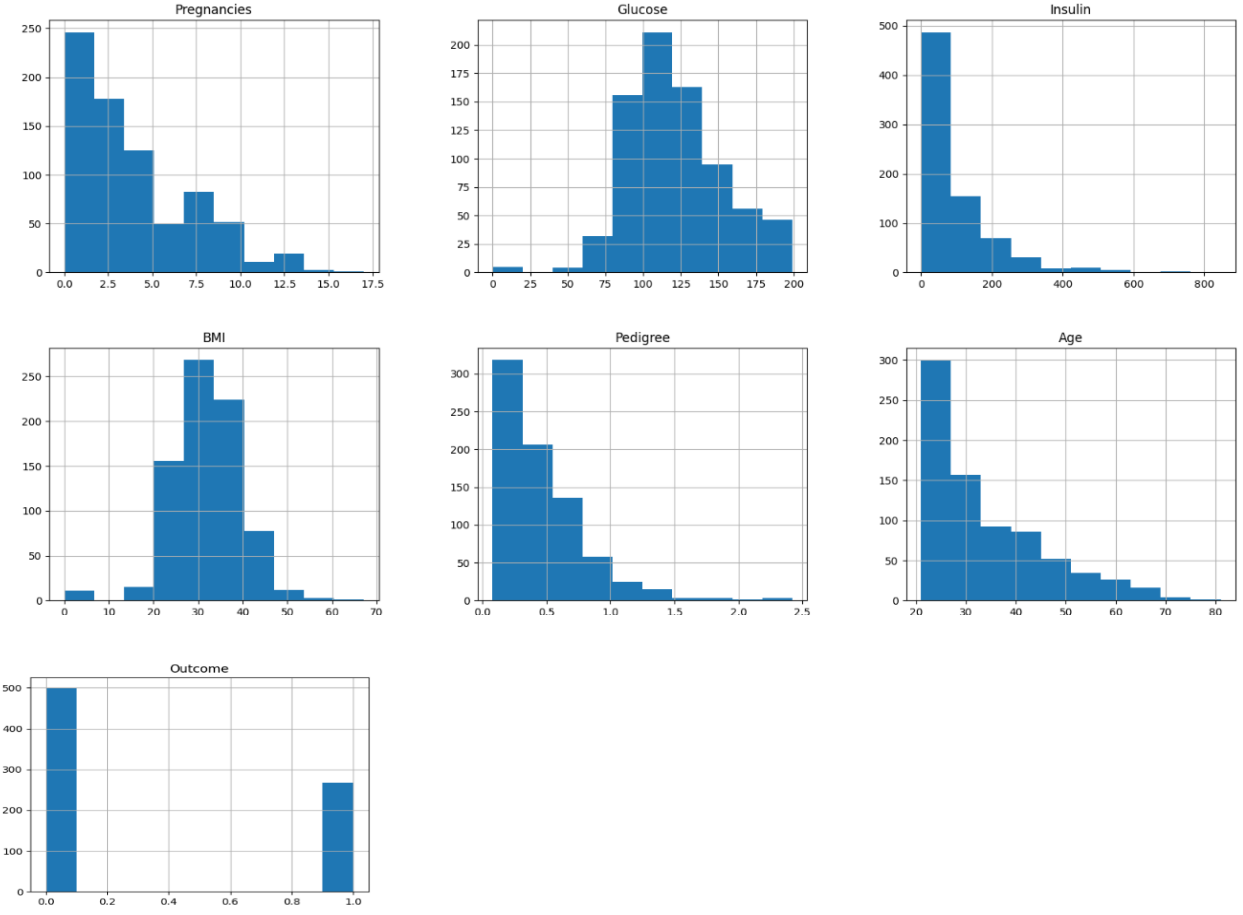
```
df.drop(['BloodPressure', 'SkinThickness'], axis=1, inplace=True)
df.isna().sum()
```

```
Pregnancies        0
Glucose            0
Insulin            0
BMI                0
Pedigree           0
Age                0
Outcome 0 dtype:  int64
```

```
df.describe()
```

	Pregnancies	Glucose	Insulin	BMI	Pedigree	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
hist = df.hist(figsize=(20,16))
```



```
X=df.iloc[:, :df.shape[1]-1]
y=df.iloc[:, -1]
X.shape, y.shape
```

```
((768, 6), (768,))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=8)
scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train) X_test =
scaler.transform(X_test)
def knn(X_train, X_test, y_train, y_test, neighbors, power):
    model = KNeighborsClassifier(n_neighbors=neighbors,
    p=power) # Fit and predict on model
    # Model is trained using the train set and predictions are made
based on the test set. Accuracy scores are calculated for the model.
    y_pred=model.fit(X_train, y_train).predict(X_test)
    print(f"Accuracy for K-Nearest Neighbors model \t:
{accuracy_score(y_test, y_pred)}")

    cm = confusion_matrix(y_test, y_pred)
    print(f'''Confusion matrix :\n
    | Positive Pred iction\t| Negative Prediction
                    +                       +
    Positive Class | True Positive (TP) {cm[0, 0]}\t| False Negative
(FN) {cm[0, 1]}
                    +                       +
    Negative Class | False Positive (FP) {cm[1, 0]}\t| True Negative
(TN) {cm[1, 1]}\n''')
    cr = classification_report(y_test, y_pred)
    print('Classification report : \n', cr)
param_grid = {
    'n_neighbors': range(1, 51),
    'p': range(1, 4)
}
grid = GridSearchCV(estimator=KNeighborsClassifier(),
param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

```
grid.best_estimator_, grid.best_params_, grid.best_score_
```

```
(KNeighborsClassifier(n_neighbors=27), {'n_neighbors': 27, 'p': 2},
0.7719845395175262)
```

```
knn(X_train, X_test, y_train, y_test, grid.best_params_['n_neighbors'],
grid.best_params_['p'])
```

```
Accuracy for K-Nearest Neighbors model : 0.7987012987012987
Confusion matrix :

    | Positive Prediction  | Negative Prediction
                +                       +
    Positive Class | True Positive (TP) 91   | False Negative (FN) 11
                +                       +
    Negative Class | False Positive (FP) 20  | True Negative (TN) 32
```

```
Classification report :
              precision    recall  f1-score   support

0                0.82      0.89        0.85   102
1                0.74      0.62        0.67   52

    accuracy                           0.80   154
   macro avg     0.78      0.75        0.76   154
weighted avg     0.79      0.80        0.79   154
```

CONCLUSION:

We successfully implemented K-Nearest Neighbours algorithm on the diabetes.csv data set and the various computation like confusion, matrix, accuracy, error rate, precision and recall on the data set.

Practical No.6

Title:

Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method.

Theory:

Clustering:

Clustering is an unsupervised machine learning technique. It is the process of division of the dataset into groups in which the members in the same group possess similarities in features. The commonly used clustering algorithms are K-Means clustering, Hierarchical clustering, Density-based clustering, Model-based clustering, etc. In this article, we are going to discuss K-Means clustering in detail.

K-Means Clustering:

It is the simplest and commonly used iterative type unsupervised learning algorithm. In this, we randomly initialize the K number of centroids in the data (the number of k is found using the Elbow method which will be discussed later in this article) and iterates these centroids until no change happens to the position of the centroid. Let's go through the steps involved in K means clustering for a better understanding.

Hierarchical Clustering in Machine Learning:

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabelled datasets into a cluster and also known as hierarchical cluster analysis or HCA. In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree- shaped structure is known as the dendrogram. Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

- **Agglomerative:** Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.

- **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.

Elbow method:

Using the "elbow" or "knee of a curve" as a cutoff point is a common heuristic in mathematical optimization to choose a point where diminishing returns are no longer worth the additional cost. In clustering, this means one should choose a number of clusters so that adding another cluster doesn't give much better modeling of the data.

The intuition is that increasing the number of clusters will naturally improve the fit (explain more of the variation), since there are more parameters (more clusters) to use, but that at some point this is over-fitting, and the elbow reflects this. For example, given data that actually consist of k labeled groups – for example, k points sampled with noise – clustering with more than k clusters will "explain" more of the variation (since it can use smaller, tighter clusters), but this is over-fitting, since it is subdividing the labeled groups into multiple clusters.

The idea is that the first clusters will add much information (explain a lot of variation), since the data actually consist of that many groups (so these clusters are necessary), but once the number of clusters exceeds the actual number of groups in the data, the added information will drop sharply, because it is just subdividing the actual groups. Assuming this happens, there will be a sharp elbow in the graph of explained variation versus clusters: increasing rapidly up to k (under-fitting region), and then increasing slowly after k (over-fitting region).

In practice there may not be a sharp elbow, and as a heuristic method, such an "elbow" cannot always be unambiguously identified.

CODE:

```
 ! pip install -q kaggle
from google.colab import files


files.upload()
```

```
Upload widget is only available when the cell has been executed in the cuíient bíowseí session.
Please íeíun this cell to enable.
```

```
 ! mkdir ~/.kaggle


! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
! kaggle datasets download kyanyoga/sample-sales-data
```

```
Downloading sample-sales-data.zip to
   /content 0% 0.00/77.5k [00:00<?, ?B/s]
 100% 77.5k/77.5k [00:00<00:00, 91.3MB/s]
```

```
! unzip '/content/sample-sales-data.zip'
```

```
Archive: /content/sample-sales-
  data.zip inflating:
  sales_data_sample.csv
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('/content/sales_data_sample.csv', encoding='latin1')
df.head()
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	...	ADDRESSLINE1	ADDRESSLINE2	CITY	STATE	POSTALCODE	COUNTRY	TERRITORY	CONTACTLASTNAME	CONTACTFIRSTNAME
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	Shipped	1	2	2003	...	897 Long Airport Avenue	NaN	NYC	NY	10022	USA	NaN	Yu	Kwai
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	Shipped	2	5	2003	...	59 rue de l'Abbaye	NaN	Reims	NaN	51100	France	EMEA	Henriot	Paul
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	Shipped	3	7	2003	...	27 rue du Colonel Pierre Avia	NaN	Paris	NaN	75508	France	EMEA	Da Cunha	Daniel
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	Shipped	3	8	2003	...	78934 Hillside Dr.	NaN	Pasadena	CA	90003	USA	NaN	Young	Julie
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	Shipped	4	10	2003	...	7734 Strong St.	NaN	San Francisco	CA	NaN	USA	NaN	Brown	Julie

5 rows × 25 columns

```
df.info()
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 # Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0 ORDERNUMBER         2823 non-null   int64
 1 QUANTITYORDERED     2823 non-null   int64
 2 PRICEEACH           2823 non-null   float64
 3 ORDERLINENUMBER     2823 non-null   int64
 4 SALES               2823 non-null   float64
 5 ORDERDATE           2823 non-null   object
 6 STATUS              2823 non-null   object
 7 QTR_ID              2823 non-null   int64
 8 MONTH_ID            2823 non-null   int64
 9 YEAR_ID             2823 non-null   int64
 10PRODUCTLINE         2823 non-null   object
 11MSRP                2823 non-null   int64
 12PRODUCTCODE         2823 non-null   object
 13CUSTOMERNAME        2823 non-null   object
 14PHONE               2823 non-null   object
 15ADDRESSLINE1        2823 non-null   object
 16ADDRESSLINE2        302 non-null    object
 17CITY                2823 non-null   object
 18STATE               1337 non-null   object
 19POSTALCODE          2747 non-null   object
 20COUNTRY             2823 non-null   object
 21TERRITORY           1749 non-null   object
 22CONTACTLASTNAME     2823 non-null   object
 23CONTACTFIRSTNAME    2823 non-null   object
 24DEALSIZE            2823 non-null   object
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB

df.describe()
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	QTR_ID	MONTH_ID	YEAR_ID	MSRP
count	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.000000	2823.00000	2823.000000
mean	10258.725115	35.092809	83.658544	6.466171	3553.889072	2.717676	7.092455	2003.81509	100.715551
std	92.085478	9.741443	20.174277	4.225841	1841.865106	1.203878	3.656633	0.69967	40.187912
min	10100.000000	6.000000	26.880000	1.000000	482.130000	1.000000	1.000000	2003.00000	33.000000
25%	10180.000000	27.000000	68.860000	3.000000	2203.430000	2.000000	4.000000	2003.00000	68.000000
50%	10262.000000	35.000000	95.700000	6.000000	3184.800000	3.000000	8.000000	2004.00000	99.000000
75%	10333.500000	43.000000	100.000000	9.000000	4508.000000	4.000000	11.000000	2004.00000	124.000000
max	10425.000000	97.000000	100.000000	18.000000	14082.800000	4.000000	12.000000	2005.00000	214.000000

```
fig = plt.figure(figsize=(12,10))
sns.heatmap(df.corr(), annot=True, fmt='.2f')
plt.show()
```

```
<ipython-input-15-8a7e33f1ae5c>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value of
numeric_only to silence this warning.
  sns.heatmap(df.corr(), annot=True, fmt='.2f')
```


```
df= df[['PRICEEACH', 'MSRP']]
```

```
df.head()
```

	PRICEEACH	MSRP
0	95.70	95
1	81.35	95
2	94.74	95
3	83.26	95
4	100.00	95

```
df.isna().any()
```

```
PRICEEACH False MSRP False dtype: bool
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
PRICEEACH	2823.0	83.658544	20.174277	26.88	68.86	95.7	100.0	100.0
MSRP	2823.0	100.715551	40.187912	33.00	68.00	99.0	124.0	214.0

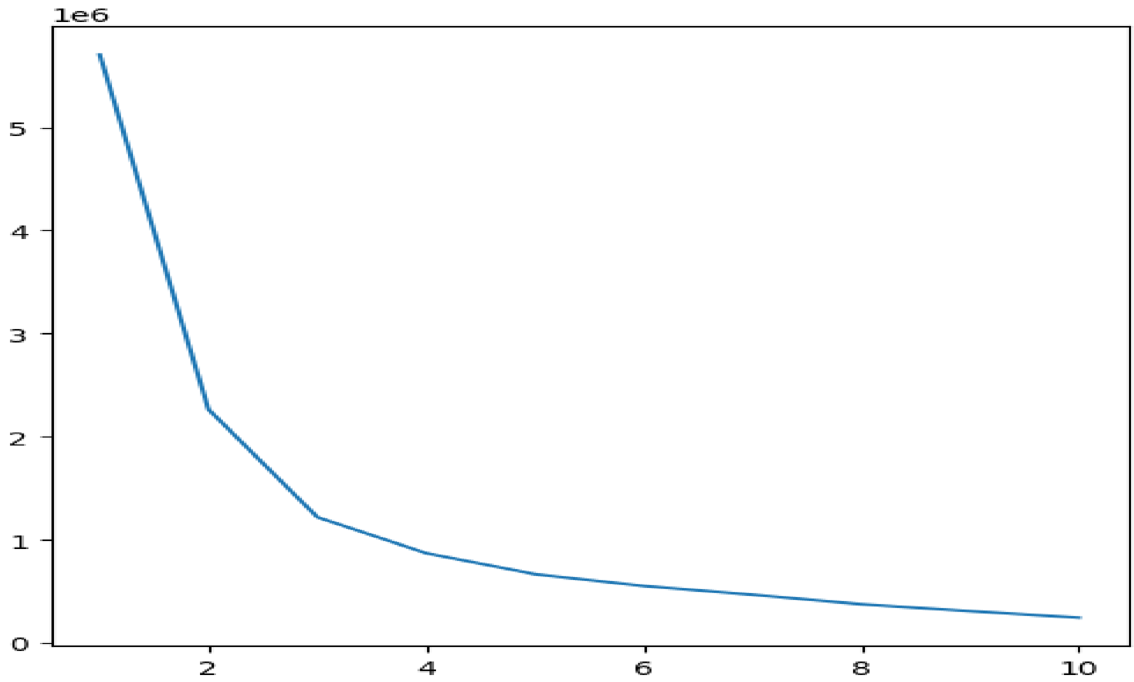
```
df.shape
```

(2823, 2)

```
from sklearn.cluster import KMeans

inertia = []

for i in range(1, 11):
    clusters = KMeans(n_clusters=i, init='k-means++', random_state=42)
    clusters.fit(df)
    inertia.append(clusters.inertia_)

plt.figure(figsize=(6, 6))
sns.lineplot(x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], y = inertia)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

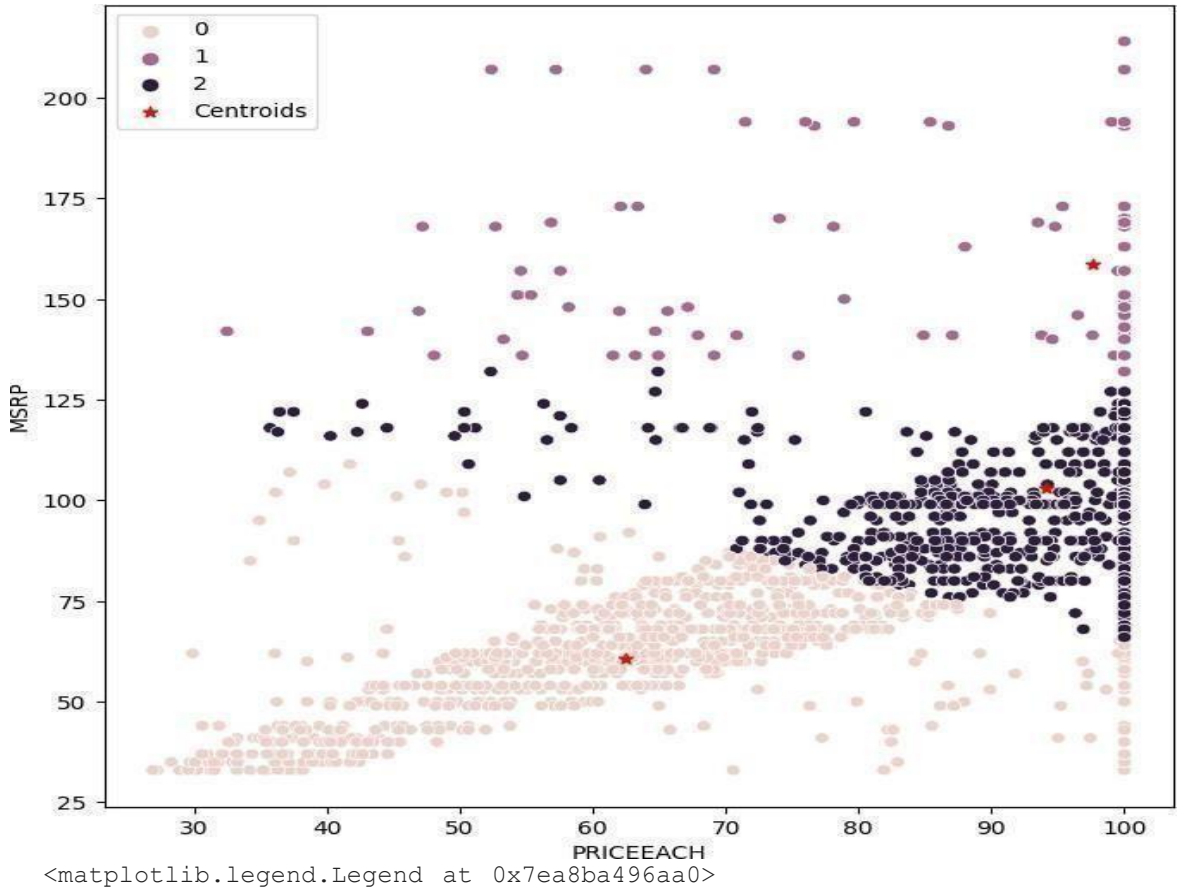
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the
warning
  warnings.warn(
<Axes: >
```




```
kmeans = KMeans(n_clusters = 3, random_state = 42)
y_kmeans = kmeans.fit_predict(df)
y_kmeans
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The
default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to
suppress the warning
  warnings.warn(
array([2, 2, 2, ..., 0, 0, 0], dtype=int32)
```

```
plt.figure(figsize=(8,8))
sns.scatterplot(x=df['PRICEEACH'], y=df['MSRP'], hue=y_kmeans)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1], c = 'red', label = 'Centroids', marker='*')
plt.legend()
```




```
<matplotlib.legend.Legend at 0x7ea8ba496aa0>
```

```
kmeans.cluster_centers_
array([[ 62.49548902, 60.71556886], [ 97.59890263, 158.7202473 ],
[
94.03841567, 102.88841567]])
```

CONCLUSION:

We successfully implemented K-means clustering hierarchical clustering on sales datasample dataset. And determined the number of clusters that used elbow clustering.