INDEX

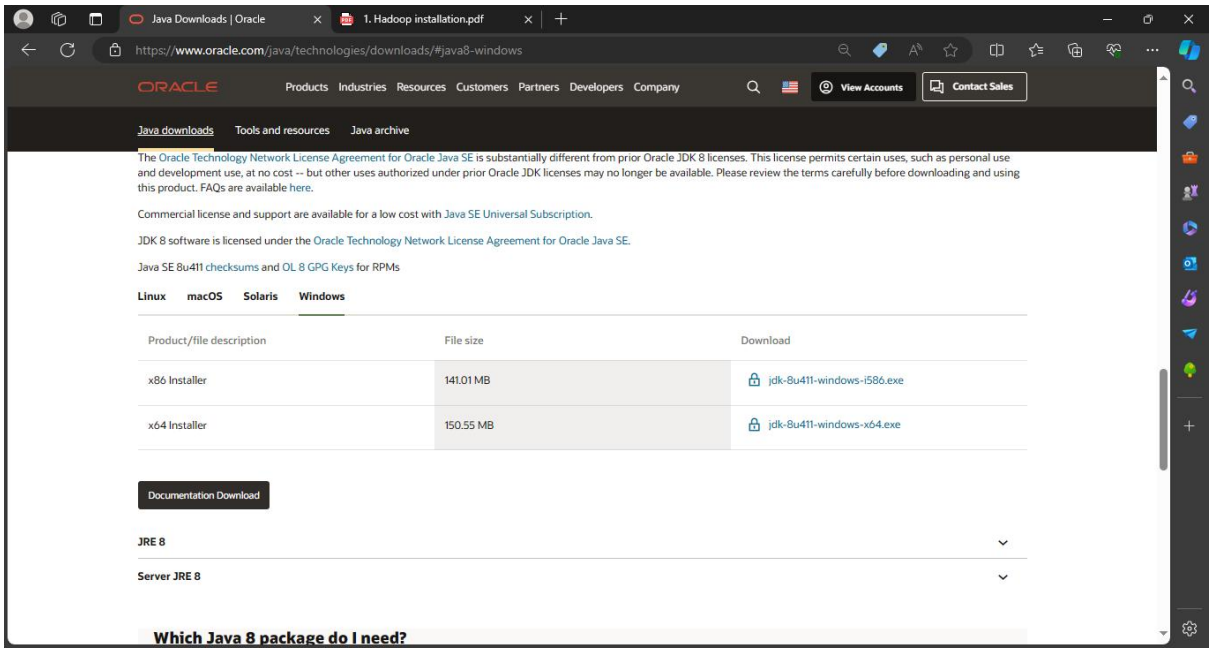
Practical No.	Title of the Practical	Date	Page No.	Sign
1	Install, configure and run Hadoop and HDFS and explore HDFS	01/04/24	1	
2	Implement an application that stores big data in Hbase / MongoDB and manipulate it using R / Python.	06/04/24	17	
3	Implement Regression Model to import a data from web storage. Name the dataset and now do Linear Regression to find out relation between variables. Also check the model is fit or not.	15/04/24	28	
4	Apply Multiple Regression on a dataset having a continuous independent variable.	19/04/24	31	
5	Build a Classification Model (Logistic Regression) a. Install relevant package for classification. b. Choose classifier for classification problem. c. Evaluate the performance of classifier.	22/04/24	36	
6	Build a Clustering Model. (K-Means, Agglomerative) a. Select clustering algorithm for unsupervised learning. b. Plot the cluster data using R/python visualizations.	26/04/24	40	
7	Implement SVM classification technique	29/04/24	44	
8	Implement Decision tree classification technique	04/05/24	48	
9	Naïve Bayes Implementation	11/05/24	51	

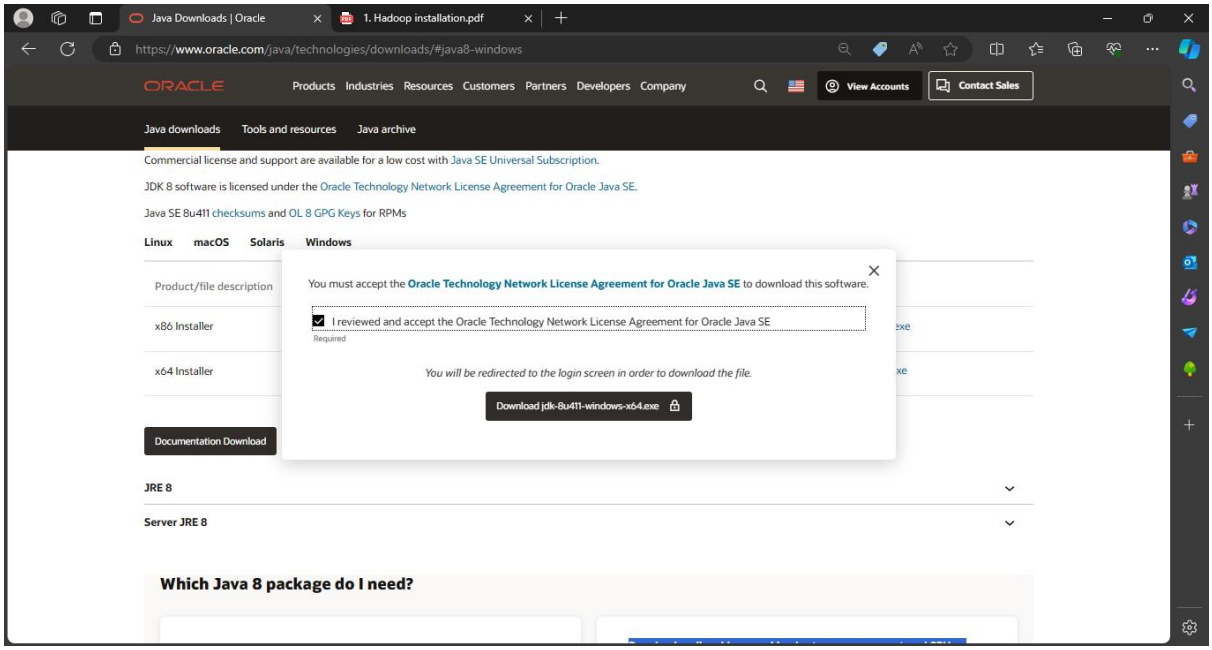
Practical 1

Aim: Install, configure and run Hadoop and HDFS and explore HDFS.

Step 1: Download Java from oracle website

Search for java SE development kit 8 download
https://www.oracle.com/java/technologies/downloads/#java8-windows





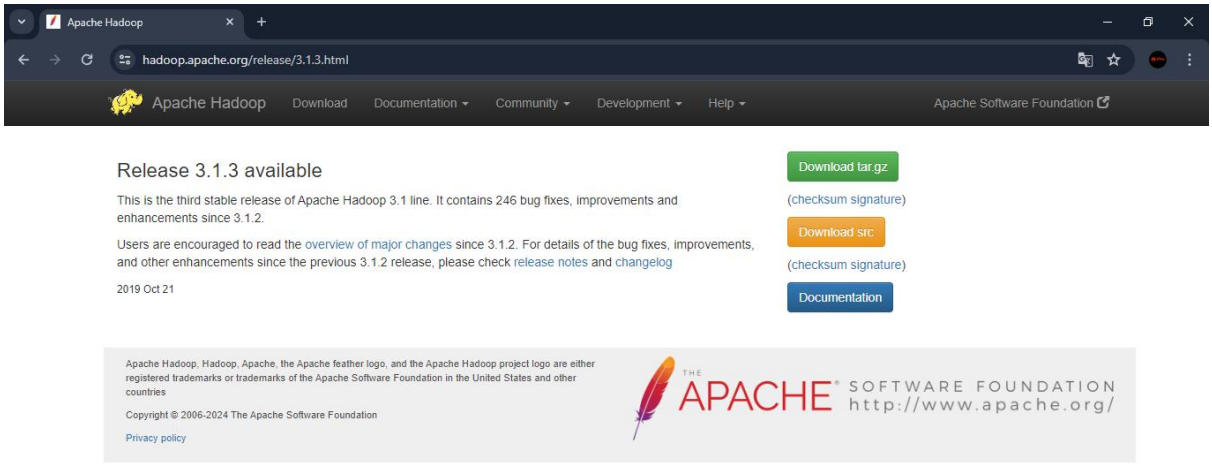
Oracle registration

Provide your email address and password if already registered, else register.

Jdk is downloaded

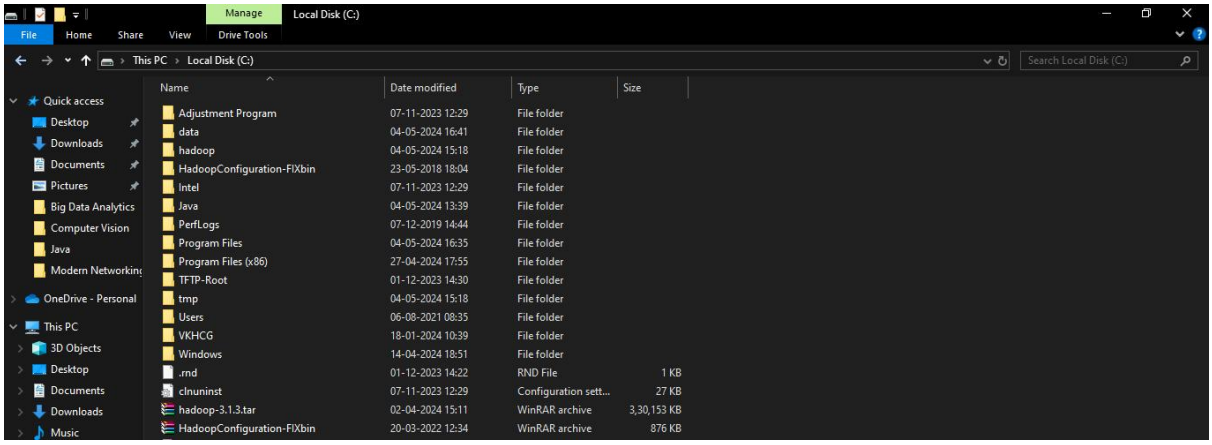
Step 2: Download Hadoop for the local system

https://hadoop.apache.org/releases.html

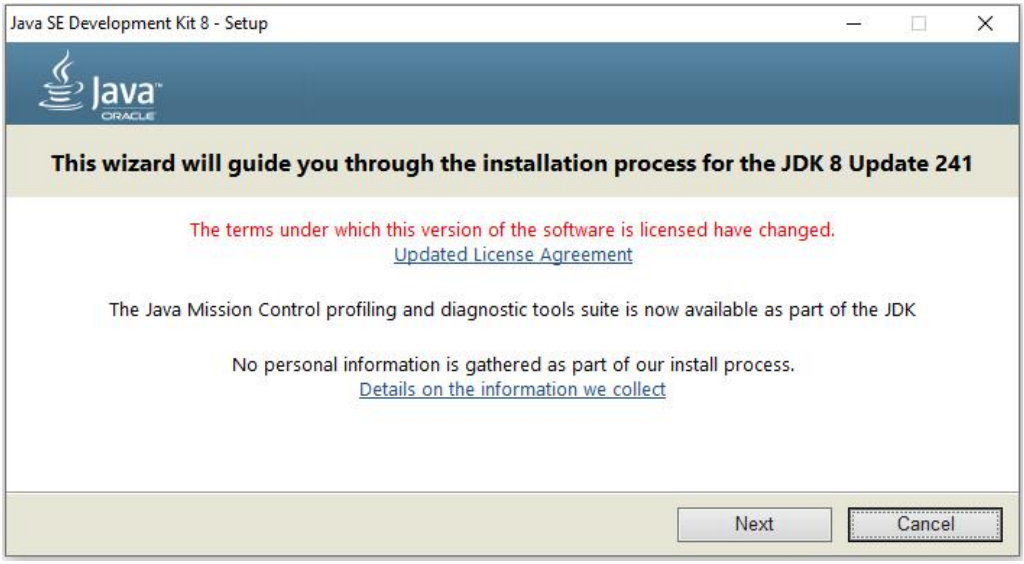


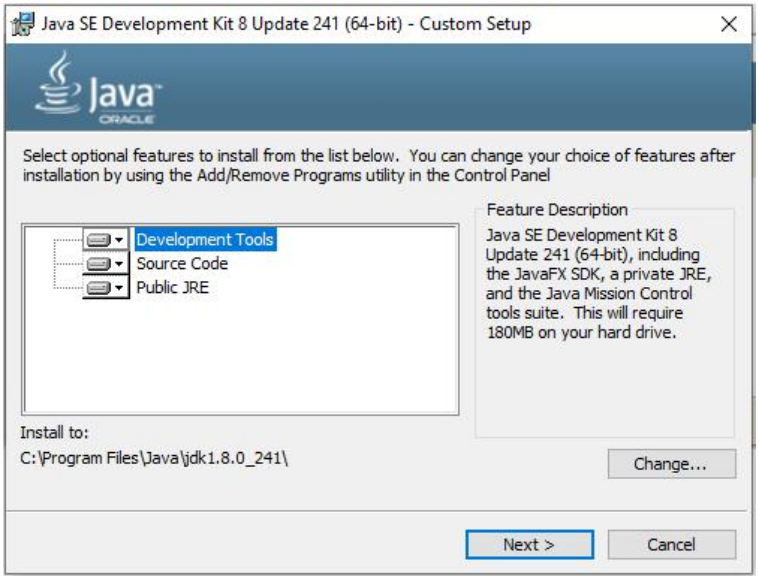
Hadoop downloaded

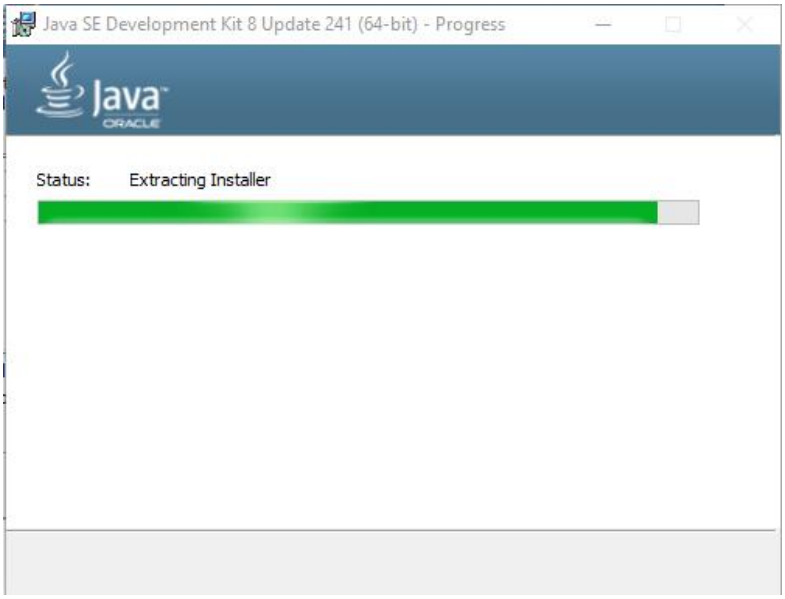
Move files to c drive

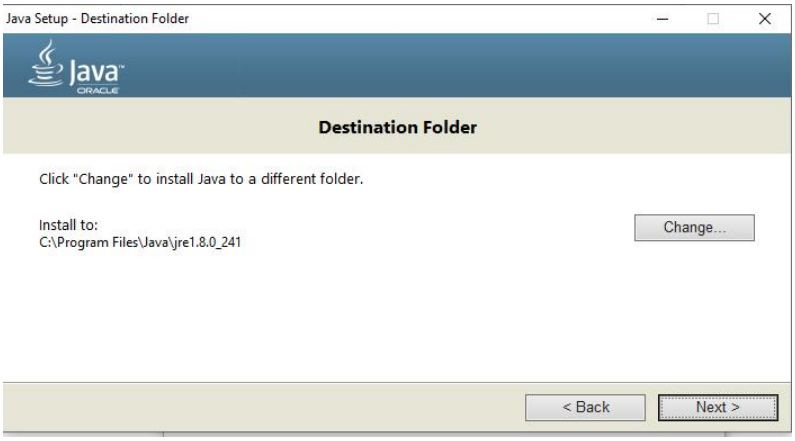


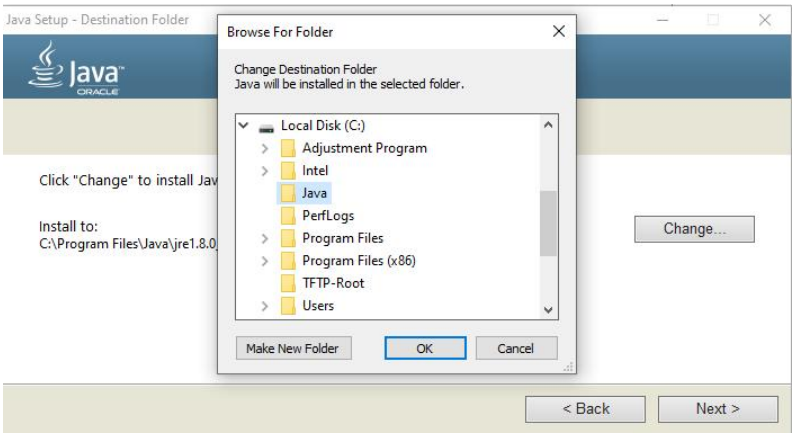
Step 3: Install Java

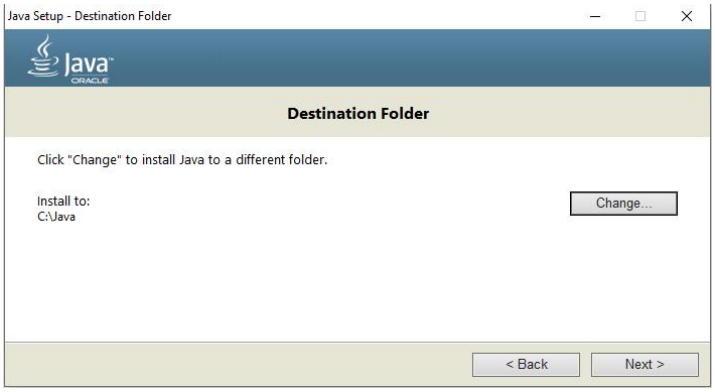




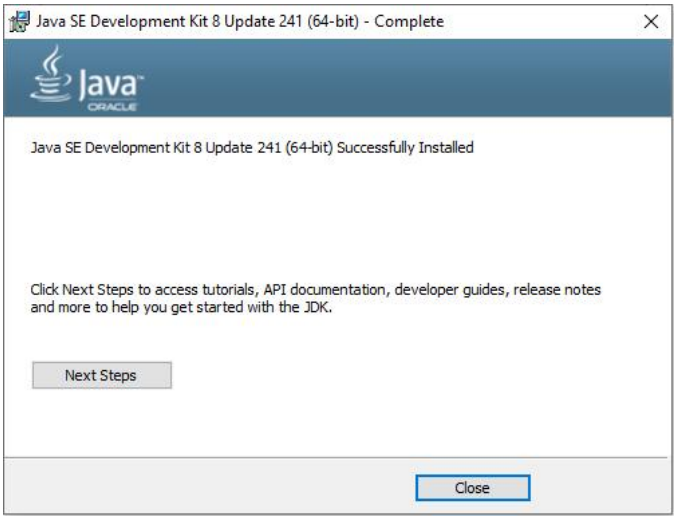


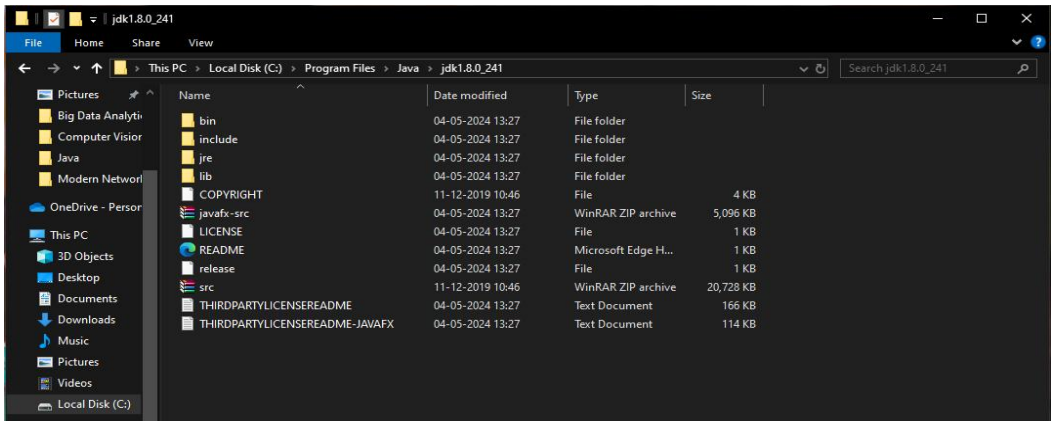

Java Setup - Destination Folder
Java
ORACLE
Destination Folder
Click "Change" to install Java to a different folder.
Install to:
C:\Program Files\Java\jre1.8.0_241
Change...
< Back
Next >

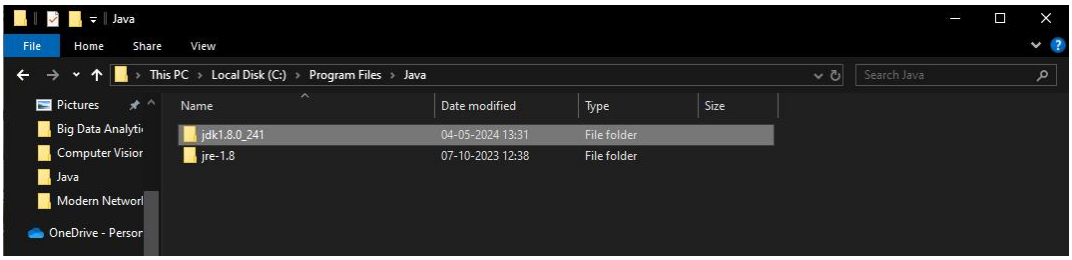

Java Setup - Destination Folder
Browse For Folder
Change Destination Folder
Java will be installed in the selected folder.
Local Disk (C:)
Adjustment Program
Intel
Java
PerfLogs
Program Files
Program Files (x86)
TFTP-Root
Users
Make New Folder
OK
Cancel
Click "Change" to install Jav
Install to:
C:\Program Files\Java\jre1.8.0
Change...
< Back
Next >


Java Setup - Destination Folder
Java
ORACLE
Destination Folder
Click "Change" to install Java to a different folder.
Install to:
C:\Java
Change...
< Back
Next >

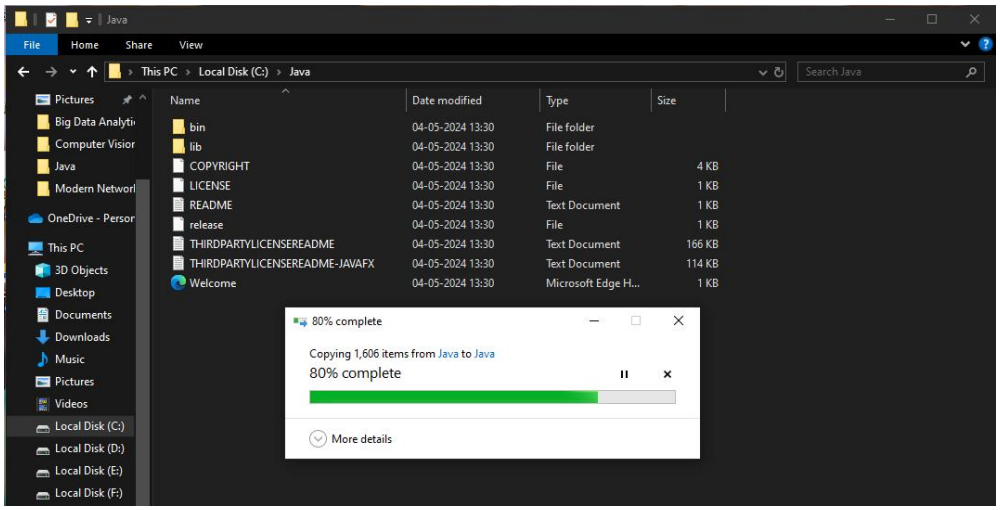

Java Setup - Progress
Java
ORACLE
Status: Installing Java
3 Billion
Devices Run Java
Java
#1 Development Platform
ORACLE

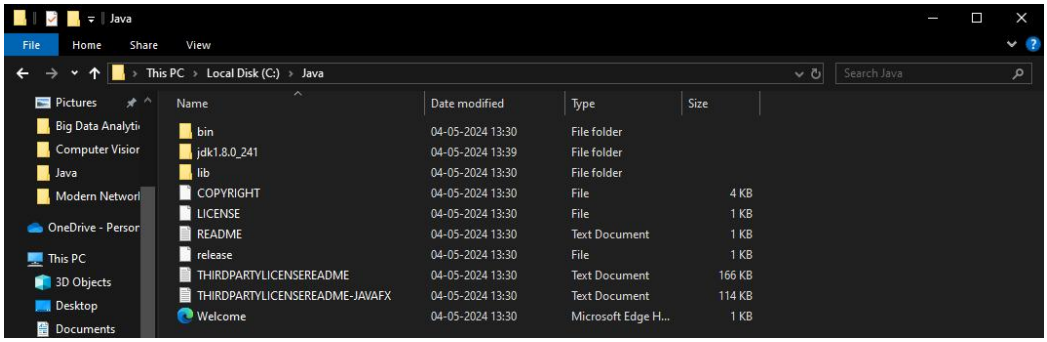






Paste it here.



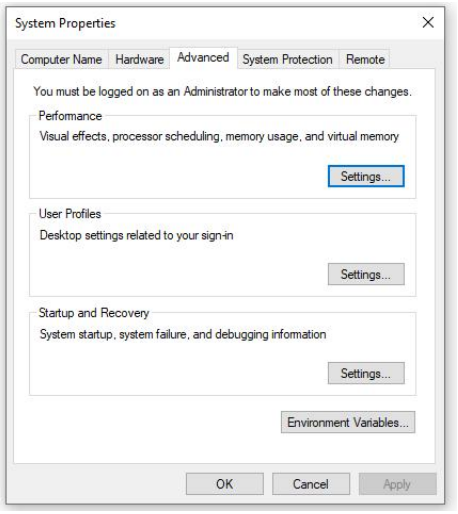


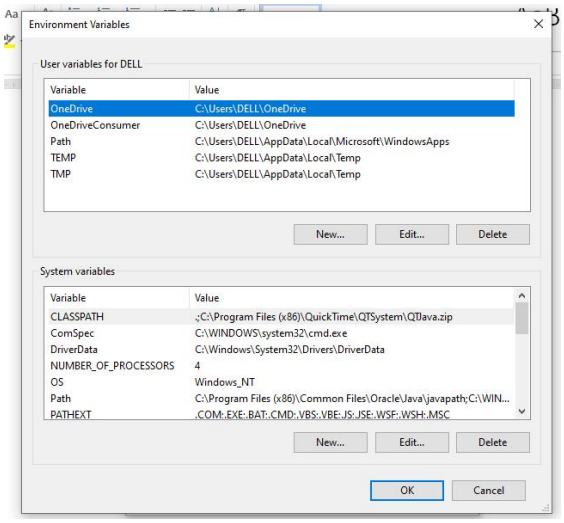
Now java will be available at:

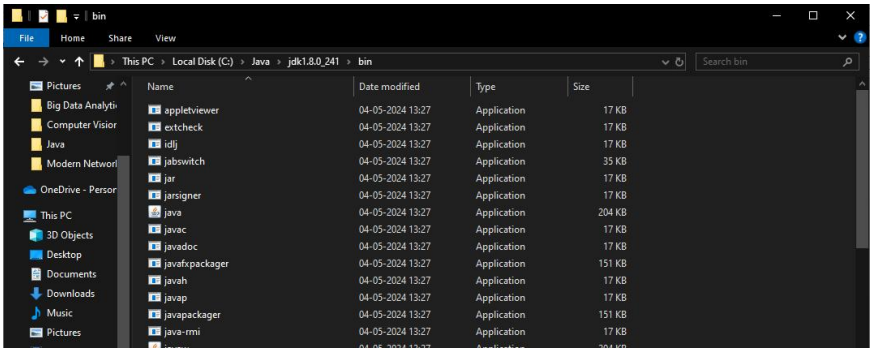
C:\Java

Step 4: Setting environment variables for java

Windows->settings->system->environment variables for system->edit the system environment variables

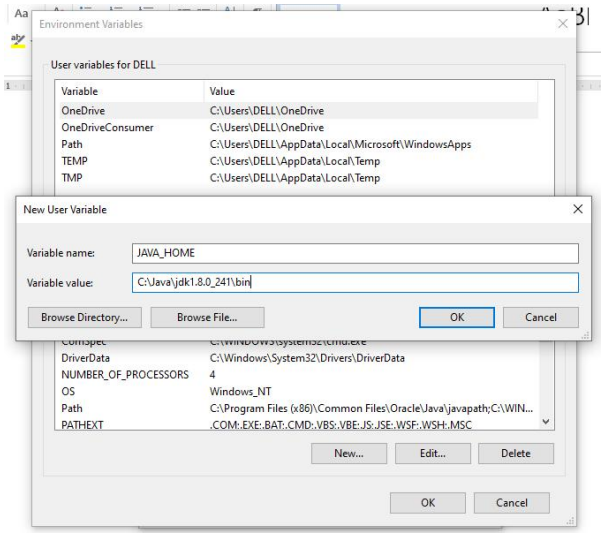






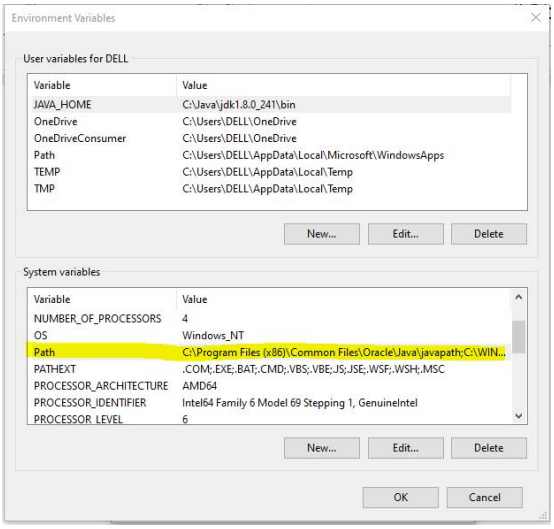
Set java home and path for java

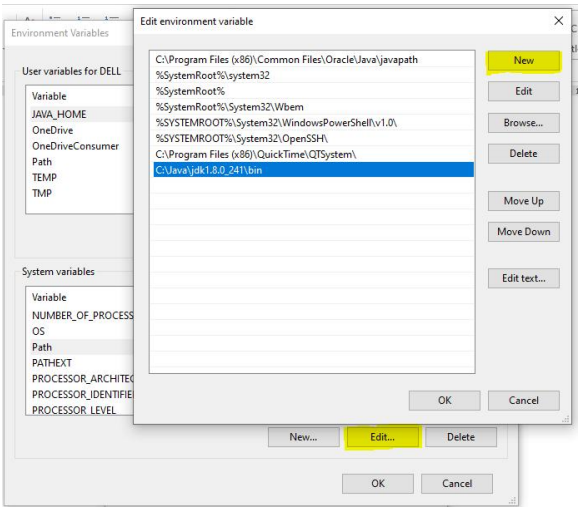
Set user variables



Set system variables

System variable->path->edit->new



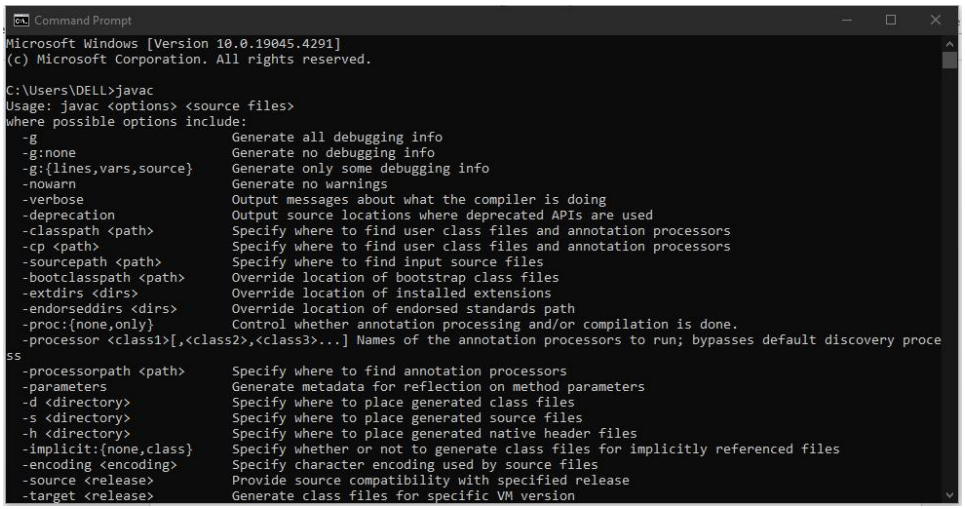


Click Ok-> OK->close

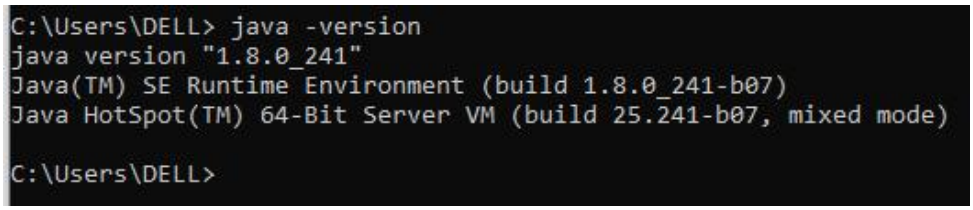
Java successfully installed

Step 5: Java version checking

Go to command prompt

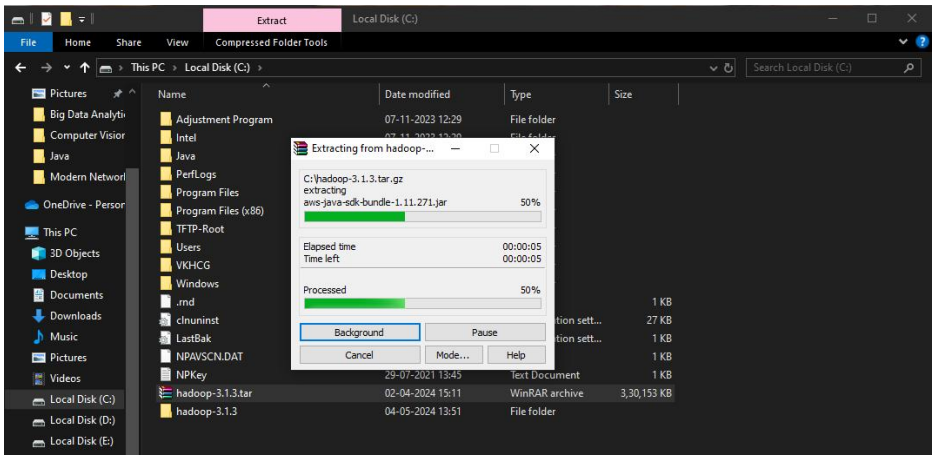


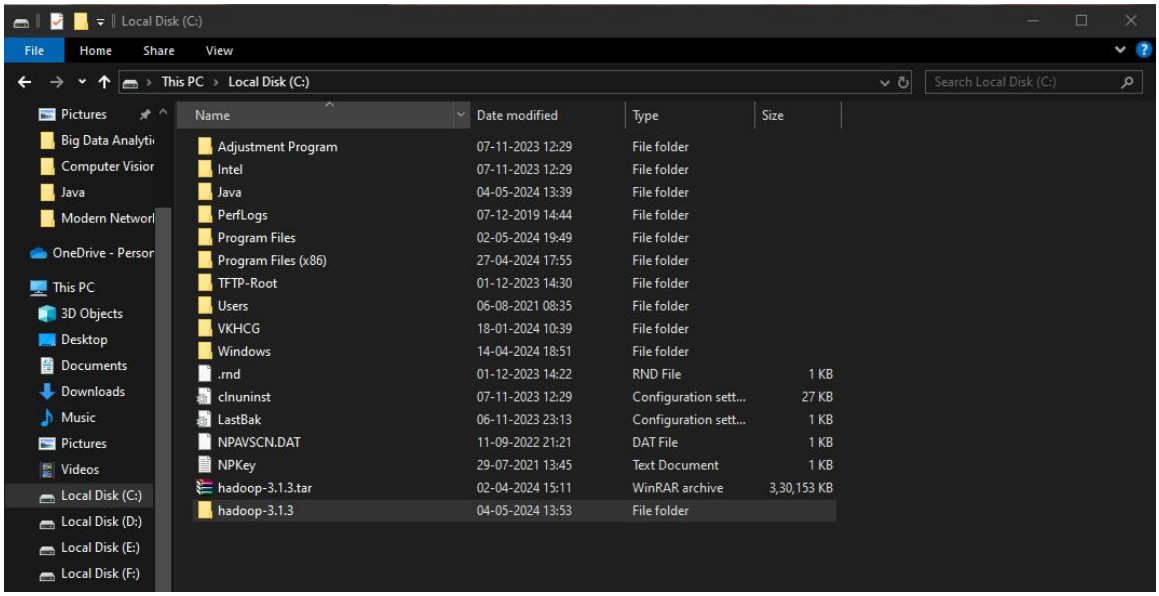
Check java version



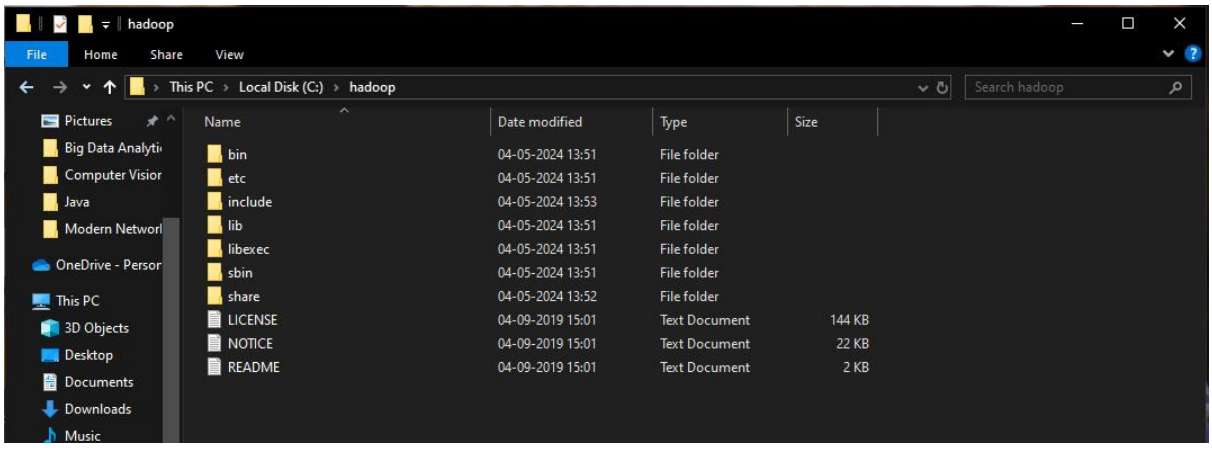
Step 6: Install Hadoop now to our local system

Unzip Hadoop setup file



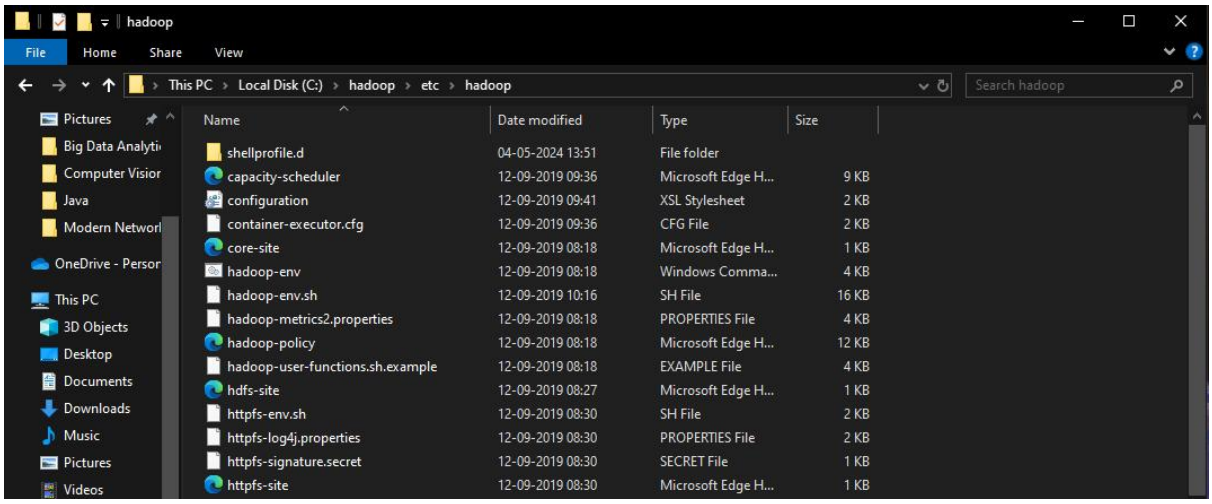


Rename Hadoop-3.1.3 folder as hadoop



Step 7: Configuration of Hadoop

hadoop->etc-> hadoop



Important files:

Core-site.xml

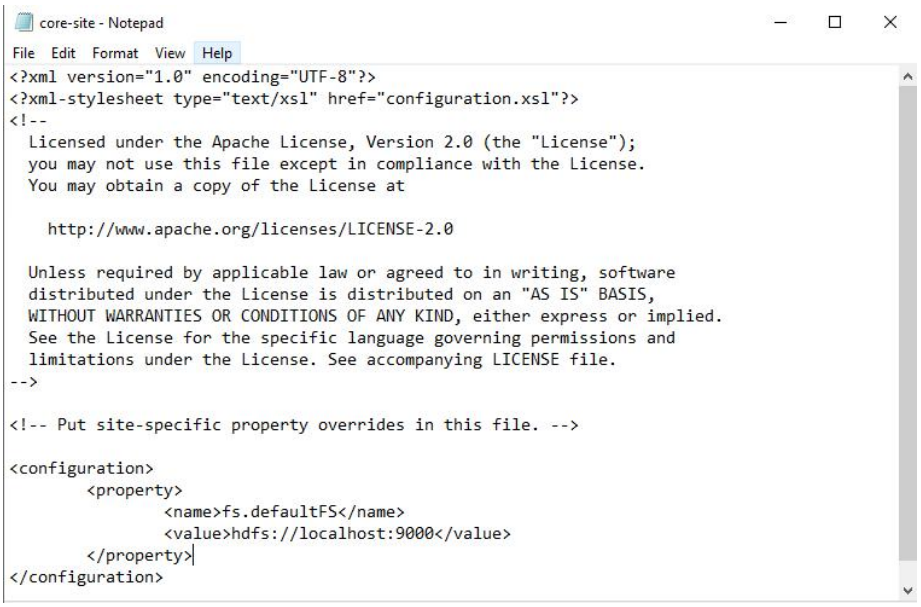
Hdfs-site.xml

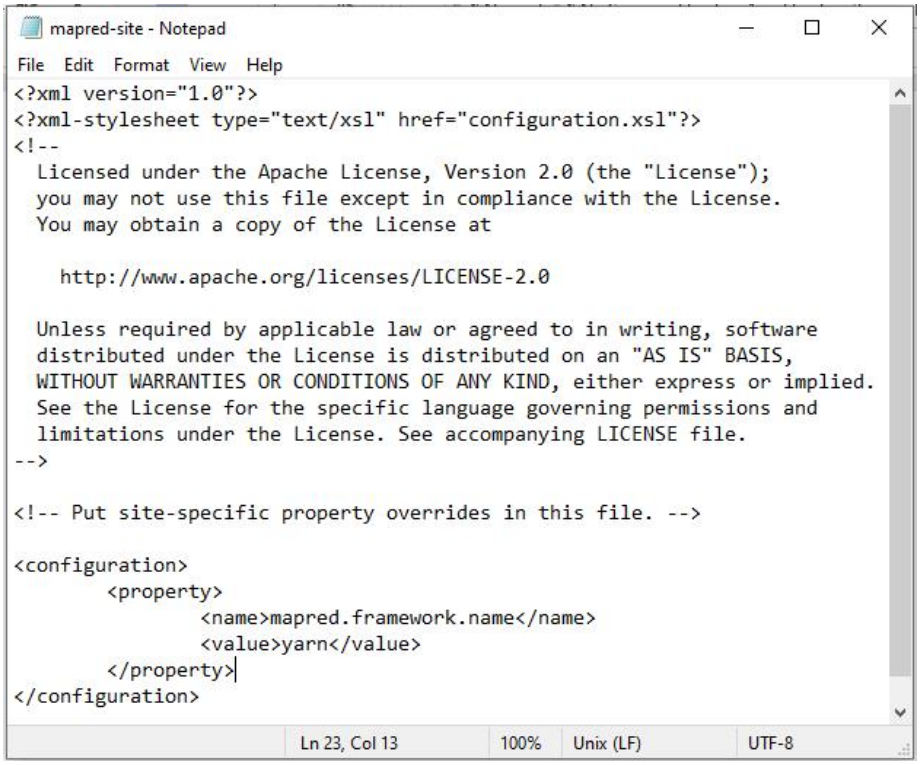
Mapred-site.xml

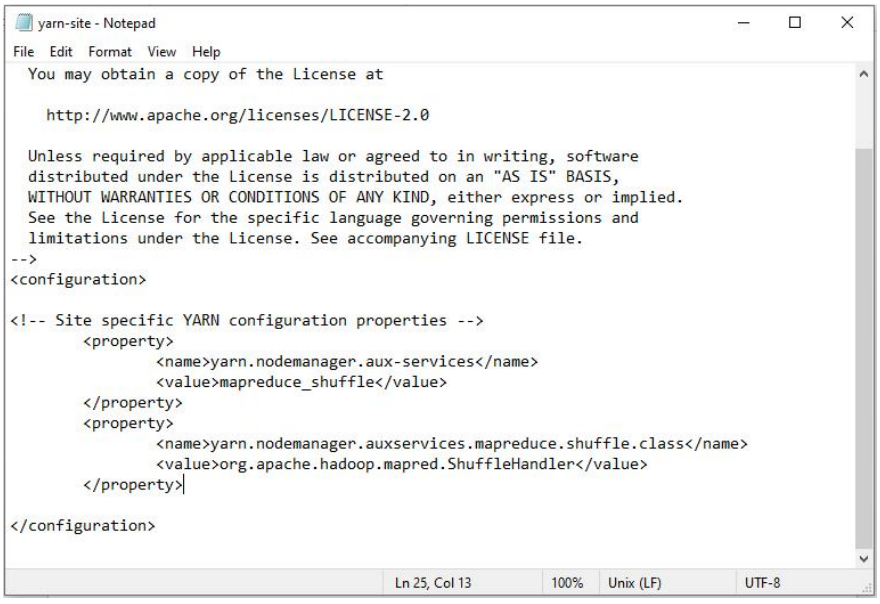
Yarn-site.xml

Hadoop-env windows command prompt file

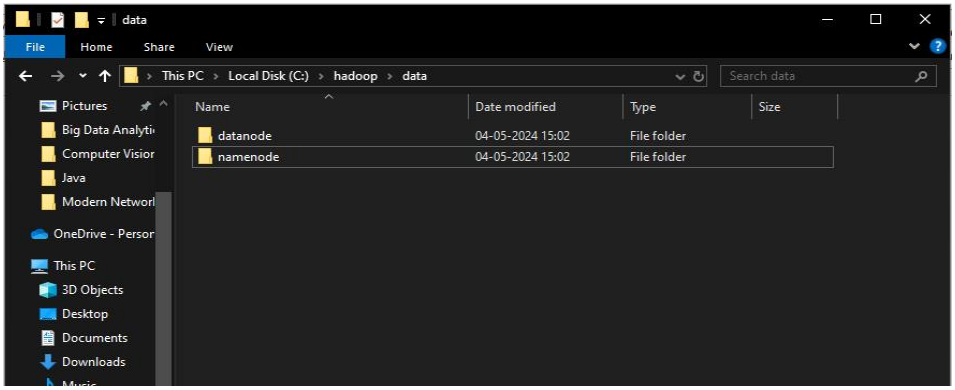
Open all these files in notepad

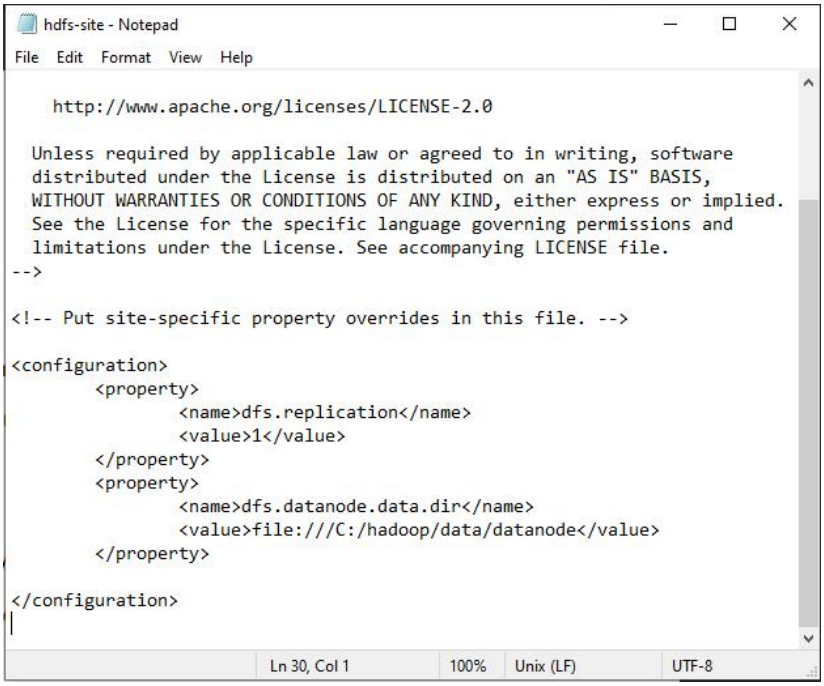


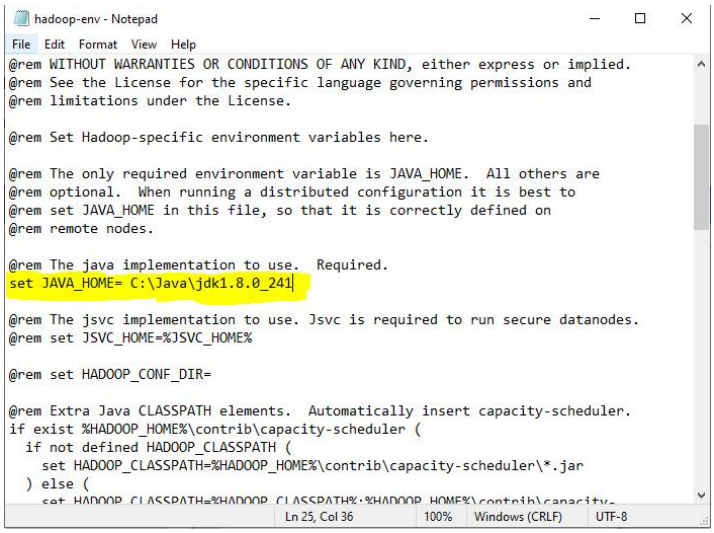




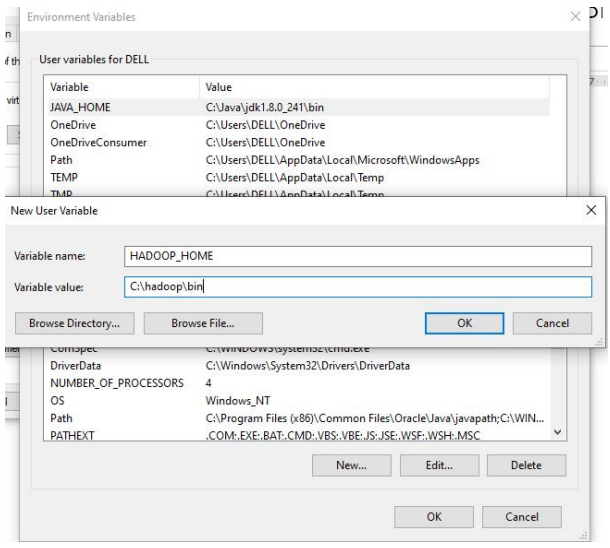
Create data directory and subdirectories in hadoop

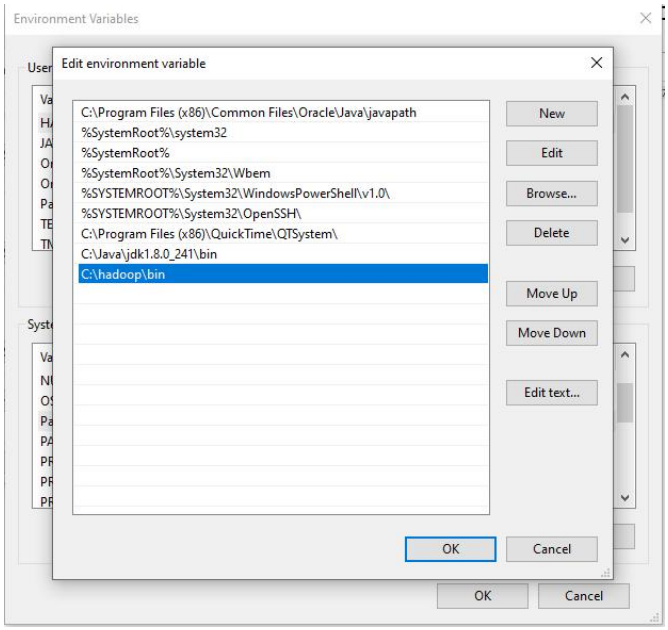


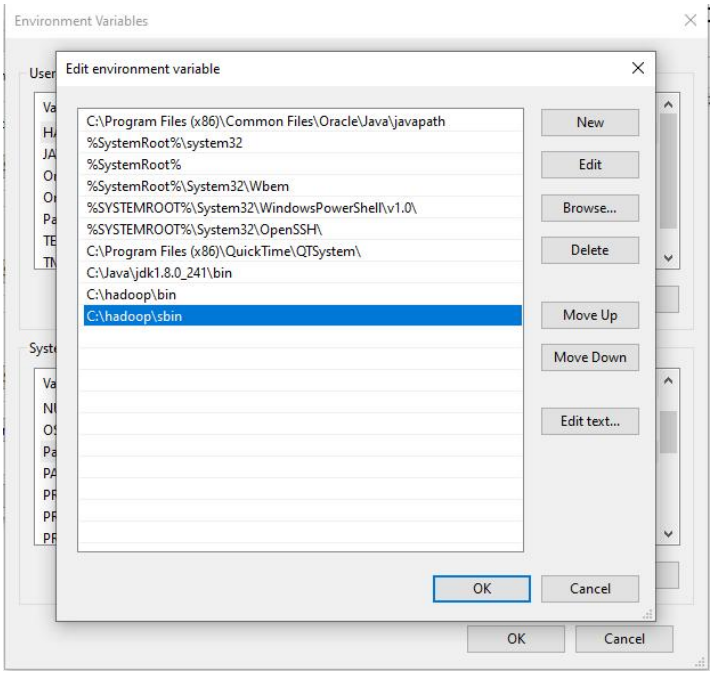




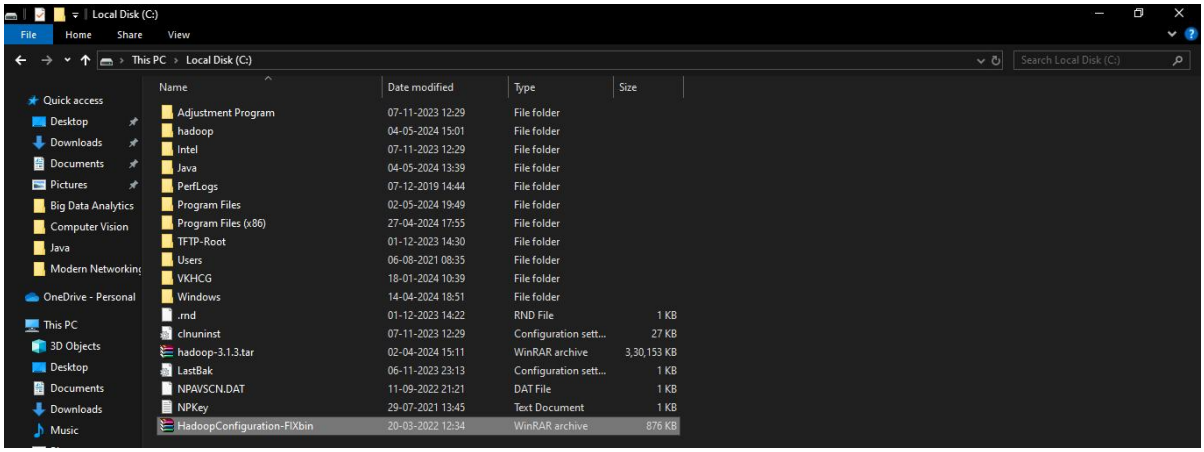
Set home and path for Hadoop

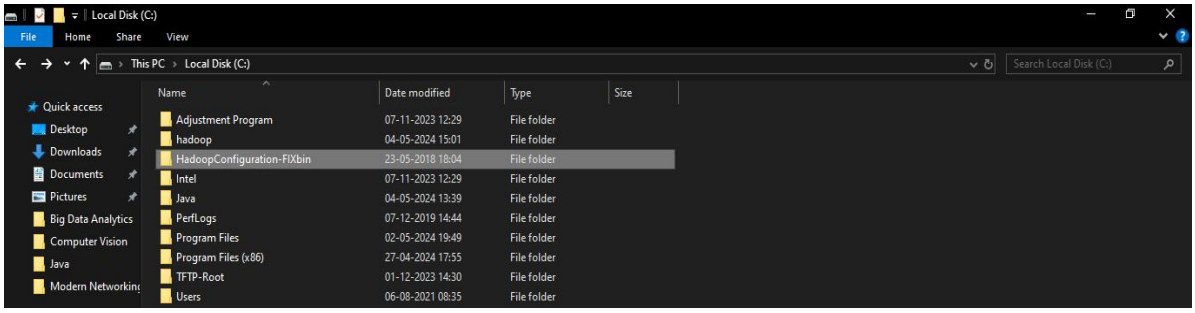




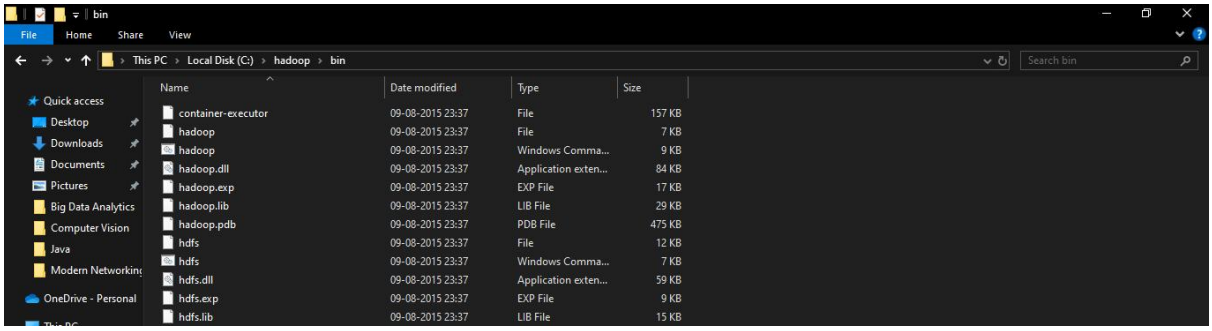


Other configuration files





Copy bin folder from HadoopConfiguration-Fixbin folder and replace hadoop\bin folder with this



Step 8: Verification of Hadoop installation

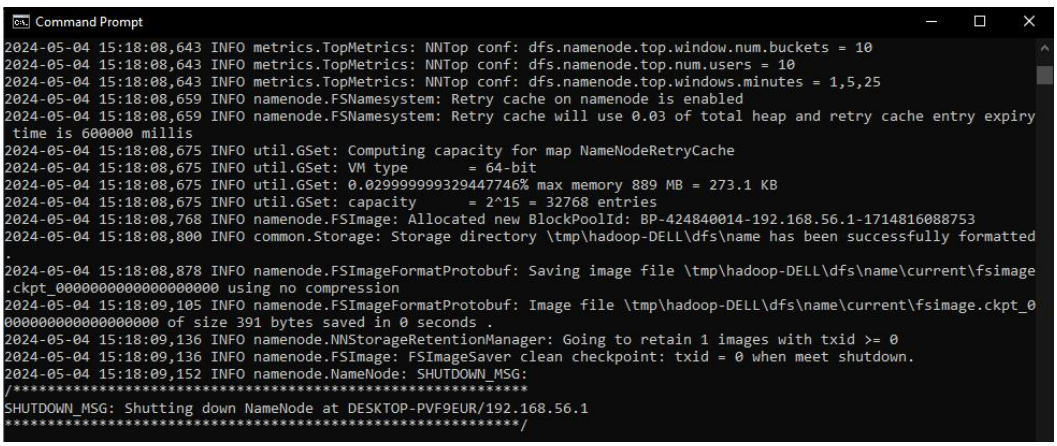
Go to command prompt.

Type:

hdfs namenode -format

set of files pop up on the terminal.

That means successful installation of Hadoop.

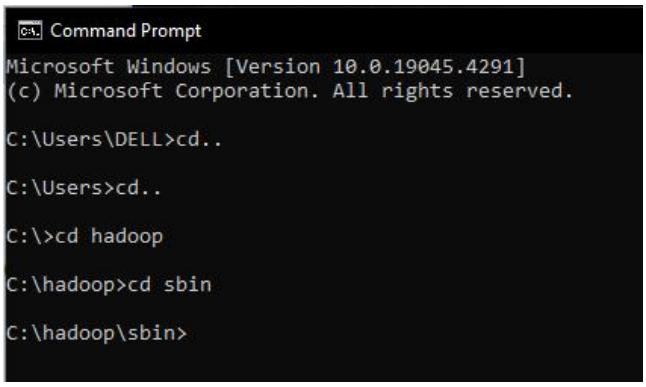


Namenode is successfully started.

Open new terminal and start all Hadoop daemons

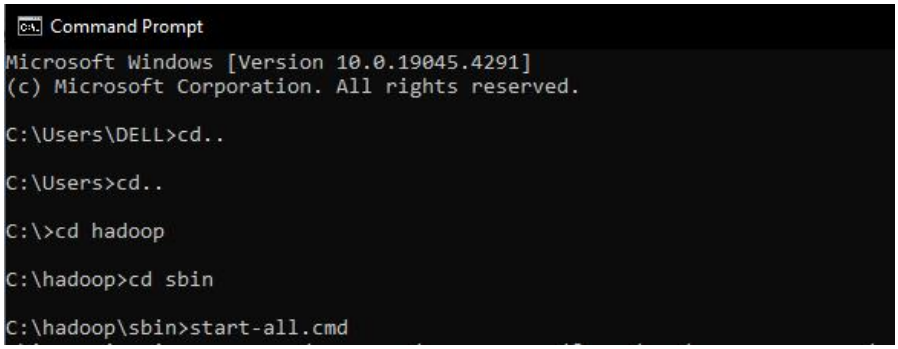
Go to Hadoop location.

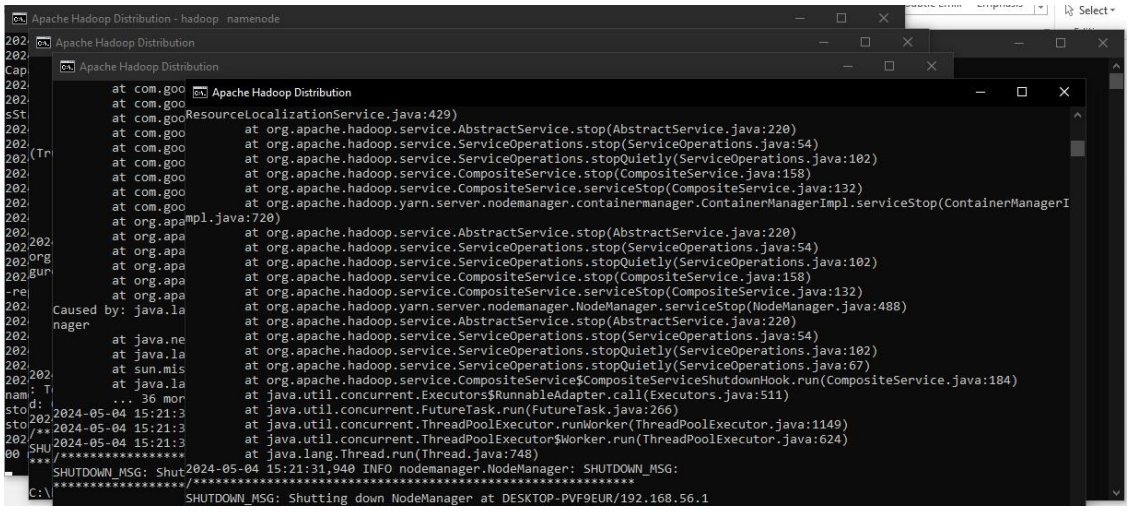
C:\hadoop\sbin



Type the command

start-all.cmd

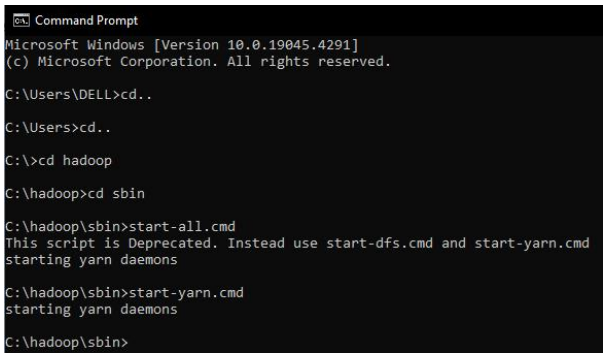


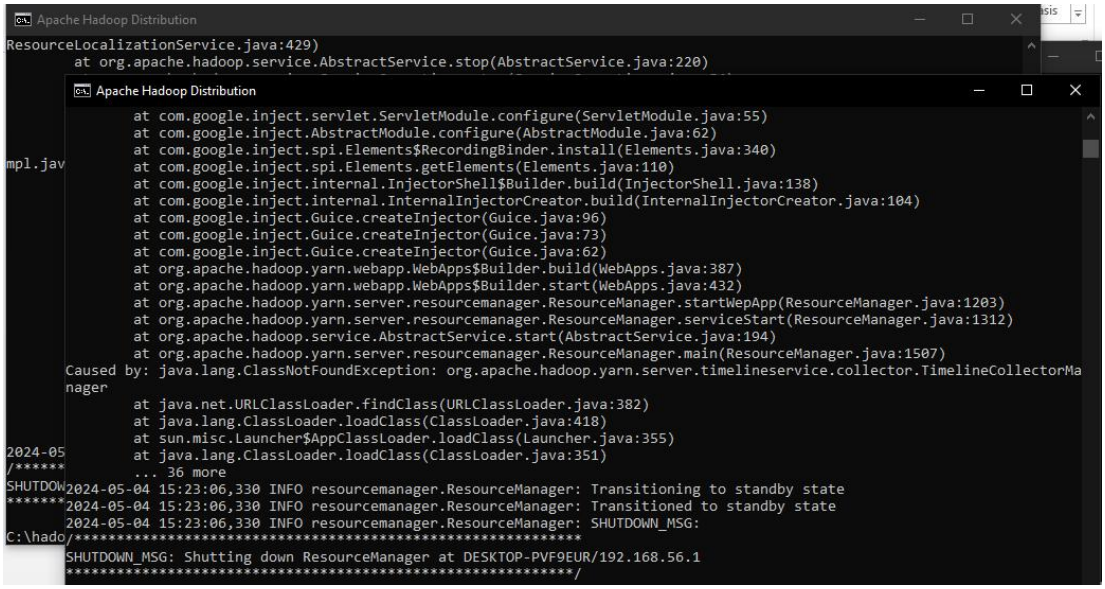


All the nodes will start successfully.

Also try the following command to start yarn daemons.

start-yarn.cmd





Practical 2

Aim: Implement an application that stores big data in Hbase / MongoDB and manipulate it using R / Python.

Theory: MongoDB is an open-source and the leading NoSQL database. It is a document-oriented database that offers high performance, easy scalability, and high availability. It uses documents and collections to organize data rather than relations. This makes it an ideal database management system for the storage of unstructured data.

MongoDB uses replica sets to ensure there is a high availability of data. Each replica set is made up of two or more replicas of data. This gives its users the ability to access their data at any time. The replica sets also create fault tolerance. MongoDB scales well to accommodate more data. It uses the sharing technique to scale horizontally and meet the changing storage needs of its users. MongoDB was developed to help developers unleash the power of data and software.

MongoDB is an unstructured database. It stores data in the form of documents. MongoDB is able to handle huge volumes of data very efficiently and is the most widely used NoSQL database as it offers rich query language and flexible and fast access to data.

The Architecture of a MongoDB Database

The information in MongoDB is stored in documents. Here, a document is analogous to rows in structured databases.

• Each document is a collection of key-value pairs

• Each key-value pair is called a field

• Every document has an _id field, which uniquely identifies the documents

• A document may also contain nested documents

• Documents may have a varying number of fields (they can be blank as well)

These documents are stored in a collection. A collection is literally a collection of documents in MongoDB. This is analogous to tables in traditional databases.

Unlike traditional databases, the data is generally stored in a single collection in MongoDB, so there is no concept of joins (except $lookup operator, which performs left-outer-join like operation). MongoDB has the nested document instead.

PyMongo is a Python library that enables us to connect with MongoDB. It allows us to perform basic operations on the MongoDB database.

We have chosen Python to interact with MongoDB because it is one of the most commonly used and considerably powerful languages for data science. PyMongo allows us to retrieve the data with dictionary-like syntax.

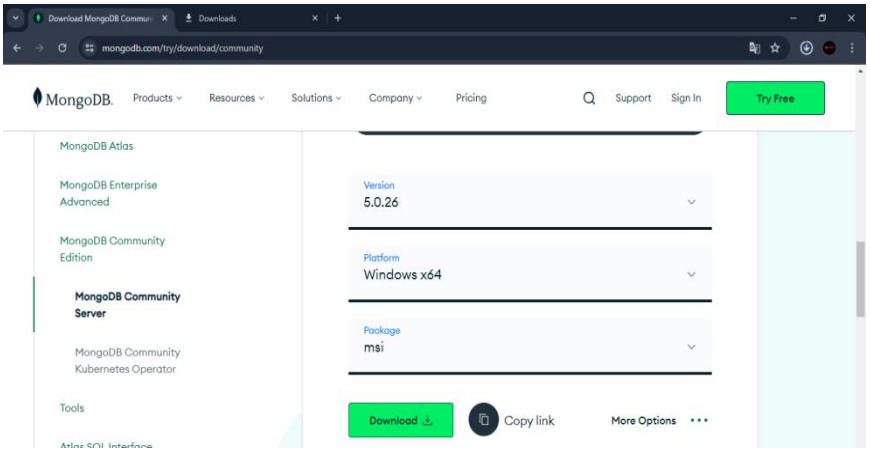
We can also use the dot notation to access MongoDB data. Its easy syntax makes our job a lot easier. Additionally, PyMongo's rich documentation is always standing there with a helping hand. We will use this library for accessing MongoDB.

Steps of the installation:

Step 1: Download MongoDB

Go to official website: MongoDB Community server

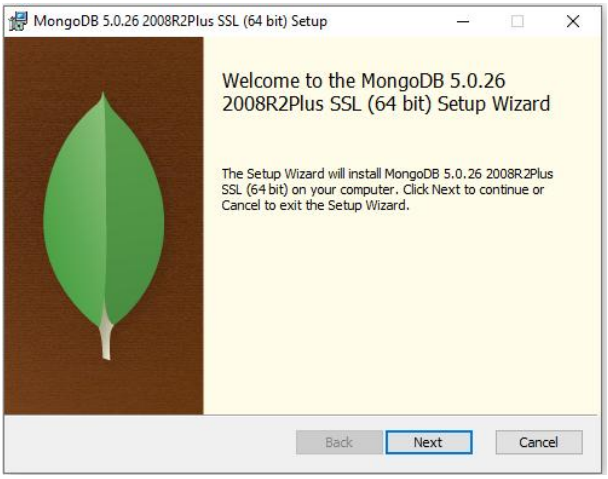
https://www.mongodb.com/try/download/community

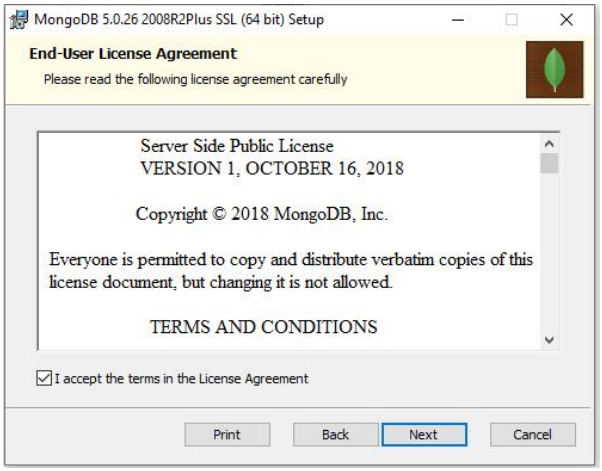


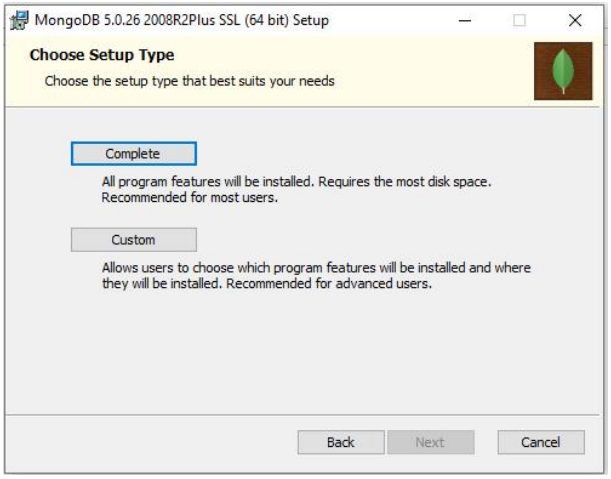
Click on download

Step 2: Install MongoDB

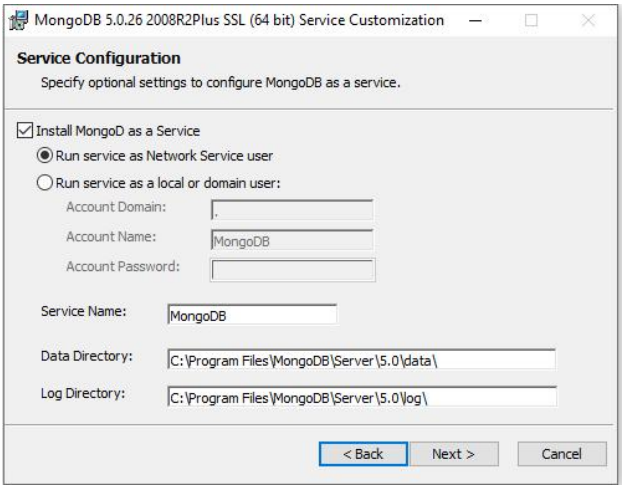
It will download msi file. Click on it and Start the installation.



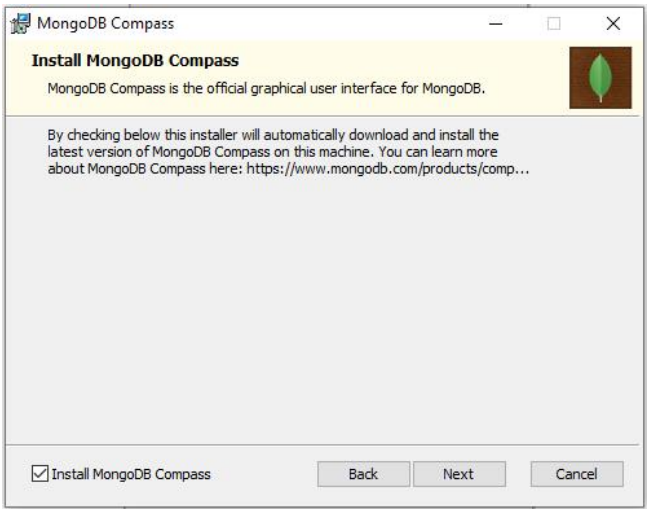




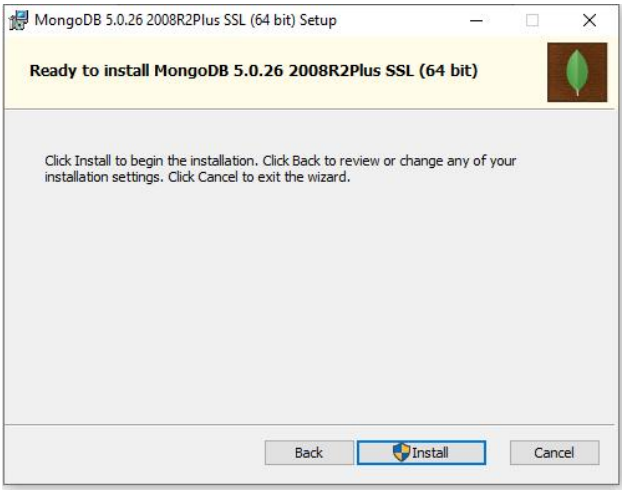
Click on complete option



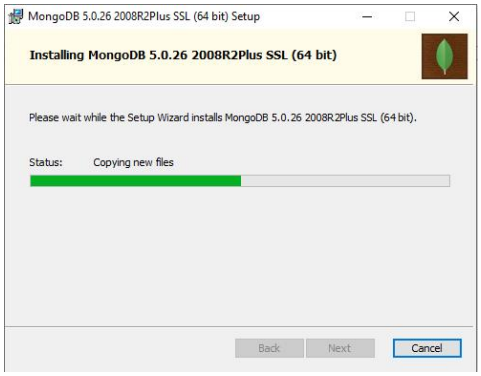
Keep all these setting as it is. Click on next.

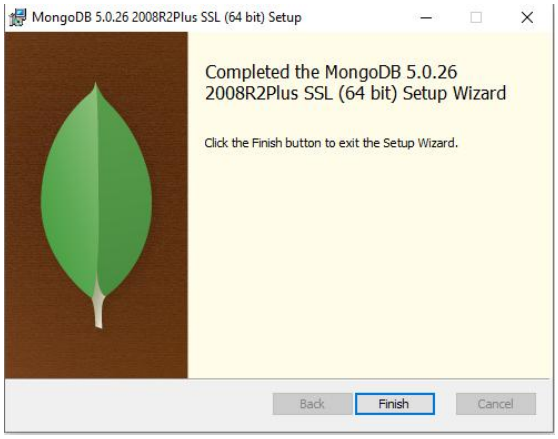


Click on next



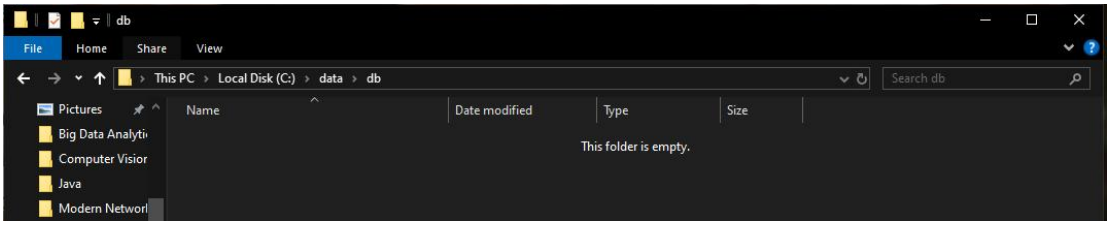
Click on install





Step 3: Verify MongoDB Installation

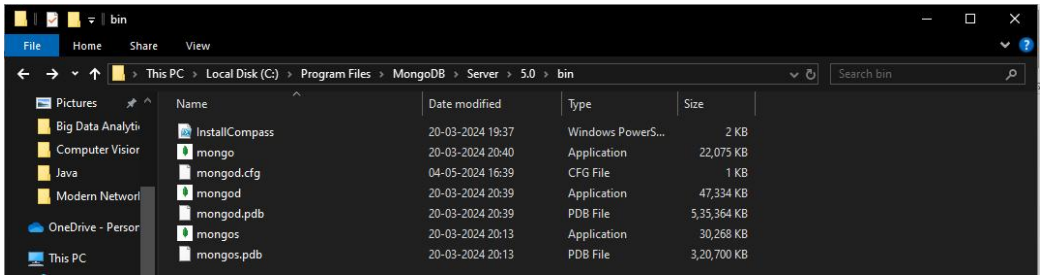
First we will create C:\data\db directory



Now go to

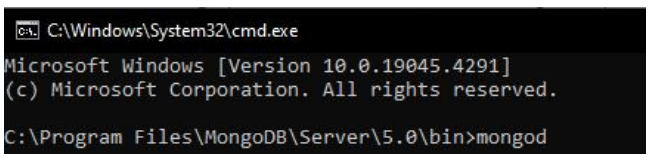
C:\Program Files\MongoDB\Server\5.0\bin

Start command prompt from this location



To start mongo db server

Enter mongod command



Mongo daemon is started now

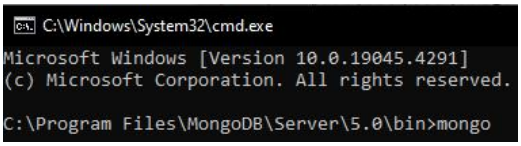
To open the mongo shell

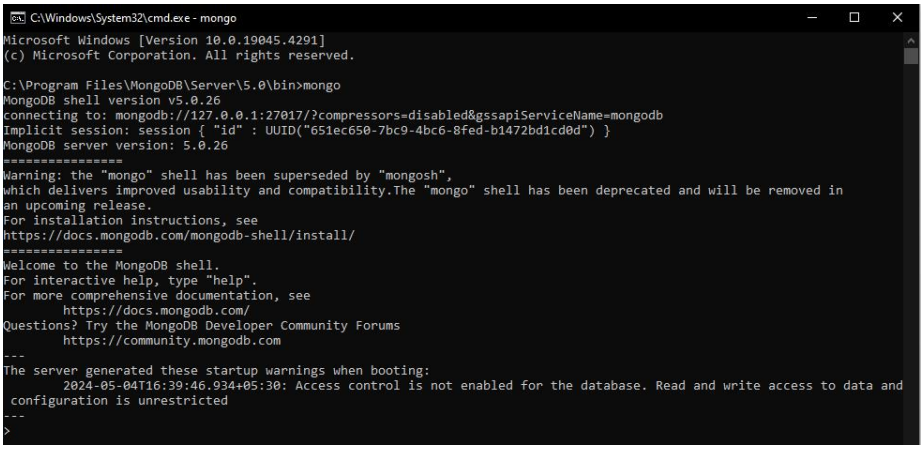
Go to

C:\Program Files\MongoDB\Server\5.0\bin

Start command prompt from this location

Fire the command: mongo

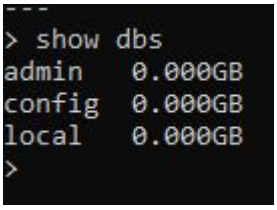




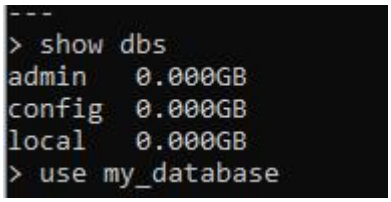
Mongo shell is started

To see all the default databases:

>show dbs



To create new database named my_database:



To create collection in the database:

And insert json values into it:

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> use my_database
switched to db my_database
> db.books.insert({"name":"It Ends With Us"})
```

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> use my_database
switched to db my_database
> db.books.insert({"name":"It Ends With Us"})
WriteResult({ "nInserted" : 1 })
```

To see entered collection:

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> use my_database
switched to db my_database
> db.books.insert({"name":"It Ends With Us"})
WriteResult({ "nInserted" : 1 })
> show collections;
books
>
```

To see all the documents in the collection:

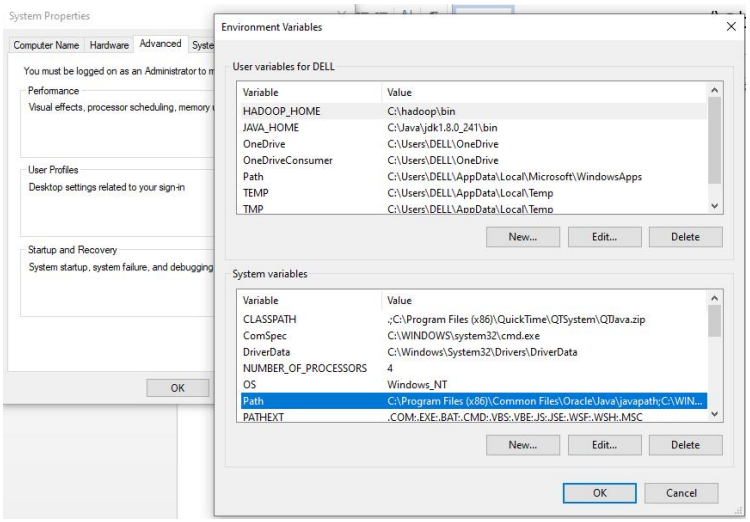
```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
> use my_database
switched to db my_database
> db.books.insert({"name":"It Ends With Us"})
WriteResult({ "nInserted" : 1 })
> show collections;
books
> db.books.find()
{ "_id" : ObjectId("66361991164b7d2f82dae287"), "name" : "It Ends With Us" }
>
```

To set path of MongoDB server and shell

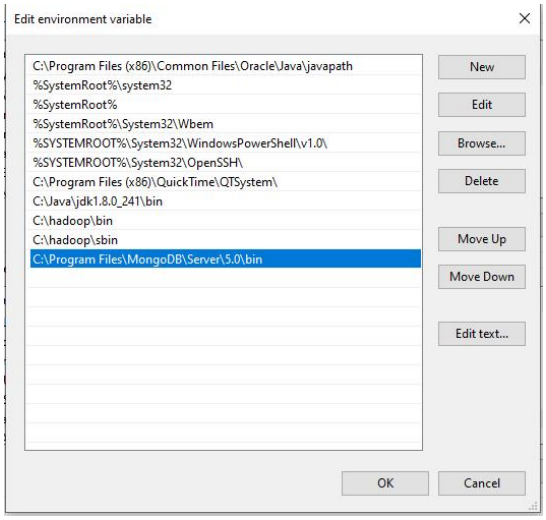
Copy in clipboard: C:\Program Files\MongoDB\Server\5.0\bin

Go to environment variables

Add mongodb path to system path variable



Click on New

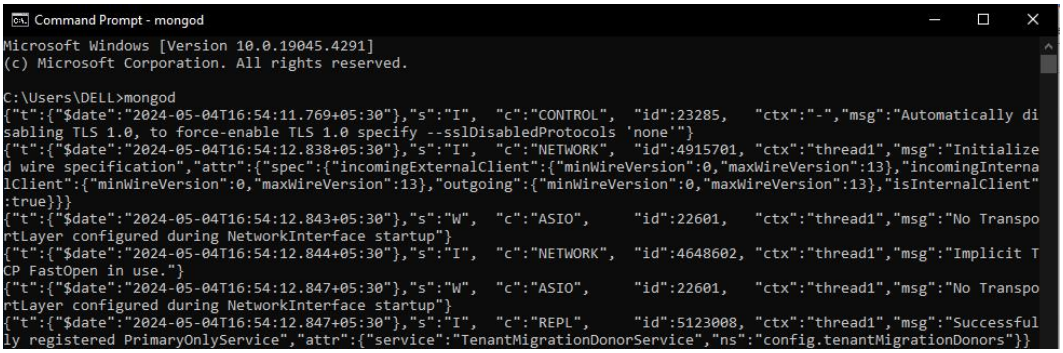


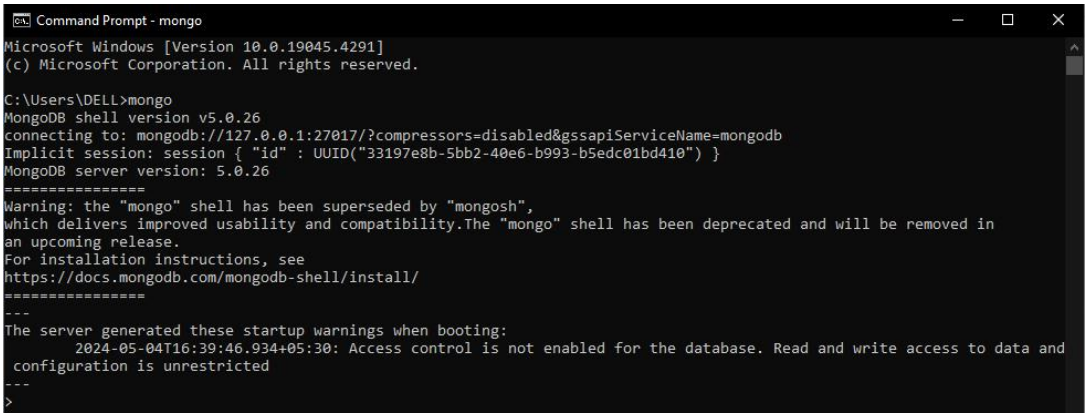
Paste the path

Click on ok..ok

Run mongod and mongo command from command prompt once again from anywhere, it will

start mongo server and mongo shell





To get the effect of running mongod as service, Restart your windows operating system.

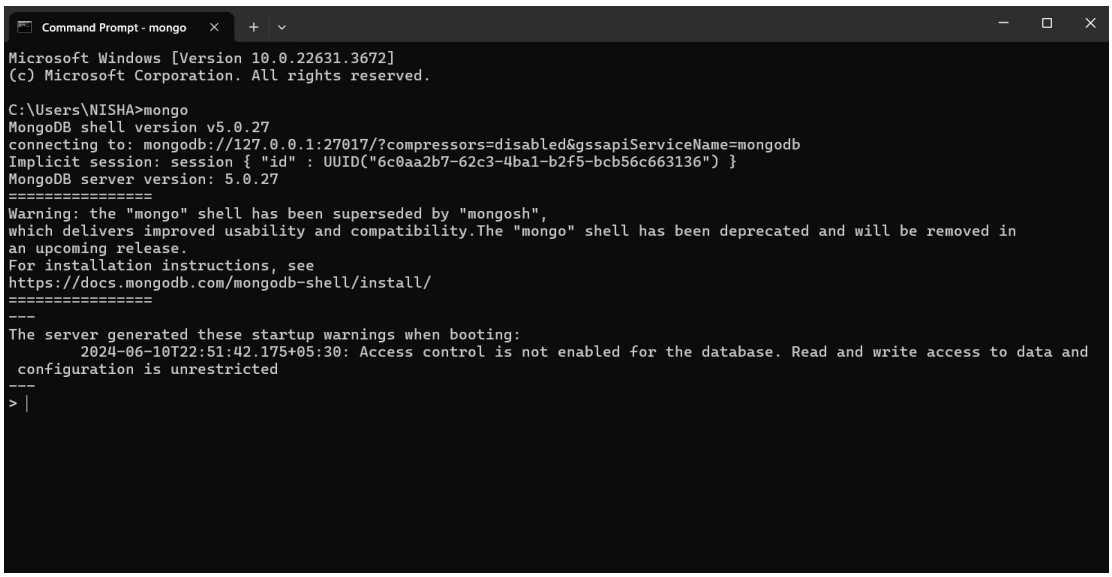
Restarted the machine

To check mongod service is running automatically:

Open command prompt

Give mongo command without running mongod command in another terminal.

It will start mongod server.

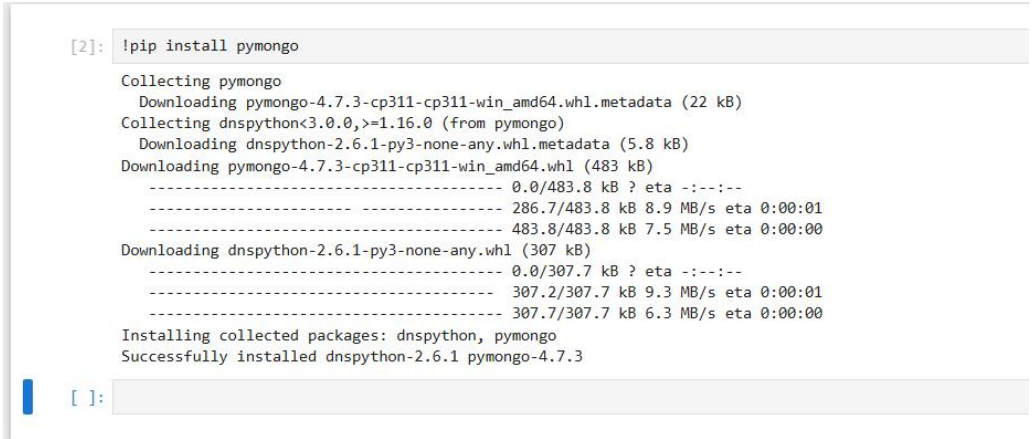


Step 4: Install MongoDB Python on Windows

We will be performing a few key basic operations on a MongoDB database in Python using

the PyMongo library.

Install package to use MongoDB
To install this package run the command !pip install pymongo on Jupyter Notebook



Step 5: Verify MongoDB Python Connection

To retrieve the data from a MongoDB database, we will first connect to it. Write and execute

the below code in your spider anaconda

import pymongo

mongo_uri = "mongodb://localhost:27017/"

client = pymongo.MongoClient(mongo_uri)

Let’s see the available databases:

print(client.list_database_names())

We will use the my_database database for our purpose. Let’s set the cursor to the same

database:

db = client.my_database

connect to analysis database

The list_collection_names command shows the names of all the available collections:

print(db.list_collection_names())

Let’s see the number of books we have. We will connect to the customers collection and then

print the number of documents available in that collection:

table=db.books

print(table.count_documents({})) #gives the number of documents in the table

CODE

```
import pymongo
mongo_uri = "mongodb://localhost:27017/"
client = pymongo.MongoClient(mongo_uri)
print(client.list_database_names())
db = client.my_database
print(db.list_collection_names())
table=db.books
print(table.count_documents({}))
print('Tanyaa - 53004230009')
```

```
['admin', 'config', 'local', 'my_database']
['books']
1
Tanyaa - 53004230009
```

Practical 3

AIM: Implement Regression Model to import a data from web storage. Name the dataset and now do Logistic Regression to find out relation between variables. Also check if the model is fit or not.

Theory:

A Regression models are used to describe relationships between variables by fitting a line to the observed data. Regression allows you to estimate how a dependent variable change as the independent variable(s) change.

Linear Regression:

Linear regression quantifies the relationship between one or more predictor variable(s) and one outcome variable. Linear regression is commonly used for predictive analysis and modelling. For example, it can be used to quantify the relative impacts of age, gender, and diet (the predictor variables) on height (the outcome variable).

Simple linear regression formula:

The formula for a simple linear regression is:

$$Y = \beta_0 + \beta_1 X + E$$

- Y is the predicted value of the dependent variable (y) for any given value of the independent variable (x).
- B0 is the intercept, the predicted value of y when the x is 0.
- B1 is the regression coefficient – how much we expect y to change as x increases.
- X is the independent variable (the variable we expect is influencing y).
- e is the error of the estimate, or how much variation there is in our estimate of the regression coefficient.

  Linear regression is a powerful tool for understanding and predicting the behaviour of a variable, however, it needs to meet a few conditions to be accurate and dependable solutions.

1. **Linearity:** The independent and dependent variables have a linear relationship with one another. This implies that changes in the dependent variable follow those in the independent variable(s) in a linear fashion.
2. **Independence:** The observations in the dataset are independent of each other. This means that the value of the dependent variable for one observation does not depend on the value of the dependent variable for another observation.
3. **Homoscedasticity**: Across all levels of the independent variable(s), the variance of the errors is constant. This indicates that the amount of the independent variable(s) has no impact on the variance of the errors.
4. **Normality:** The errors in the model are normally distributed.
5. **No multicollinearity:** There is no high correlation between the independent variables. This indicates that there is little or no correlation between the independent variables.

CODE

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
# Load Dataset
diabetes = datasets.load_diabetes()
# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y = True)
# Description of the dataset
print(diabetes['DESCR'])
print(diabetes.feature_names)
```

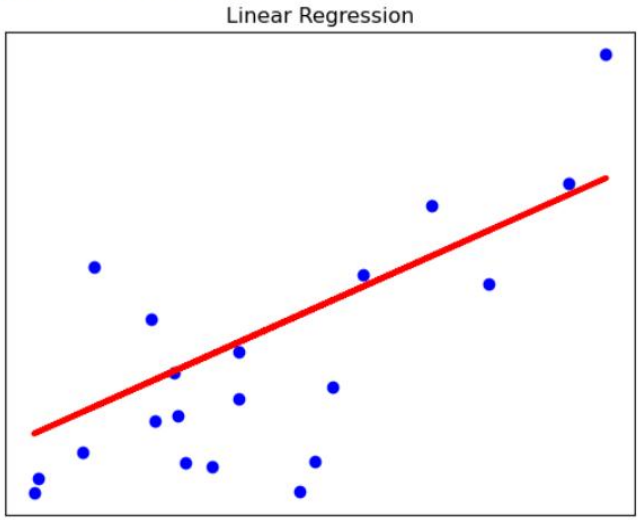
```
Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n =
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

**Data Set Characteristics:**

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression one year after baseline

  :Attribute Information:
      - age     age in years
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of `n_samples` (i.e. the sum
of squares of each column totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion), 407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)

['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

```
# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]
# Split the data intro training and testing datasets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
# Split the targets into training and testing datasets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]
```

```
# Create linear regression object

regr = linear_model.LinearRegression()

# Train the model using the training sets

regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing sets

diabetes_y_pred = regr.predict(diabetes_X_test)

# The Coefficients

print('Coefficients : \n', regr.coef_)

# The mean squared error

print("Mean Squared Error: %.2f" %
mean_squared_error(diabetes_y_test,diabetes_y_pred))

# The coefficient of determination: 1 is perfect prediction

print("Coefficient of determination: %.2f" %
r2_score(diabetes_y_test,diabetes_y_pred))

# plot the outputs

plt.scatter(diabetes_X_test, diabetes_y_test, color='blue')

plt.plot(diabetes_X_test, diabetes_y_pred, color='red', linewidth=3)

plt.xticks(())

plt.yticks(())

plt.title('Linear Regression')

plt.show()

print('Tanya Dhamija - 53004230009')
```

OUTPUT

```
Coefficients :
 [938.23786125]
Mean Squared Error: 2548.07
Coefficient of determination: 0.47
```




```
Tanya Dhamija - 53004230009
```

Practical 4

Aim: Apply Multiple Regression on a dataset having a continuous independent variable.

Theory: Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable.

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Assumptions for Multiple Linear Regression:

• A linear relationship should exist between the Target and predictor variables.

• The regression residuals must be normally distributed.

• MLR assumes little or no multicollinearity (correlation between the independent variable) in data.

Multiple linear regression (MLR) is used to determine a mathematical relationship among several random variables. In other terms, MLR examines how multiple independent variables are related to one dependent variable. Once each of the independent factors has been determined to predict the dependent variable, the information on the multiple variables can be used to create an accurate prediction on the level of effect they have on the outcome variable. The model creates a relationship in the form of a straight line (linear) that best approximates all the individual data points.

Multiple linear regression formula:

$Y = \beta 0 + \beta 1\ X1 + \cdots + \beta nXn + E$

• Y = the predicted value of the dependent variable

• B0 = the y-intercept (value of y when all other parameters are set to 0)

• B1X1= the regression coefficient (B1) of the first independent variable (X1) (a.k.a. theeffect that increasing the value of the independent variable has on the predicted y value)

• … = do the same for however many independent variables you are testing

• BnXn = the regression coefficient of the last independent variable

• e = model error (a.k.a. how much variation there is in our estimate of y)

A multiple regression considers the effect of more than one explanatory variable on some outcome of interest. It evaluates the relative effect of these explanatory, or independent, variables on the dependent variable when holding all the other variables in the model constant.

CODE

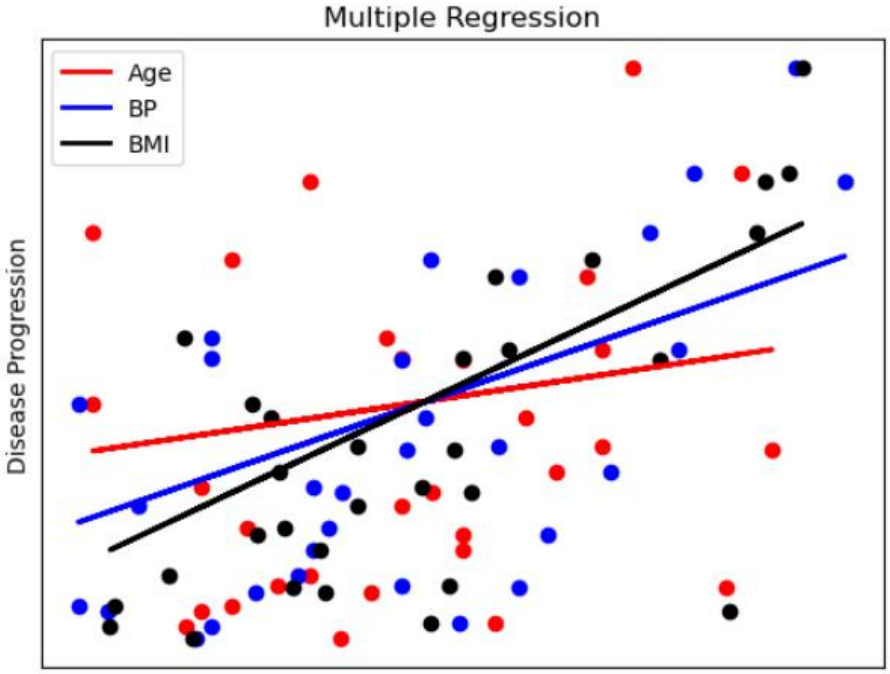
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
diabetes = datasets.load_diabetes()
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
# Description of the dataset
print(diabetes['DESCR'])
print(diabetes.feature_names)
diabetes_X = diabetes_X[:, np.newaxis, 0]
diabetes_X_train = diabetes_X[:-30]
diabetes_X_test = diabetes_X[-30:]
diabetes_y_train = diabetes_y[:-30]
diabetes_y_test = diabetes_y[-30:]
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)
diabetes_y_pred = regr.predict(diabetes_X_test)
print('Age')
print("Coefficients: \n", regr.coef_)
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test,
diabetes_y_pred))
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test,
diabetes_y_pred))
plt.scatter(diabetes_X_test, diabetes_y_test, color="red")
plt.plot(diabetes_X_test, diabetes_y_pred, color="red", linewidth=2,
label='Age')
plt.xticks(())
plt.yticks(())
plt.title('Multiple Regression')
#plt.xlabel('Age')
plt.ylabel('Disease Progression')
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
print(diabetes.feature_names)
```

```
diabetes_X = diabetes_X[:, np.newaxis, 3]
diabetes_X_train = diabetes_X[:-30]
diabetes_X_test = diabetes_X[-30:]
diabetes_y_train = diabetes_y[:-30]
diabetes_y_test = diabetes_y[-30:]
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)
print('BP')
print("Coefficients: \n", regr.coef_)
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test,
diabetes_y_pred))
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test,
diabetes_y_pred))
plt.scatter(diabetes_X_test, diabetes_y_test, color="blue")
plt.plot(diabetes_X_test, diabetes_y_pred, color="blue", linewidth=2,
label='BP')
plt.xticks(())
plt.yticks(())
plt.title('Multiple Regression')
plt.ylabel('Disease Progression')
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
print(diabetes.feature_names)
diabetes_X = diabetes_X[:, np.newaxis, 2]
diabetes_X_train = diabetes_X[:-30]
diabetes_X_test = diabetes_X[-30:]
diabetes_y_train = diabetes_y[:-30]
diabetes_y_test = diabetes_y[-30:]
regr = linear_model.LinearRegression()
regr.fit(diabetes_X_train, diabetes_y_train)
diabetes_y_pred = regr.predict(diabetes_X_test)
print('BMI')
print("Coefficients: \n", regr.coef_)
print("Mean squared error: %.2f" % mean_squared_error(diabetes_y_test,
```

```
diabetes_y_pred))
print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test,
diabetes_y_pred))
plt.scatter(diabetes_X_test, diabetes_y_test, color="black")
plt.plot(diabetes_X_test, diabetes_y_pred, color="black", linewidth=2,
label='BMI')
plt.xticks(())
plt.yticks(())
plt.title('Multiple Regression')
plt.ylabel('Disease Progression')
plt.legend()
plt.show()
print('Tanya Dhamija - 53004230009')
```

OUTPUT

```
      - sex
      - bmi     body mass index
      - bp      average blood pressure
      - s1      tc, total serum cholesterol
      - s2      ldl, low-density lipoproteins
      - s3      hdl, high-density lipoproteins
      - s4      tch, total cholesterol / HDL
      - s5      ltg, possibly log of serum triglycerides level
      - s6      glu, blood sugar level

Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times the square root of `n_samples` (i.e. the sum
of squares of each column totals 1).

Source URL:
https://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion), 407-499.
(https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)

['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
Age
Coefficients:
 [298.46194553]
Mean squared error: 5275.14
Coefficient of determination: -0.02
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
BP
Coefficients:
 [693.36333298]
Mean squared error: 3429.00
Coefficient of determination: 0.33
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
BMI
Coefficients:
 [941.43097333]
Mean squared error: 3035.06
Coefficient of determination: 0.41
```




```
Tanya Dhamija - 53004230009
```

Practical 5

Aim: Build a Classification Model

Theory: Classification is defined as the process of recognition, understanding, and grouping of objects and ideas into preset categories i.e. "sub-populations." With the help of these pre-categorized training datasets, classification in machine learning programs leverage a wide range of algorithms to classify future datasets into respective and relevant categories. Based on training data, the Classification algorithm is a Supervised Learning technique used to categorize new observations. In classification, a program uses the dataset or observations provided to learn how to categorize new observations into various classes or groups. For instance, 0 or 1, red or blue, yes or no, spam or not spam, etc. Targets, labels, or categories can all be used to describe classes. The Classification algorithm uses labelled input data because it is a supervised learning technique and comprises input and output information. A discrete output function (y) is transferred to an input variable in the classification process (x).

Types of Classification Algorithms

1. Logistic Regression: – It is one of the linear models which can be used for classification. It uses the sigmoid function to calculate the probability of a certain event occurring. It is an ideal method for the classification of binary variables.

2. K-Nearest Neighbours (KNN): – It uses distance metrics like Euclidean distance, Manhattan distance, etc. to calculate the distance of one data point from every other data point. To classify the output, it takes a majority vote from k nearest neighbours of each data point.

3. Decision Trees: – It is a non-linear model that overcomes a few of the drawbacks of linear algorithms like Logistic regression. It builds the classification model in the form of a tree structure that includes nodes and leaves. This algorithm involves multiple if else statements which help in breaking down the structure into smaller structures and eventually providing the final outcome. It can be used for regression as well as classification problems.

4. Random Forest: – It is an ensemble learning method that involves multiple decision trees to predict the outcome of the target variable. Each decision tree provides its own outcome. In the case of the classification problem, it takes the majority vote of these multiple decision trees to classify the final outcome. In the case of the regression problem, it takes the average of the values predicted by the decision trees.

5. Naïve Bayes: – It is an algorithm that is based upon Bayes' theorem. It assumes that any particular feature is independent of the inclusion of other features. i.e., They are not correlated to one another.

6. Support Vector Machine: – It represents the data points in multi-dimensional space. These data points are then segregated into classes with the help of hyperplanes. It plots an n-dimensional space for the n number of features in the dataset and then tries to create the hyperplanes such that it divides the data points with maximum margin.

CODE

```
import pandas as pd

col_names = ['pregnant','glucose','bp','skin','insulin','bmi','pedigree','age','label']

# Load Dataset

pima = pd.read_csv('diabetes.csv', header=None, names=col_names)

pima.head()
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
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

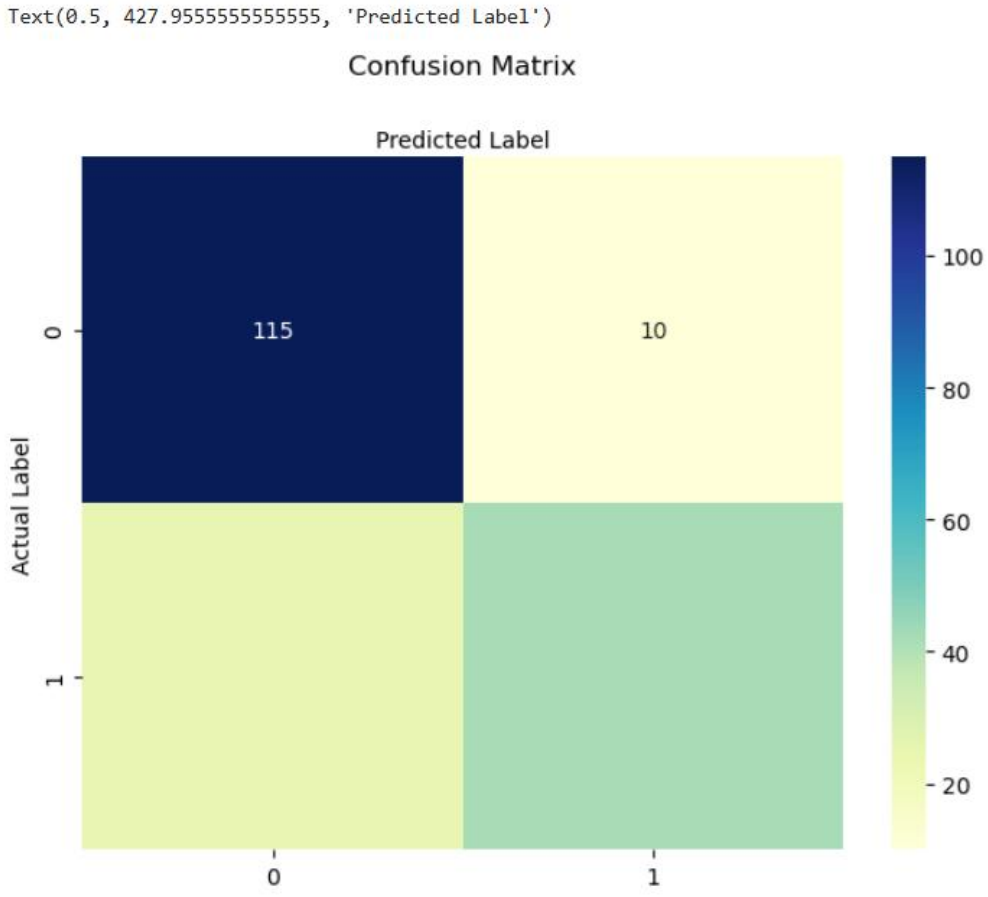
```
# Split dataset in features and target variable

feature_cols = ['pregnant','insulin','bmi','age','glucose','bp','pedigree']

X = pima[feature_cols]  # Features

Y = pima.label  # Target variable

# Split X and Y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.25,random_state=16)

# import the class

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(random_state=16)

# fit the model with data

logreg.fit(X_train,Y_train)

Y_pred = logreg.predict(X_test)

# import the metrics class

from sklearn import metrics

cnf_matrix = metrics.confusion_matrix(Y_test,Y_pred)

cnf_matrix
```

```
array([[115,  10],
       [ 25,  42]], dtype=int64)
```

```
# Visualizing confusion matrix using HeatMap
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
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)
# create HeatMap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap='YlGnBu', fmt='g')
ax.xaxis.set_label_position('top')
plt.tight_layout()
plt.title('Confusion Matrix', y=1.1)
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
```

OUTPUT

```
Text(0.5, 427.9555555555555, 'Predicted Label')
```

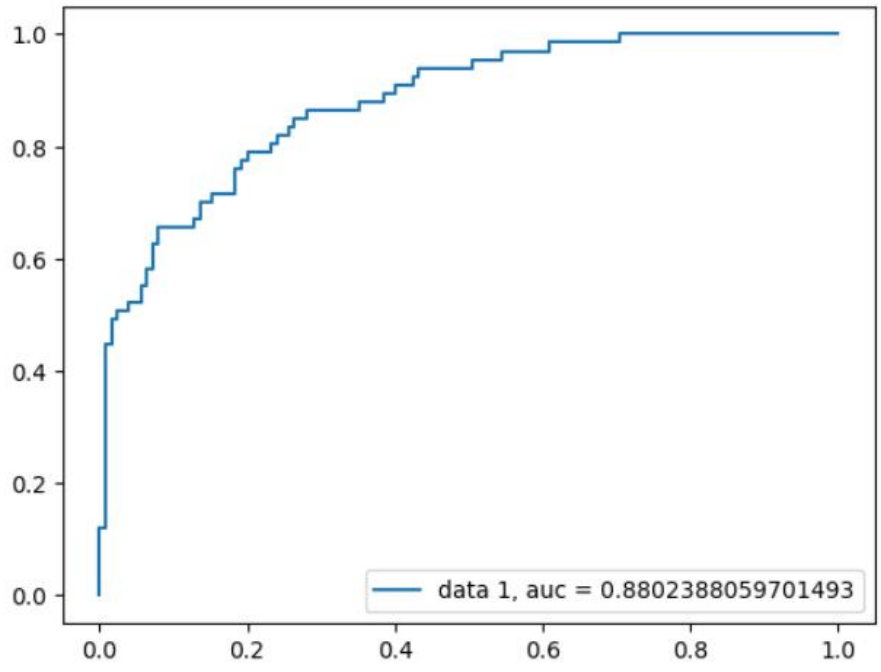


```
from sklearn.metrics import classification_report
target_names = ['Without Diabetes', 'With Diabetes']
print('Classification Report:-')
print(classification_report(Y_test,Y_pred,target_names=target_names))
```

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

Without Diabetes       0.82      0.92      0.87       125
   With Diabetes       0.81      0.63      0.71        67

        accuracy                           0.82       192
       macro avg       0.81      0.77      0.79       192
    weighted avg       0.82      0.82      0.81       192
```

```
# ROC Curve
Y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(Y_test, Y_pred_proba)
auc = metrics.roc_auc_score(Y_test, Y_pred_proba)
plt.plot(fpr, tpr, label = 'data 1, auc = '+str(auc))
plt.legend(loc=4)
plt.show()
print('Tanya Dhamija - 53004230009')
```

OUTPUT




```
Tanya Dhamija - 53004230009
```

Practical 6

Aim: Build a clustering model

Theory: 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, behaviour, 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 unlabelled dataset.

Types of Clustering algorithms:

1. K-Means Clustering: – It initializes a pre-defined number of k clusters and uses distance metrics to calculate the distance of each data point from the centroid of each cluster. It assigns the data points into one of the k clusters based on its distance.
2. Agglomerative Hierarchical Clustering (Bottom-Up Approach):– It considers each data point as a cluster and merges these data points on the basis of distance metric and the criterion which is used for linking these clusters.
3. Divisive Hierarchical Clustering (Top-Down Approach):– It initializes with all the data points as one cluster and splits these data points on the basis of distance metric and the criterion. Agglomerative and Divisive clustering can be represented as a dendrogram and the number of clusters to be selected by referring to the same.
4. DBSCAN (Density-based Spatial Clustering of Applications with Noise):– It is a density-based clustering method. Algorithms like K-Means work well on the clusters that are fairly separated and create clusters that are spherical in shape. DBSCAN is used when the data is in arbitrary shape and it is also less sensitive to the outliers. It groups the data points that have many neighboring data points within a certain radius.
5. OPTICS (Ordering Points to Identify Clustering Structure):– It is another type of densitybased clustering method and it is similar in process to DBSCAN except that it considers a few more parameters. But it is more computationally complex than DBSCAN. Also, it does not separate the data points into clusters, but it creates a reachability plot which can help in the interpretation of creating clusters.

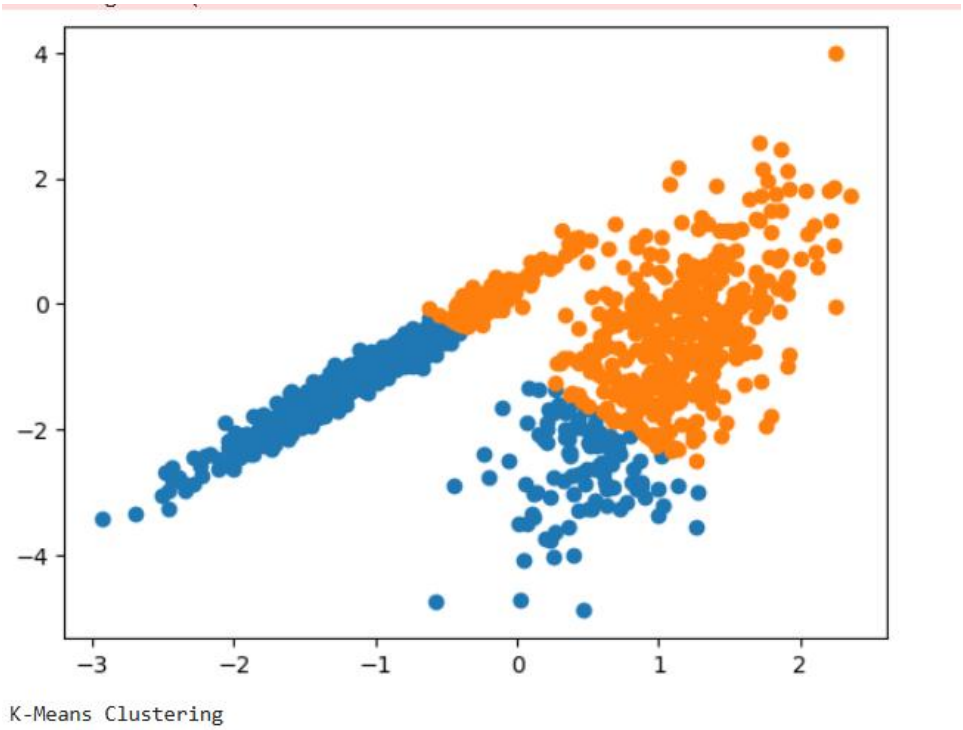
Code:

1) K-Means Clustering

```
# k-means clustering
from numpy import unique
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import KMeans
from matplotlib import pyplot
```

```
# define dataset
X, _ = make_classification(n_samples=1000, n_features=2, n_informative=2,
n_redundant=0, n_clusters_per_class=1, random_state=4)
# define the model
model = KMeans(n_clusters=2)
# fit the model
model.fit(X)
# assign a cluster to each example
yhat = model.predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
    # get row indexes for samples with this cluster
    row_ix = where(yhat == cluster)
    # create scatter of these samples
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
print("K-Means Clustering")
```

OUTPUT



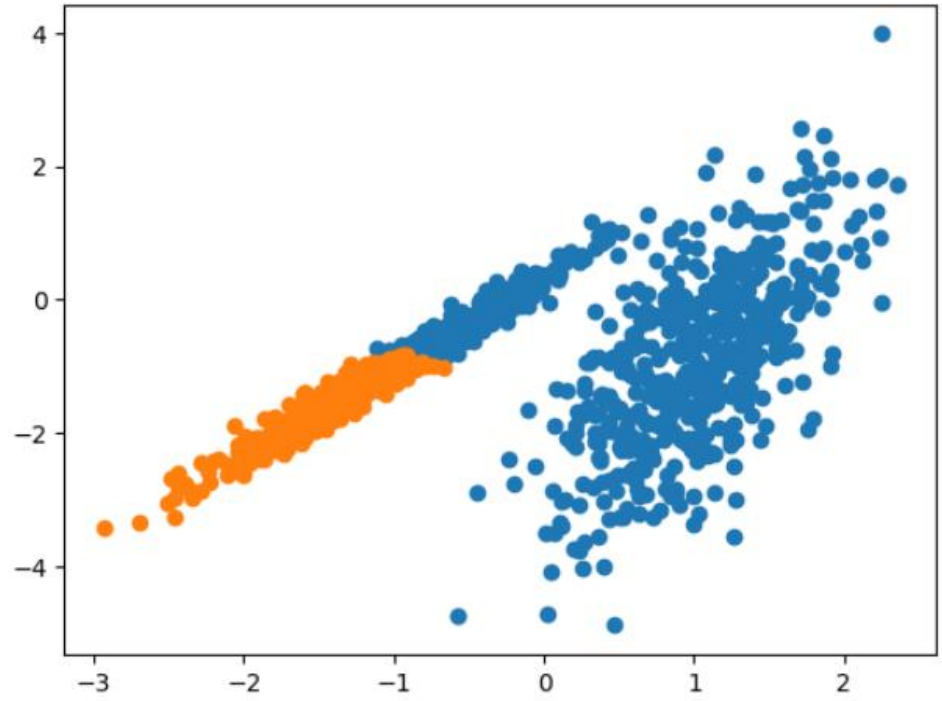
K-Means Clustering

2) Agglomerative Clustering

Code:

```
# Agglomerative clustering
from numpy import where
from sklearn.datasets import make_classification
from sklearn.cluster import AgglomerativeClustering
from matplotlib import pyplot
# define dataset
X, _ = make_classification(n_samples=1000, n_features=2,
n_informative=2, n_redundant=0, n_clusters_per_class=1,
random_state=4)
# define the model
model = AgglomerativeClustering(n_clusters=2)
# fit model and predict clusters
yhat = model.fit_predict(X)
# retrieve unique clusters
clusters = unique(yhat)
# create scatter plot for samples from each cluster
for cluster in clusters:
    # get row indexes for samples with this cluster
    row_ix = where(yhat == cluster)
    # create scatter of these samples
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
pyplot.show()
print('Tanya Dhamija - 53004230009')
print('Agglomerative Clustering')
```

OUTPUT




```
Tanya Dhamija - 53004230009
Agglomerative Clustering
```

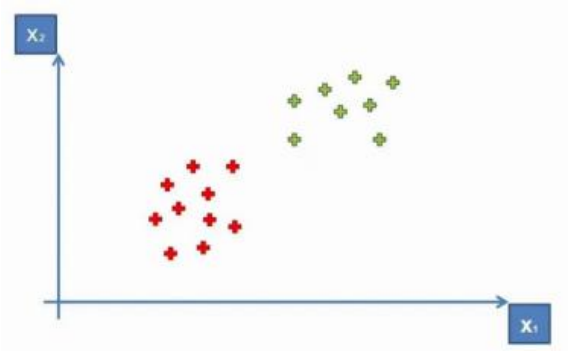
Practical 7

Aim: Implement SVM classification technique

Theory: SVM (Support Vector Machine) is a supervised machine learning algorithm. That's why training data is available to train the model. SVM uses a classification algorithm to classify a two-group problem. SVM focus on decision boundary and support vectors.

How SVM Works?

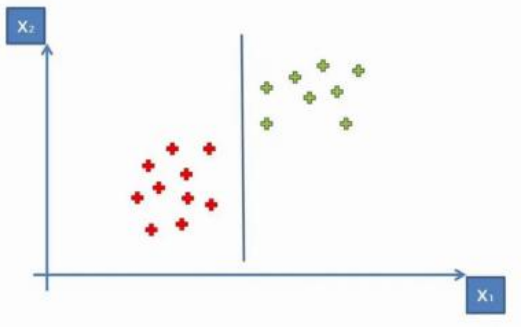
Here, we have two points in two-dimensional space, we have two columns x1 and x2. And we have some observations such as red and green, which are already classified. This is linearly separable data.



But, now how do we derive a line that separates these points? This means a separation or decision boundary is very important for us when we add new points. So to classify new points, we need to create a boundary between two categories, and when in the future we will add new points and we want to classify them, then we know where they belong. Either in a Green Area or Red Area.

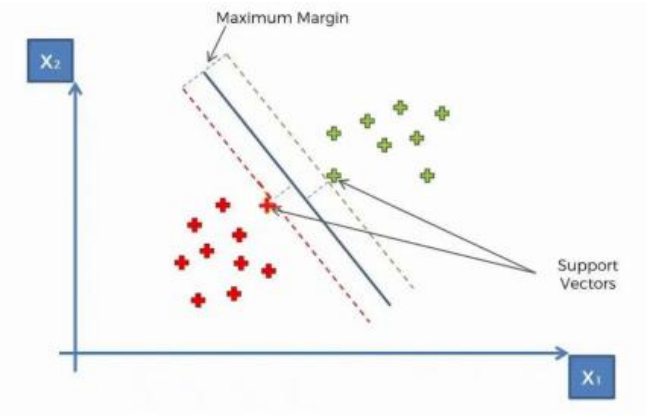
So how can we separate these points?

One way is to draw a vertical line between two areas, so anything on the left is Red and anything on the right is Green.



However, there is one more way, draw a horizontal line or diagonal line. You can create multiple diagonal lines, which achieve similar results to separate our points into two classes. But our main task is to find the optimal line or best decision boundary. And for this SVM is used. SVM finds the best decision boundary, which helps us to separate points into different spaces.SVM finds the best or optimal line through the

maximum margin, which means it has max distance and equidistance from both classes or spaces. The sum of these two classes has to be maximized to make this line the maximum margin.



These, two vectors are support vectors. In SVM, only support vectors are contributing. That's why these points or vectors are known as support vectors. Due to support vectors, this algorithm is called a Support Vector Algorithm (SVA).

In the picture, the line in the middle is a maximum margin hyperplane or classifier. In a two-dimensional plane, it looks like a line, but in a multi-dimensional, it is a hyperplane. That's how SVM works.

CODE

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Load Dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
dataset.head()
```

1]:

	15624510	Male	19	19000	0
0	15810944	Male	35	20000	0
1	15668575	Female	26	43000	0
2	15603246	Female	27	57000	0
3	15804002	Male	19	76000	0
4	15728773	Male	27	58000	0

```
# Split Dataset into X and Y
X = dataset.iloc[:, [2,3]].values
Y = dataset.iloc[:, 4].values
```

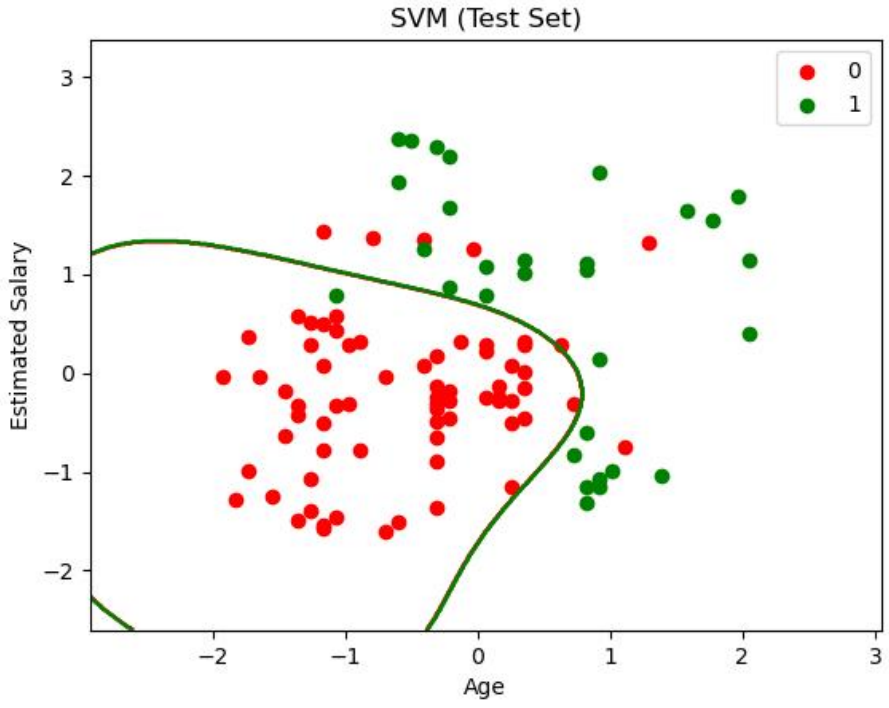
```
# Split the X and Y dataset into Training set and Testing set

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)

# Perform feature scaling- feature scaling helps us to normalize the data within a particular range

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)

# Fit SVM to the training set

from sklearn.svm import SVC

classifier = SVC(kernel='rbf', random_state=0)

classifier.fit(X_train, Y_train)

# Predict the test set results

Y_pred = classifier.predict(X_test)

# Make the confusion matrix

from sklearn.metrics import confusion_matrix, accuracy_score

cnf = confusion_matrix(Y_test, Y_pred)

print('Confusion Matrix:-')

print(cnf)

print('Accuracy Score:-')

accuracy_score(Y_test, Y_pred)
```

```
Confusion Matrix:-
[[63  7]
 [ 1 29]]
Accuracy Score:-
0.92
```

```
# Visualise the test set results

from matplotlib.colors import ListedColormap

X_set, Y_set = X_test, Y_test

X1, X2 = np.meshgrid(np.arange(start=X_set[:,0].min()-1, stop=X_set[:,0].max()+1,step=0.01),
```

```
                    np.arange(start=X_set[:,1].min()-1,
stop=X_set[:,1].max()+1,step=0.01))

plt.contour(X1, X2,
classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape),
alpha=0.75, cmap=ListedColormap(('red','green')))

plt.xlim(X1.min(),X1.max())

plt.ylim(X2.min(),X2.max())

for i,j in enumerate(np.unique(Y_set)):

    plt.scatter(X_set[Y_set==j,0], X_set[Y_set==j,1],
c=ListedColormap(('red','green'))(i),label=j)


plt.title('SVM (Test Set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

Print('Tanya - 53004230035')
```

OUTPUT



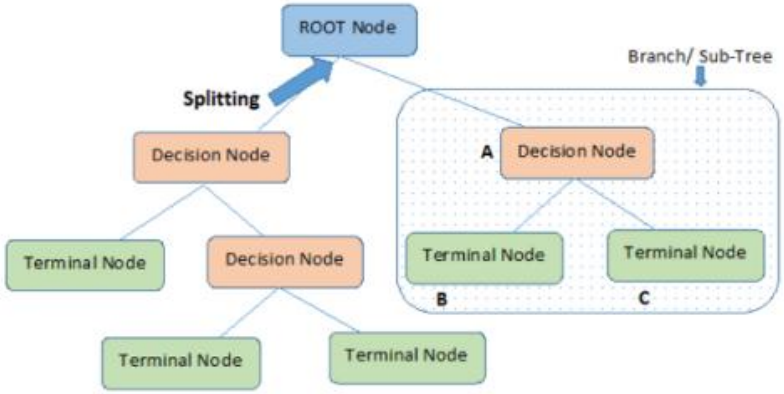

```
Tanya - 53004230009
```

Practical 8

Aim: Implement Decision Tree classification technique

Theory: A decision tree is a non-parametric supervised learning algorithm for classification and regression tasks. It has a hierarchical tree structure consisting of a root node, branches, internal nodes, and leaf nodes. Decision trees are used for classification and regression tasks, providing easy-to-understand models.

A decision tree is a hierarchical model used in decision support that depicts decisions and their potential outcomes, incorporating chance events, resource expenses, and utility. This algorithmic model utilizes conditional control statements and is non-parametric, supervised learning, useful for both classification and regression tasks. The tree structure is comprised of a root node, branches, internal nodes, and leaf nodes, forming a hierarchical, tree-like structure.



CODE

```
import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import train_test_split

from sklearn import metrics

col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi',
'pedigree', 'age', 'label']

pima = pd.read_csv('diabetes.csv',header=None, names=col_names)

pima.head()
```

1]:

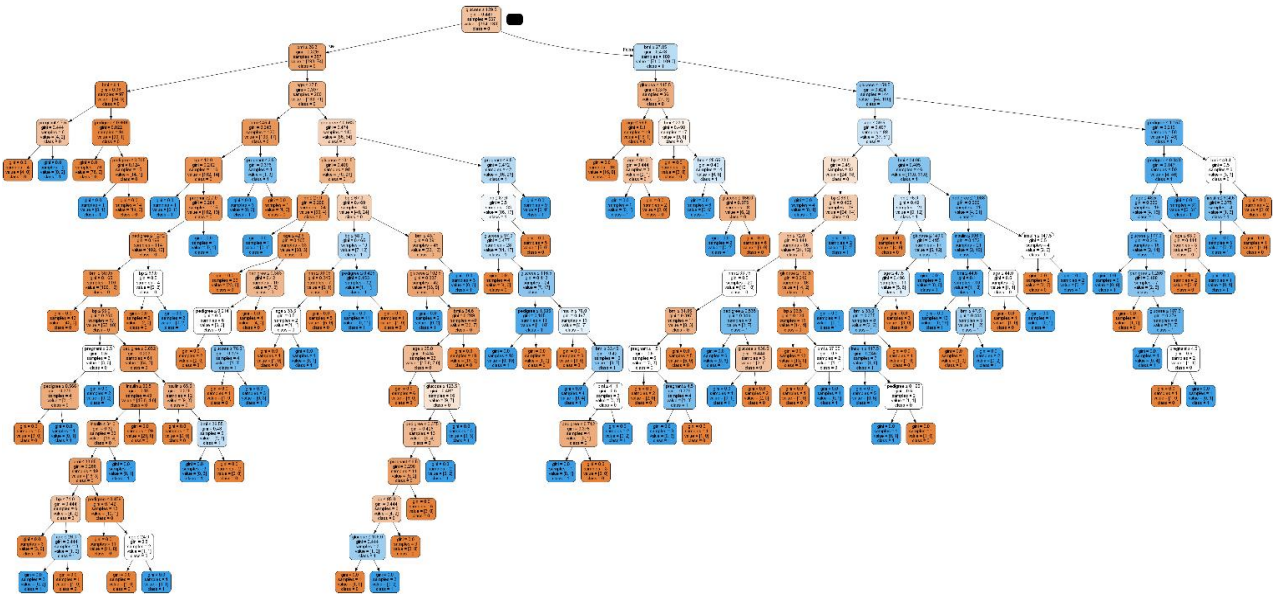
	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
# Split dataset into features and target variable

feature_cols = ['pregnant', 'insulin', 'bmi', 'age', 'glucose', 'bp', 'pedigree']

X = pima[feature_cols]

Y = pima.label

# Split dataset into training set and testing set

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.3,random_state=1)

# 70% training 30% testing

# Create Decision Tree Classifier Object

clf = DecisionTreeClassifier()

# Train Decision Tree Classifier

clf = clf.fit(X_train,Y_train)

#Predict the response for test dataset

Y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?

print('Accuracy:- ',metrics.accuracy_score(Y_test,Y_pred))

print('Tanya Dhamija - 53004230009')
```

```
Accuracy:-  0.7012987012987013
Tanya Dhamija - 53004230009
```

```
# Visualizing Decision Trees

from sklearn.tree import export_graphviz

from six import StringIO

from IPython.display import Image

import pydotplus

import os

os.environ["PATH"] += os.pathsep + r'C:\Users\DELL\anaconda3\Library\bin\graphviz'

dot_data = StringIO()

export_graphviz(clf, out_file=dot_data, filled=True, rounded=True, special_characters=True,

                feature_names=feature_cols, class_names=['0','1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

graph.write_png('diabetes.png')

Image(graph.create_png())
```

OUTPUT



Practical 9

AIM: Naïve Bayes Implementation

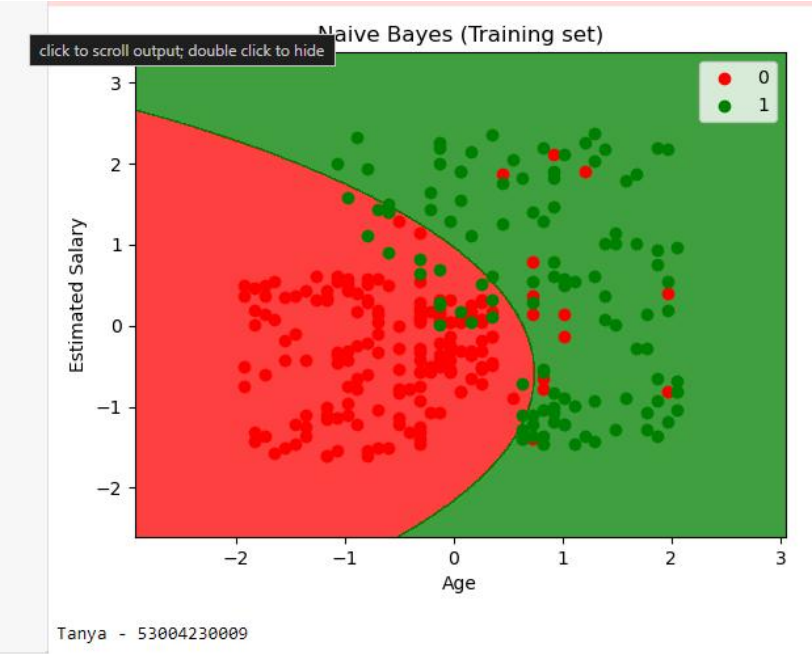
CODE

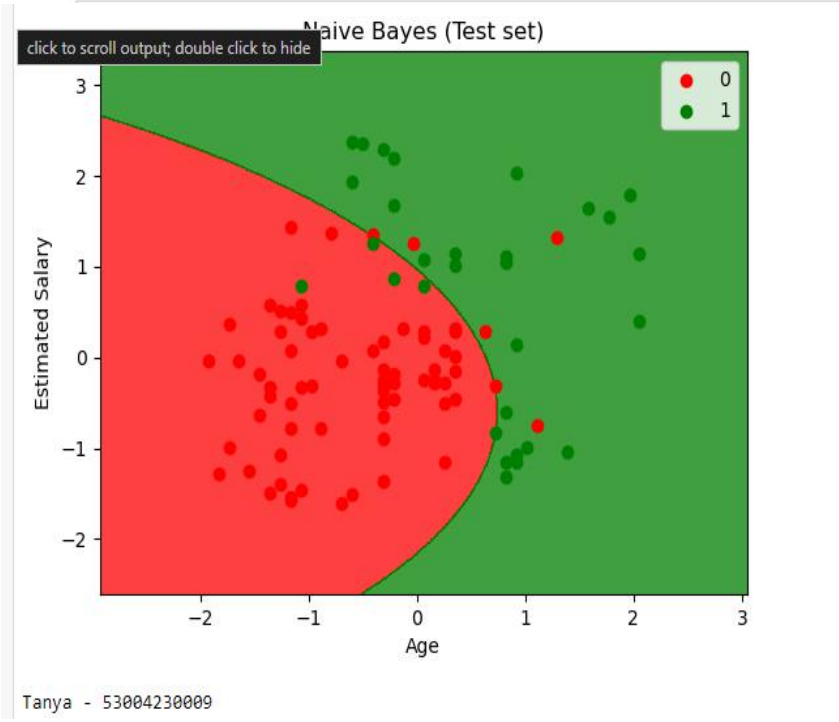
```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


# Importing the dataset

dataset = pd.read_csv('Social_Network_Ads.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 classifier 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 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1,        X2,        classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],

                c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

print('Nishi Jain-53004230036')

plt.show()

print('Tanya - 53004230009')


# Visualising the Test set results

from matplotlib.colors import ListedColormap

X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop =
X_set[:, 0].max() + 1, step = 0.01),

                     np.arange(start = X_set[:, 1].min() - 1, stop =
X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1,        X2,        classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):

    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```

```
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Naive Bayes (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
print('Tanya - 53004230009')
```

OUTPUT





INDEX

Practical No.		Practicals	Date	Page No.	Sign
1		Perform Geometric Transformation	04/03/24	1	
2		Perform Image Stitching	09/03/24	5	
3		Perform Camera Calibration	11/03/24	9	
4	A	Perform Face Detection	16/03/24	13	
	B	Perform Object Detection	18/03/24	17	
	C	Perform Pedestrian Detection	23/03/24	22	
	D	Perform Face Recognition	30/03/24	26	
5		Construct 3D Model from Images	01/04/24	29	
6		Implement Object Detection and Tracking from Video	08/04/24	33	
7		Perform Feature Extraction using RANSAC	15/04/24	37	
8		Perform Colorization	25/04/24	40	
9		Perform Text Detection and Recognition	02/05/24	44	
10		Perform Image Matting and Composting	09/05/24	47	

Practical 1

Aim: Perform Geometric Transformations

Theory:

Often, during image processing, the geometry of the image (such as the width and height of the image) needs to be transformed. This process is called geometric transformation, images are nothing but matrices.

Since images are matrices, if we apply an operation to the images (matrices) and we end up with another matrix, we call this a transformation. This basic idea will be used extensively to understand and apply various kinds of geometric transformations.

Here are the geometric transformations that we are performing in this practical:

Translation: Translation basically means shifting the object's location. It means shifting the object horizontally or vertically by some defined off-set (measured in pixels).
x' = x + A (Eq. 1)
y' = y + B (Eq. 2)
Here, let's say A = 100, B = 80 (Translation in x and y-direction respectively)
(x, y) – point in input image
(x', y') – point in output image
It means that each pixel is shifted 100 pixels in the x-direction and each pixel is shifted 80 pixels in the y-direction.

Rotation: As evident by its name, this technique rotates an image by a specified angle and by the given axis or point. It performs a geometric transform which maps the position of a point in current image to the output image by rotating it by the user-defined angle through the specified axis.
The points that lie outside the boundary of an output image are ignored.

Image scaling or resizing: Scaling means resizing an image which means an image is made bigger or smaller in x- or/and y-direction.

Affine transformation: An affine transformation is a transformation that preserves collinearity and the ratio of distances (for example – the midpoint of a line segment is still the midpoint even after the transformation))

The parallel lines in an original image will be parallel in the output image.

In general, an affine transformation is a composition of translations, rotations, shears, and magnifications.

Perspective transformation: Perspective transformation, also known as homograph, is a projective transformation that maps points in one plane to another while preserving straight lines. It can account for changes in perspective and orientation, making it suitable for applications like image stitching, rectifying images, and augmented reality. In computer vision, perspective transformation is often represented as a 3×3 matrix

called the homograph matrix. When applied to an image, this matrix can correct perspective distortions, align images from different viewpoints,
and create panoramas.

CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load on image
image = cv2.imread('left.jpg')
# Get image dimensions
height, width = image.shape[:2]
# Display the original image
plt.figure(figsize=(6,6))
plt.subplot(2,3,1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')
plt.show()
# Define the translation matrix
translation_matrix = np.float32([[1,0,300],[0,1,100]])
# Apply translation transformation
translated_image = cv2.warpAffine(image, translation_matrix,
(width,height))
# Display the translated image
plt.figure(figsize=(6,6))
plt.subplot(2,3,2)
plt.imshow(cv2.cvtColor(translated_image, cv2.COLOR_BGR2RGB))
plt.title('Translated Image')
plt.axis('off')
plt.show()
# Define the rotation matrix
rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), 60, 1)
# Apply rotation transformation
rotated_image = cv2.warpAffine(image, rotation_matrix, (width,height))
# Display the rotated image
plt.figure(figsize=(6,6))
plt.subplot(2,3,3)
plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
plt.title('Rotated Image')
```

```
plt.axis('off')
plt.show()
# Define the scaling matrix
scaling_matrix = np.float32([[0.8,0,0],[0,0.8,0]])
# Apply Scaling transformation
scaled_image = cv2.warpAffine(image, scaling_matrix, (int(width*0.5),
int(height*0.5)))
# Display the scaled image
plt.figure(figsize=(6,6))
plt.subplot(2,3,4)
plt.imshow(cv2.cvtColor(scaled_image, cv2.COLOR_BGR2RGB))
plt.title('Scaled Image')
plt.axis('off')
plt.show()
# Define affine transformation points
original_points = np.float32([[50,50], [200,50], [50,200]])
transformed_points = np.float32([[10,100], [200,50], [100,250]])
# Compute the affine transformation matrix
affine_matrix = cv2.getAffineTransform(original_points, transformed_points)
# Apply affine transformation
affine_transformed_image = cv2.warpAffine(image, affine_matrix,
(width,height))
# Display the affine transformed image
plt.figure(figsize=(6,6))
plt.subplot(2,3,5)
plt.imshow(cv2.cvtColor(affine_transformed_image, cv2.COLOR_BGR2RGB))
plt.title('Affine Transformed Image')
plt.axis('off')
plt.show()
# Define perspective transformation points
original_points = np.float32([[56,65], [368,52], [28,387], [389,390]])
transformed_points = np.float32([[0,0], [300,0], [0,300], [300,300]])
# Compute the perspective transformation matrix
perspective_matrix = cv2.getPerspectiveTransform(original_points,
transformed_points)
# Apply perspective transformation
perspective_transformed_image = cv2.warpPerspective(image,
perspective_matrix,
(width,height))
# Display the perspective transformed image
```

```
plt.figure(figsize=(6,6))
plt.subplot(2,3,6)
plt.imshow(cv2.cvtColor(perspective_transformed_image, cv2.COLOR_BGR2RGB))
plt.title('Perspective Transformed Image')
plt.axis('off')
plt.show()
```

OUTPUT

Practical 2

Aim: Perform Image Stitching

Theory:

Image stitching is the process of combining multiple images to create a single larger image, often referred to as a panorama. The images depict the same three-dimensional scene, and they must overlap in some regions. Image stitching aims to create a seamless transition between adjacent images while preserving the geometry and visual quality.

Image stitching is a multi-step technique that involves the following image-processing operations:

- **Feature detection:** Feature detection is the task of identifying distinctive and salient points in an image. These points, generally called keypoints, serve as landmarks for aligning the acquired images.

- **Feature matching:** Once features are detected, the next step is to find corresponding features between the images. The goal is to identify points in one image that correspond to the same real-world location in another image. For each detected feature, a descriptor is computed. The descriptor is a compact numerical representation of the local image information around the feature point. It captures the key characteristics of the feature and is used for matching.

- **Transformation estimation:** Once the pairs of matching features are identified, the transformations that align each image with a reference image are estimated. Such transformation is called homography.

- **Image warping:** This step involves applying the transformations found in the previous step to all the images, except the reference image. In this way, the images are aligned in the same reference system. The process of image warping involves applying a transformation to each pixel's coordinates in the original image and determining its location in the transformed image

- **Blending:** Uneven lighting conditions and exposure differences between the acquired images lead to visible seams in the final panorama. Image blending techniques allow us to mitigate the seam problem.

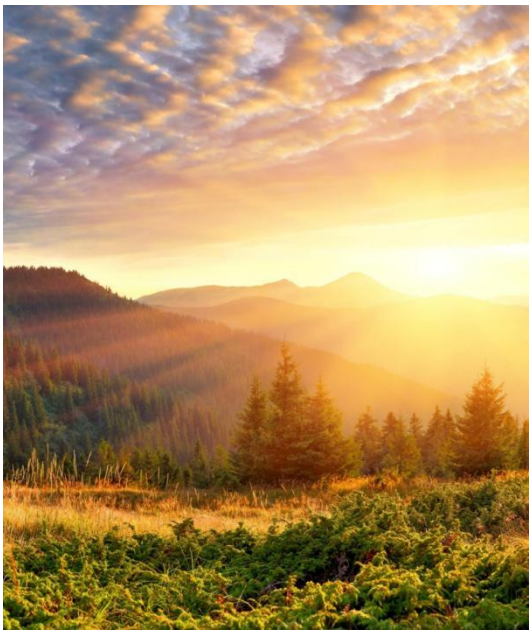
CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load images
img_ = cv2.imread('right1.jpg')
img1 = cv2.cvtColor(img_, cv2.COLOR_BGR2GRAY)
img = cv2.imread('left1.jpg')
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#Initialize SIFT detector
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
# Create a BFMatcher object with distance measurement cv2.NORM_L2
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck = False)
# Perfomr the matching between the SIFT descriptors of the images
matches = bf.knnMatch(des1,des2,k=2)
# Apply the ratio test to find good matches
good = []
for m,n in matches:
 if m.distance < 0.75*n.distance:
     good.append(m)
# Atleast 4 matches are to be there to find the homography
if len(good)>4:
 # prepare source and destination points
 src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1,1,2)
 dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1,1,2)
 # Compute Homography
 H, status = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
 # Use homography to warp image
 dst = cv2.warpPerspective(img_,H,
(img.shape[1]+img_.shape[1],img.shape[0]))
 # Convert warped image from BGR to RGB for matplotlib
 dst_rgb = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
 # Display the warped image
 plt.subplot(122), plt.imshow(dst_rgb), plt.title('Warped Image')
 plt.show()
 # Place the left image on the appropriate position
 dst[0:img.shape[0], 0:img.shape[1]] = img
```

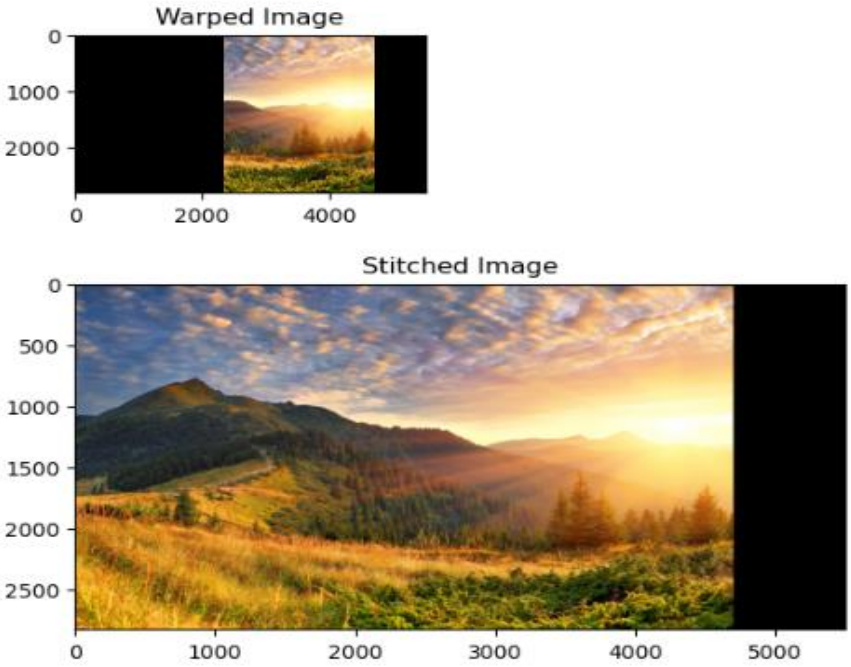
```
 # Convert the combined image from BGR to RGB for matplotlib
 combined_rgb = cv2.cvtColor(dst, cv2.COLOR_BGR2RGB)
    # Save the stitched image as output.jpg in the BGR format
 cv2.imwrite('output.jpg',dst)
 # Display the stitched image
 plt.imshow(combined_rgb)
 plt.title('Stitched Image')
 plt.show()
else:
 raise     AssertionError('Not     enough     matches     are     found     -
{}/{}'.format(len(good),4))
```

Output

Left **Right**



Wrapped



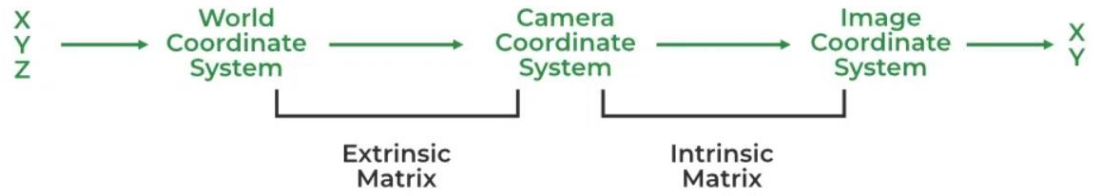
Practical 3

Aim: Perform Camera Calibration

Theory:

Images are captured by cameras, which use lenses to convert 3D objects from the real world into 2D images. However, because lenses are used during the transformation, some distortions are also introduced to the pictures. In order to prevent capturing distorted images, the camera needs to be calibrated, to accurately relate a 3D point in the real world to its matching 2D projection (pixel) in the image. Hence, camera calibration means determining the parameters of the camera to capture an undistorted image which is carried out by the function calibrateCamera() in Opencv.

The object in the real world exists in the World Coordinate System (3D) which when captured by the camera is viewed in Camera Coordinate System (3D). Finally, to project the captured image, the result is viewed in Image Coordinate System (2D).



Usually, two types of distortion happen in an image. First radial distortion, causes straight lines to look slightly twisted or bent when a camera photographs them. Second, Tangential distortion, which causes the picture to be extended slightly longer or skewed and causes the objects to appear closer or farther away than they actually are, is most commonly caused by the lens not being parallel to the imaging plane.

The two major types of parameters required for calibration are as follows:

- Extrinsic Parameters: It includes the orientation of the camera in the World Coordinate System by rotation and translation matrix.
- Intrinsic Parameters: It includes the lens system parameters such as focal length, optical center, aperture, field-of-view, resolution, etc

The calibration procedure includes solving the given matrices using basic geometric equation calculations. The equations are chosen depending on the calibration objects. Opencv, to date supports three types of objects for calibration:

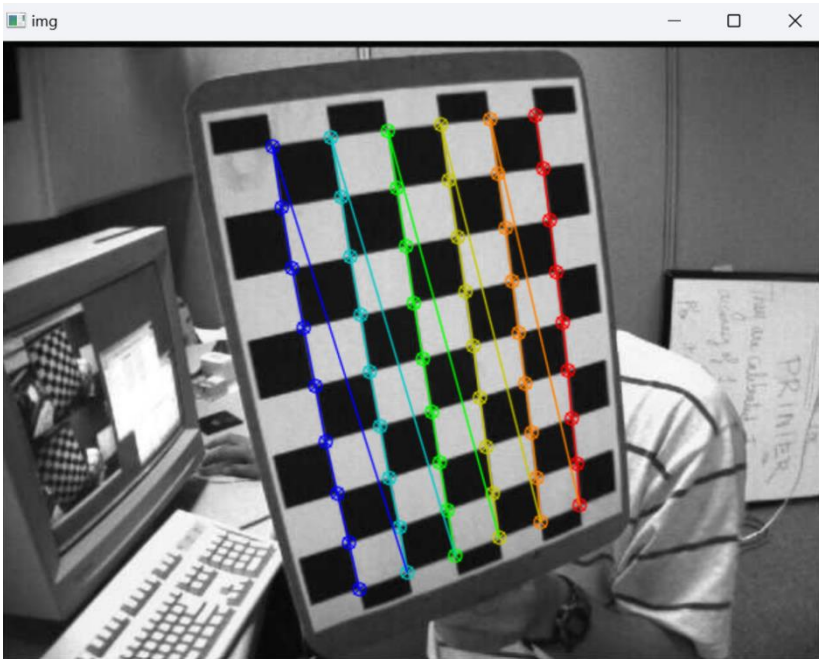
- Classical black-white chessboard
- Symmetrical circle pattern
- Asymmetrical circle pattern

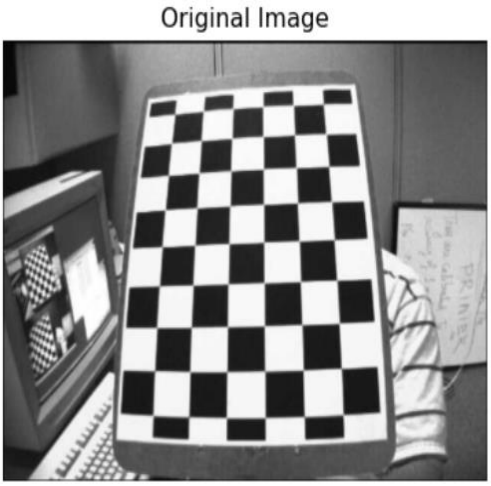
CODE

```
import numpy as np
import cv2
import glob
import matplotlib.pyplot as plt
# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*9,3), np.float32)
objp[:,:2] = np.mgrid[0:9,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob('left/*.jpg') #read a series of images

for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #Convert the image to gray

    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (9,6), None)

    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)
```

```
        corners2=cv2.cornerSubPix(gray, corners, (11,11), (-1,-1),
criteria) #refine the corner locations
        imgpoints.append(corners2)

        # Draw and display the corners
        cv2.drawChessboardCorners(img, (9,6), corners2, ret)
        cv2.imshow('img', img)
        cv2.waitKey(500)
cv2.destroyAllWindows()

#calibration
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1],None,None)

img = cv2.imread('left/left12.jpg')
h, w = img.shape[:2]
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1,
(w,h))

# undistort
dst = cv.undistort(img, mtx, dist, None, newcameramtx)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]

# Plot original and undistorted images

fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[0].axis('off')
ax[1].imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
ax[1].set_title('Undistorted Image')
ax[1].axis('off')
plt.show()
```

Output:





Practical 4

Aim: Perform the following:

A) Face Detection

Theory: There are some steps that how face detection operates, which are as follows:

Firstly, the image is imported by providing the location of the image. Then the picture is transformed from RGB to Grayscale because it is easy to detect faces in the grayscale. After that, the image manipulation used, in which the resizing, cropping, blurring and sharpening of the images done if needed.

The next step is image segmentation, which is used for contour detection or segments the multiple objects in a single image so that the classifier can quickly detect the objects and faces in the picture.

The next step is to use Haar-Like features algorithm, which is proposed by Voila and Jones for face detection. This algorithm used for finding the location of the human faces in a frame or image. All human faces share some universal properties of the human face like the eyes region is darker than its neighbour pixels and nose region is brighter than eye region.

The haar-like algorithm is also used for feature selection or feature extraction for an object in an image, with the help of edge detection, line detection, centre detection for detecting eyes, nose, mouth, etc. in the picture. It is used to select the essential features in an image and extract these features for face detection.

The next step is to give the coordinates of x, y, w, h which makes a rectangle box in the picture to show the location of the face or we can say that to show the region of interest in the image. After this, it can make a rectangle box in the area of interest where it detects the face.

i. Image Face Detection

CODE

```
!pip install open-cv python
# detect face from input image and save it on the disk.
import cv2
# Load the pre-trained Haar Cascade model for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
# load the image where you want to detect face
image_path = 'face.jpg' # path to your image
image = cv2.imread(image_path)
# convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5,
minSize=(30,30))
# draw rectangles around each face
for (x,y,w,h) in faces:
 cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),2)
# save the image with faces highlighted
output_path = 'faces_detected.jpg' # corrected file extension
cv2.imwrite(output_path, image)
print(f"Faces Detected: {len(faces)}. Output saved to {output_path}")
```

Output

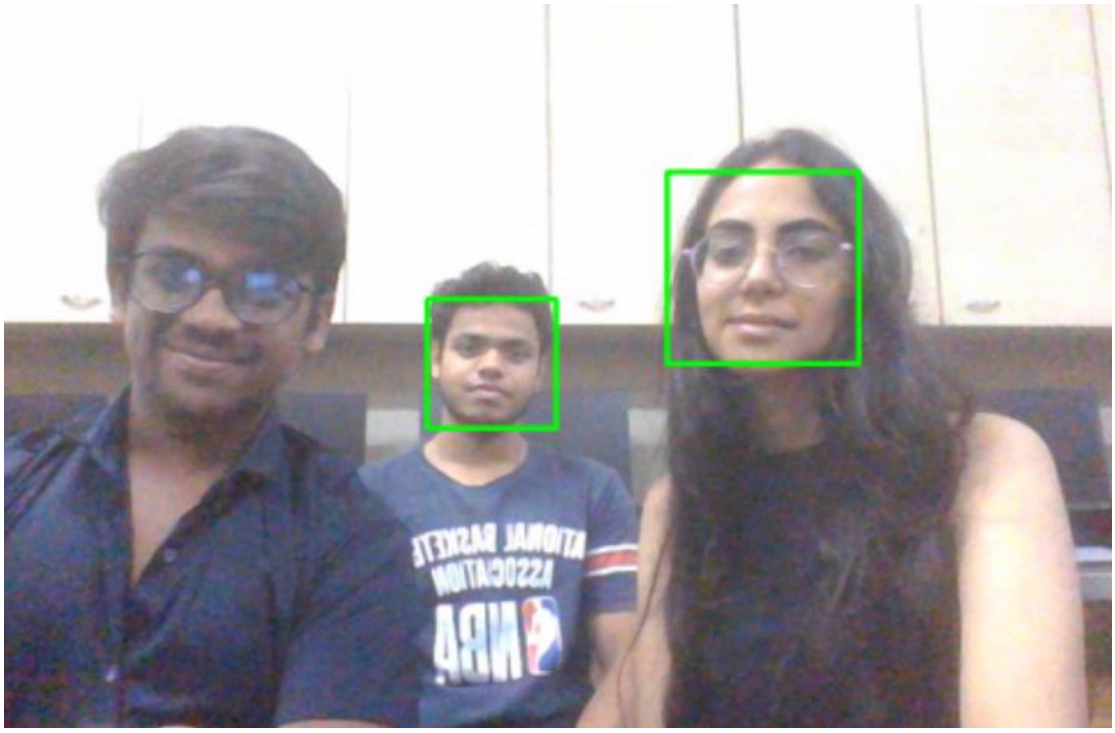
```
ERROR: Could not find a version that satisfies the requirement open-cv (from versions: none)
ERROR: No matching distribution found for open-cv
Faces Detected: 27. Output saved to faces_detected.jpg
```

ii. Live Face Detection

CODE

```
# detect faces and show it on screen
import cv2
import matplotlib.pyplot as plt
# initialize the Haar Cascade face detection model
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
# start capturing video from the camera (default camera is usually at
index 0)
cap = cv2.VideoCapture(0)
# Capture a single frame
ret, frame = cap.read()
if ret: # check if the frame was successfully captured
 # convert the captured frame to grayscale
 gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
 # detect faces in the grayscale frame
 faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5,
minSize=(30,30))
 # draw rectangles around the detected faces
 for (x,y,w,h) in faces:
     cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 2)
 # Display the resulting frame with faces highlighted using matplotlib
 plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
 plt.axis('off') # turn off axis labels
 plt.show()
# Release the capture
cap.release()
print('Number of faces detected: ',len(faces))
```

Output



```
Number of faces detected:  2
```

B) Object Detection

Theory: Object detection, within computer vision, involves identifying objects within images or videos. Object detection is merely to recognize the object with bounding box in the image, where in image classification, we can simply categorize(classify) that is an object in the image or not in terms of the likelihood (Probability). It is like object recognition, which involves identifying the object category, but object detection goes a step further and localizes the object within the image. Object detection applications can be performed using two different data analysis techniques:

A. Image Processing

In image processing, the algorithm analyses an image to detect objects based on certain features, such as edges or textures. This approach typically involves applying a series of image processing techniques, such as filtering, thresholding, and segmentation, to extract regions of interest in the image. These regions are then analyzed to determine whether they contain an object or not.

B. Deep Neural Network

In a deep neural network, the algorithm is trained on large, labelled datasets to detect objects. This approach involves using a convolutional neural network (CNN) architecture, which is a type of deep learning algorithm specifically designed for image analysis. The network learns to detect objects by analysing features of the image at different levels of abstraction, using multiple layers of artificial neurons to perform increasingly complex analyses. The output of the network is a bounding box around the detected object, along with a confidence score that represents how certain the network is that the object is present.

Categories of Object Detection:

- Single-shot detectors: Single-shot detectors combine localization and classification and they perform very quickly.
- Two-stage detectors: Two-stage detectors separate object localization and classification in the head, generally returns higher localization accuracy.

Object detection algorithms and architectures:

R-CNN (region-based convolutional neural network) is a two-stage detector that uses a method called region proposals to generate 2,000 region predictions per image. R-CNN then warps the extracted regions to a uniform size and runs those regions through separate networks for feature extraction and classification.

YOLO (You Only Look Once) is a family of single-stage detection architectures based in Darknet, an open-source CNN framework. YOLO's speed makes it preferable for real-time object detection and has earned it the common descriptor of state-of-the-art object detector. YOLO makes less than 100 bounding box predictions per image, it also produces less background false positives, although it has a higher localization error, generally focuses on speed and accuracy.

CODE

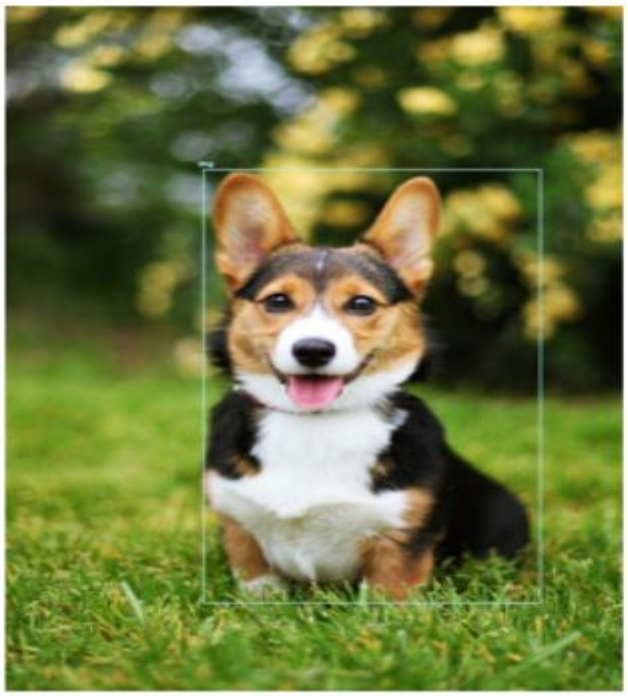
```
import argparse
import numpy as np
import cv2
import matplotlib.pyplot as plt

args = argparse.Namespace(
    image="C:/dog.jpg",
    weights="yolov3.weights",
    config="yolov3.cfg",
    classes="yolov3.txt"
)
def get_output_layers(net):
    layer_names = net.getLayerNames()
    try:
        output_layers = [layer_names[i-1] for i in
net.getUnconnectedOutLayers()]
    except:
        output_layers = [layer_names[i-1] for i in
net.getUnconnectedOutLayers()]
    return output_layers

def draw_prediction(img, class_id, confidence, x, y, x_plus_w, y_plus_h):
    label = str(classes[class_id])
    color = COLORS(class_id)
    cv2.rectangle(img, (x,y),(x_plus_w, y_plus_h), label, color,2)
    cv2.putText(img, label,  (x-10,y-10),cv2.FONT_HERSHEY_COMPLEX, color,
2)
# read input image
image = cv2.imread(args.image)

Width = image.shape[1]
Height = image.shape[0]
scale = 0.00392
classes = None

# read class names from text file
classes = None
with open(args.classes, 'r') as f:
```

```
    classes = [line.strip() for line in f.readlines()]

# generate different colors for different classes
COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

# read pre-trained model and config file
net = cv2.dnn.readNet(args.weights, args.config)

# create input blob
blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True,
crop=False)

# set input blob for the network
net.setInput(blob)

# function to get the output layer names
# in the architecture

# function to draw bounding box on the detected object with class name
def draw_bounding_box(img, class_id, confidence, x, y, x_plus_w, y_plus_h):

    label = str(classes[class_id])

    color = COLORS[class_id]

    cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)

    cv2.putText(img, label, (x-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
color, 2)

# run inference through the network
# and gather predictions from output layers
outs = net.forward(get_output_layers(net))

# initialization
class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4
```

```
# for each detetion from each output layer
# get the confidence, class id, bounding box params
# and ignore weak detections (confidence < 0.5)
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])



# apply non-max suppression
indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold,
nms_threshold)

# go through the detections remaining
# after nms and draw bounding box
for i in indices:
    try:
        box=boxes[i]
    except:
        i=i[0]
        box=boxes[i]

    x = box[0]
    y = box[1]
    w = box[2]
    h = box[3]
```

```
    draw_bounding_box(image, class_ids[i], confidences[i], round(x),
round(y), round(x+w), round(y+h))

# display output image
#cv2.imshow("object detection", image)

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')  # turn off axis labels
plt.show()
# wait until any key is pressed
#cv2.waitKey()

 # save output image to disk
#cv2.imwrite("object-detection.jpg", image)

# release resources
#cv2.destroyAllWindows()
```

OUTPUT



C) Pedestrian detection

Theory:

Pedestrian detection is a very important area of research because it can enhance the functionality of a pedestrian protection system in Self Driving Cars. We can extract features like head, two arms, two legs, etc, from an image of a human body and pass them to train a machine learning model. After training, the model can be used to detect and track humans in images and video streams. However, OpenCV has a built-in method to detect pedestrians. It has a pre-trained haarcascade fullbody model to detect pedestrians in images and video streams.

Haar Cascade: Haar cascade is an algorithm that can detect objects in images, irrespective of their scale in image and location.

This algorithm is not so complex and can run in real-time. We can train a haar-cascade detector to detect various objects like cars, bikes, buildings, fruits, etc.

Haar cascade uses the cascading window, and it tries to compute features in every window and classify whether it could be an object.

There are some pre-trained haar cascade models like:

- Human face detection
- Eye detection
- Nose / Mouth detection
- Vehicle detection
- Full body detection

In this practical we will be using the full body detection pre-trained model.

Method 1
Code for jupyter

CODE

```
import cv2
import matplotlib.pyplot as plt
pedestrian_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_fullbody.xml')
video_path = 'C:/Users/NISHA/Desktop/photos/video.mp4'
cap = cv2.VideoCapture(video_path)
if not cap.isOpened():
    print("Error: Unable to open the video")
    exit()
while True:
    ret, frame = cap.read()
    if not ret:
        break
    gray_frame = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    # Detect pedestrians in the grayscale frame
    pedestrians = pedestrian_cascade.detectMultiScale(gray_frame, scaleFactor=1.1, minNeighbors=5, minSize=(30,30))
    # Draw rectangle around the detected pedestrians
    for (x,y,w,h) in pedestrians:
        cv2.rectangle(frame, (x,y), (x+w,y+h), (0,255,0), 2)
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.axis('off')   # Turn off the axis
    plt.show()
cap.release()
```

OUTPUT

Method 2:
Code for ideal script

CODE

```
import cv2
import matplotlib.pyplot as plt
import matplotlib.animation as animation

#Initialize the Haar Cascade pedestrian detection model

pedestrian_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_fullbody.xml')

# Load the video
video_path = 'D:/VIDEO.mp4'
cap= cv2.VideoCapture(video_path)

#check if the video is opened successfully
if not cap.isOpened():
    print("Error: Unable to open the video.")
    exit()

#Function to detect pedestrians and draw bounding boxes
def detect_pedestrians(frame):
    #Convert the frame to grayscale
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    #Detect pedestrians in the grayscale frame
    pedestrians = pedestrian_cascade.detectMultiScale(gray_frame,
scaleFactor=1.05,minNeighbors=5,minSize=(30,30))

    #Draw rectangles around the detected pedestrains
    for(x,y,w,h) in pedestrians:
        cv2.rectangle( frame,(x,y),(x+w,y+h),(0,255,0),2)
    return frame
fig,ax=plt.subplots()
```

```
#Function to update the animation
def update(frame):
    #Read a frME FROM THE VIDEO
    ret,frame = cap.read()
    if not ret:
        ani.event_source.stop()
        return

    #Detect pedestrians and draw boundng boxes
    frame_with_pedestrains=detect_pedestrians(frame)

    ax.clear()
    ax.imshow(cv2.cvtColor(frame_with_pedestrains, cv2.COLOR_BGR2RGB))
    ax.axis('off')

#create the animation
ani=animation.FuncAnimation(fig,update,interval=50)

#Display the animation
plt.show()
#Release the video capture object
cap.release
```

OUTPUT

D) Face Recognition

Theory:

Facial recognition works in three steps: detection, analysis, and recognition.

Detection

Detection is the process of finding a face in an image. Enabled by computer vision, facial recognition can detect and identify individual faces from an image containing one or many people's faces. It can detect facial data in both front and side face profiles.

Computer vision

Machines use computer vision to identify people, places, and things in images with accuracy at or above human levels and with much greater speed and efficiency. Using complex artificial intelligence (AI) technology, computer vision automates extraction, analysis, classification, and understanding of useful information from image data. The image data takes many forms, such as the following:

- Single images
- Video sequences
- Views from multiple cameras
- Three-dimensional data
-

Analysis

The facial recognition system then analyzes the image of the face. It maps and reads face geometry and facial expressions. It identifies facial landmarks that are key to distinguishing a face from other objects. The facial recognition technology typically looks for the following:

- Distance between the eyes
- Distance from the forehead to the chin
- Distance between the nose and mouth
- Depth of the eye sockets
- Shape of the cheekbones
- Contour of the lips, ears, and chin

The system then converts the face recognition data into a string of numbers or points called a faceprint. Each person has a unique faceprint, similar to a fingerprint. The information used by facial recognition can also be used in reverse to digitally reconstruct a person's face.

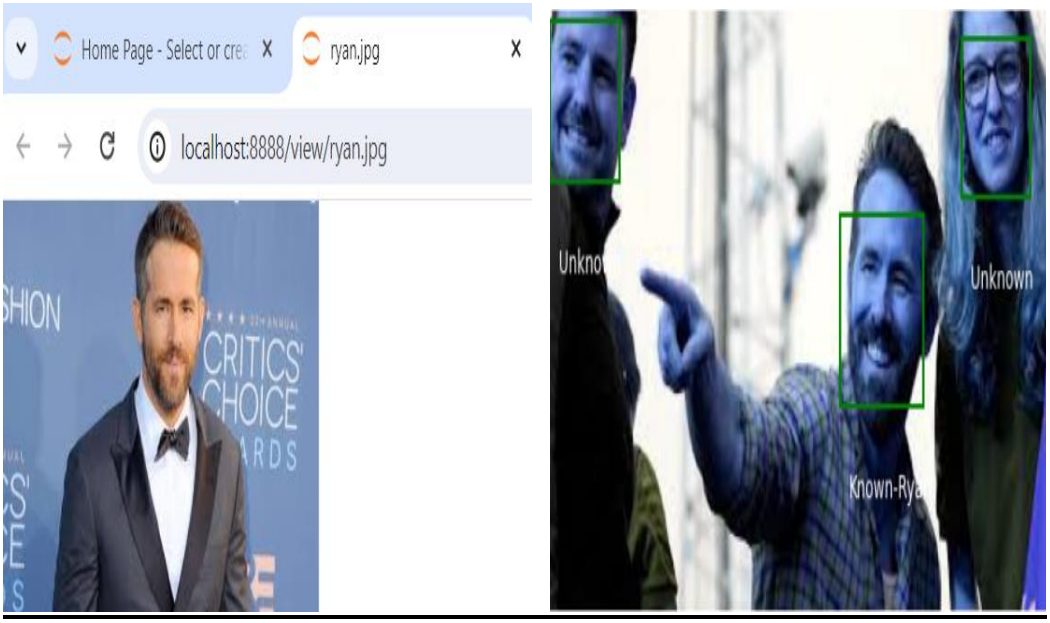
Recognition

Facial recognition can identify a person by comparing the faces in two or more images and assessing the likelihood of a face match. For example, it can verify that the face shown in a selfie taken by a mobile camera matches the face in an image of a government-issued ID like a driver's license or passport, as well as verify that the face shown in the selfie does not match a face in a collection of faces previously captured.

CODE

```
!pip install dlib-19.22.99-cp38-cp38-win_amd64.whl
!pip install face_recognition
import face_recognition
from matplotlib.patches import Rectangle
import matplotlib.pyplot as plt

known_image = face_recognition.load_image_file('ryan.jpg')
known_face_encoding = face_recognition.face_encodings(known_image)[0]

unknown_image = face_recognition.load_image_file('test_ryan.jpg')
unknown_face_locations = face_recognition.face_locations(unknown_image)
unknown_face_encodings = face_recognition.face_encodings(unknown_image,
unknown_face_locations)

unknown_image_rgb = unknown_image[:, :, ::-1]

fig,ax= plt.subplots(figsize=(8,6))
ax.imshow(unknown_image_rgb)

for (top, right, bottom, left), unknown_face_encoding in
zip(unknown_face_locations, unknown_face_encodings):
    results=face_recognition.compare_faces([known_face_encoding],
unknown_face_encoding)
    if results[0]:
        name="Known-Ryan"
    else:
        name="Unknown"

    ax.add_patch(Rectangle((left, top), right - left, bottom-top,
fill=False, color='green', linewidth=2))
    ax.text(left+6, bottom+25, name, color='white', fontsize=12)

    plt.axis('off')
```

OUTPUT


Home Page - Select or cre
ryan.jpg
localhost:8888/view/ryan.jpg
Unknown
Known-Rya

Practical 5

Aim: Construct 3D model from Images:
Theory:

Transforming a 2D image into a 3D space using OpenCV refers to the process of converting a two-dimensional image into a three-dimensional spatial representation using the Open Source Computer Vision Library (OpenCV). OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library written in C++ with interfaces for Python, Java, and MATLAB. It provides a wide range of features for processing and evaluating pictures and movies. This transformation involves inferring the depth information from the 2D image, typically through techniques such as stereo vision, depth estimation, or other computer vision algorithms, to create a 3D model with depth perception. This process enables various applications such as 3D reconstruction, depth sensing.

Modules Needed

- Matplotlib: It is a plotting library for Python programming it serves as a visualization utility library, Matplotlib is built on NumPy arrays, and designed to work with the broader SciPy stack.
- Numpy: It is a general-purpose array-processing package. It provides a high-performance multidimensional array and matrices along with a large collection of high-level mathematical functions.
- mpl_toolkits: It provides some basic 3d plotting (scatter, surf, line, mesh) tools.

SIFT Algorithm

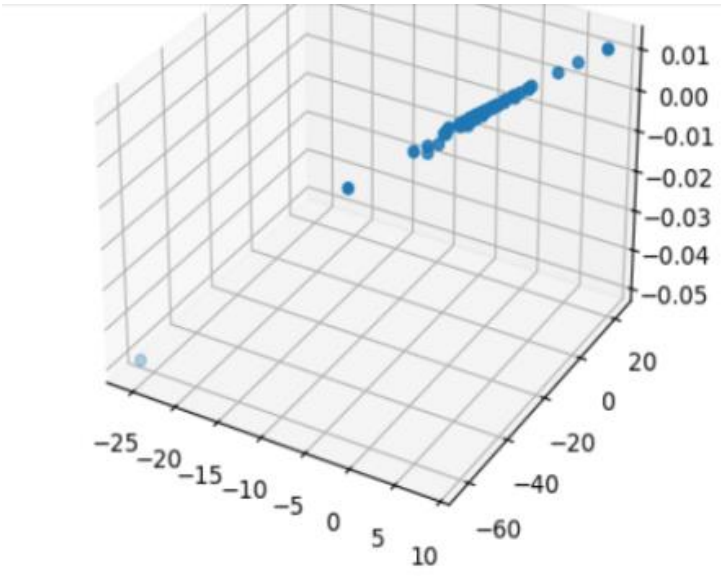
The SIFT (Scale-Invariant Feature Transform) algorithm is a computer vision technique used for feature detection and description. It detects distinctive key points or features in an image that are robust to changes in scale, rotation, and affine transformations. SIFT (scale invariant feature transform) works by identifying key points based on their local intensity extrema and computing descriptors that capture the local image information around those key points. These descriptors can then be used for tasks like image matching, object recognition, and image retrieval. SIFT algorithm helps locate the local features in an image, commonly known as the ‘keypoints’ of the image. These keypoints are scale & rotation invariants that can be used for various computer vision applications, like image matching, object detection, scene detection, etc.

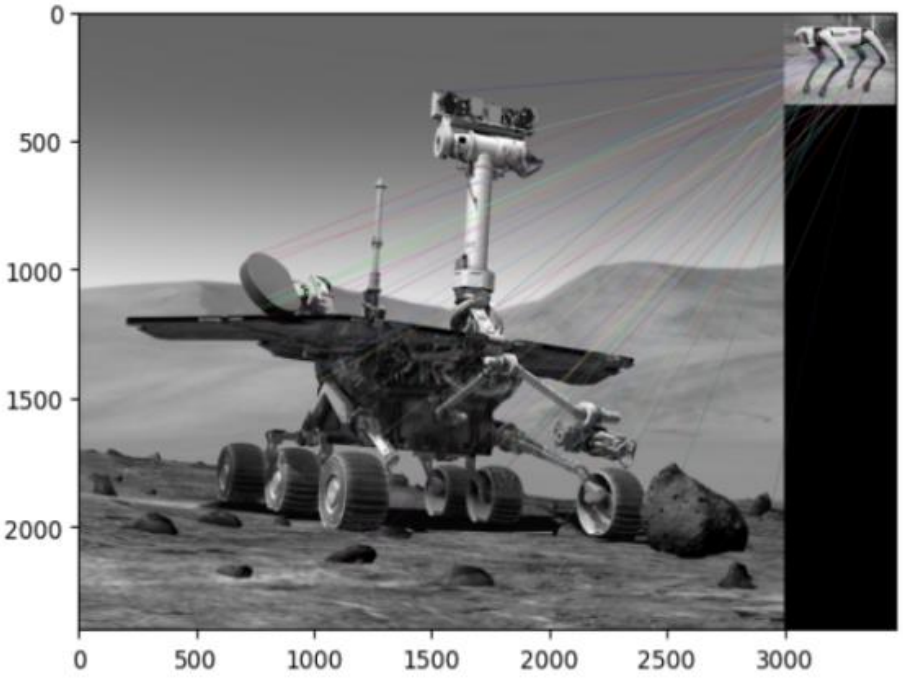
CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def extract_keypoints_and_descriptors(image):
    sift = cv2.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(image, None)
    return keypoints, descriptors
def match_keypoints(desc1, desc2):
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
    matches = bf.match(desc1, desc2)
    matches = sorted(matches, key=lambda x: x.distance)
    return matches
def draw_matches(img1, kp1, img2, kp2, matches):
    return cv2.drawMatches(img1, kp1, img2, kp2, matches[:50], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

def reconstruct_3d_points(K, kp1, kp2, matches, E):
    points1 = np.float32([kp1[m.queryIdx].pt for m in matches])
    points2 = np.float32([kp2[m.trainIdx].pt for m in matches])
    _, R, t, mask = cv2.recoverPose(E, points1, points2, K)

    # Triangulate points
    P1 = np.hstack((np.eye(3, 3), np.zeros((3, 1))))
    P2 = np.hstack((R, t))

    points_4d_hom = cv2.triangulatePoints(P1, P2, points1.T, points2.T)
    points_4d = points_4d_hom / np.tile(points_4d_hom[-1, :], (4, 1))
    points_3d = points_4d[:3, :].T
    return points_3d
# Load two images of a scene
img1 = cv2.imread('3d1.jpg', cv2.IMREAD_GRAYSCALE)
img2 = cv2.imread('3d2.jpg', cv2.IMREAD_GRAYSCALE)

# Placeholder values for the camera matrix K (Students you should change it
based on your camera parameters.
fx = 1000  # Focal length in pixels
```

```
fy = 1000  # Focal length in pixels
cx = img1.shape[1] / 2  # Optical center X coordinate, assuming center of
the image
cy = img1.shape[0] / 2  # Optical center Y coordinate, assuming center of
the image
K = np.array([[fx, 0, cx],
              [0, fy, cy],
              [0, 0, 1]], dtype=float)
# Extract keypoints and descriptors
kp1, desc1 = extract_keypoints_and_descriptors(img1)
kp2, desc2 = extract_keypoints_and_descriptors(img2)

# Match keypoints
matches = match_keypoints(desc1, desc2)
# Estimate the essential matrix
points1 = np.float32([kp1[m.queryIdx].pt for m in matches])
points2 = np.float32([kp2[m.trainIdx].pt for m in matches])
F, mask = cv2.findFundamentalMat(points1, points2, cv2.FM_RANSAC)
E = K.T @ F @ K

# Reconstruct 3D points
points_3d = reconstruct_3d_points(K, kp1, kp2, matches, E)
# Visualize 3D points
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(points_3d[:, 0], points_3d[:, 1], points_3d[:, 2])
plt.show()

# Display matched keypoints
matched_img = draw_matches(img1, kp1, img2, kp2, matches)
plt.imshow(matched_img)
plt.show()
```

OUTPUT


0.01
0.00
−0.01
−0.02
−0.03
−0.04
−0.05
20
0
−20
−40
−60
−25
−20
−15
−10
−5
0
5
10


0
500
1000
1500
2000
0
500
1000
1500
2000
2500
3000

Practical 6

Aim: Implement object detection and tracking from video

Theory: Object detection, within computer vision, involves identifying objects within images or videos. Object detection is merely to recognize the object with bounding box in the image, where in image classification, we can simply categorize(classify) that is an object in the image or not in terms of the likelihood (Probability). It is like object recognition, which involves identifying the object category, but object detection goes a step further and localizes the object within the image. Object detection applications can be performed using two different data analysis techniques:

A. Image Processing

In image processing, the algorithm analyses an image to detect objects based on certain features, such as edges or textures. This approach typically involves applying a series of image processing techniques, such as filtering, thresholding, and segmentation, to extract regions of interest in the image. These regions are then analyzed to determine whether they contain an object or not.

B. Deep Neural Network

In a deep neural network, the algorithm is trained on large, labelled datasets to detect objects. This approach involves using a convolutional neural network (CNN) architecture, which is a type of deep learning algorithm specifically designed for image analysis. The network learns to detect objects by analysing features of the image at different levels of abstraction, using multiple layers of artificial neurons to perform increasingly complex analyses. The output of the network is a bounding box around the detected object, along with a confidence score that represents how certain the network is that the object is present.

Categories of Object Detection:

- Single-shot detectors: Single-shot detectors combine localization and classification and they perform very quickly.
- Two-stage detectors: Two-stage detectors separate object localization and classification in the head, generally returns higher localization accuracy.
-

Object detection algorithms and architectures

There are a number of machine learning approaches to object detection tasks. Examples include the Viola-Jones framework8 and the histogram of oriented gradients. Recent object detection research and development, however, has focused largely on convolutional neural networks (CNNs).

R-CNN (region-based convolutional neural network) is a two-stage detector that uses a method called region proposals to generate 2,000 region predictions per image. R-CNN then warps the extracted regions to a uniform size and runs those regions through separate networks for feature extraction and classification.

YOLO (You Only Look Once) is a family of single-stage detection architectures based in Darknet, an open-source CNN framework. YOLO's speed makes it preferable for real-time object detection and has earned it the common descriptor of state-of-the-art

object detector. YOLO makes less than 100 bounding box predictions per image, it also produces less background false positives, although it has a higher localization error, generally focuses on speed and accuracy.

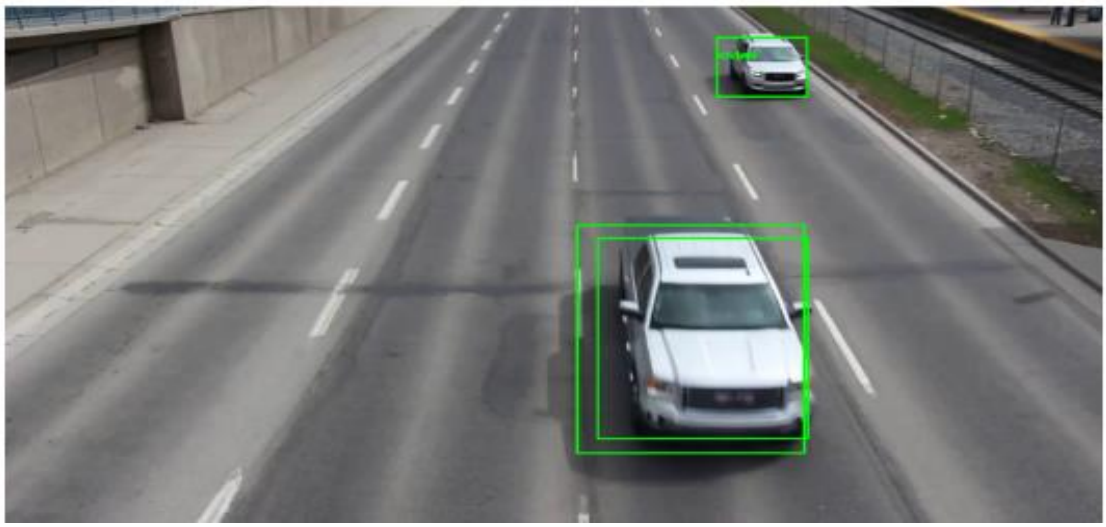
CODE

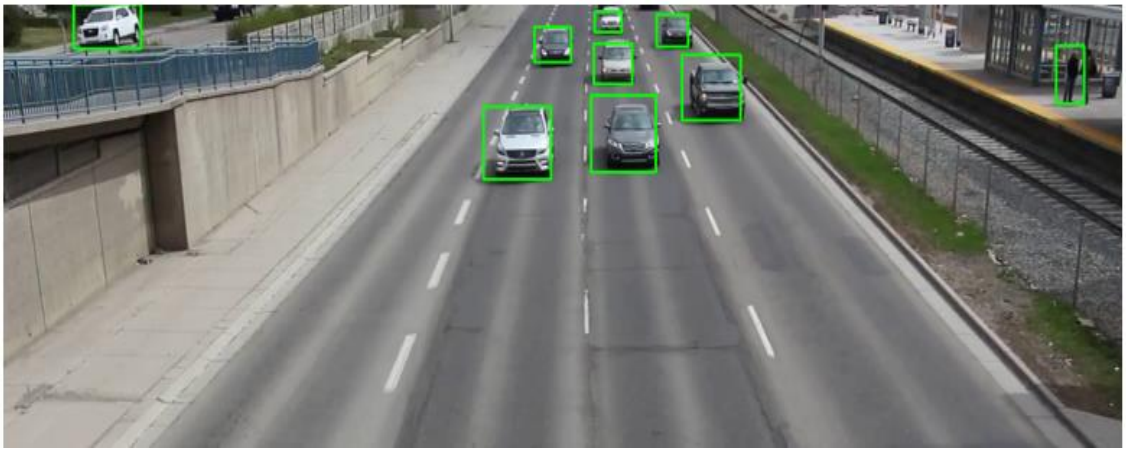
```
import cv2
import numpy as np
from IPython.display import display, clear_output
import matplotlib.pyplot as plt

net=cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes=[]
with open("coco.names", "r") as f:
    classes=[line.strip() for line in f.readlines()]
layer_names=net.getLayerNames()
output_layers=[layer_names[i-1] for i in net.getUnconnectedOutLayers()]

cap=cv2.VideoCapture('video2.mp4')

try:
    while cap.isOpened():
        ret,frame=cap.read()
        if not ret:
            break

        height, width, channels = frame.shape

        blob=cv2.dnn.blobFromImage(frame, 0.00392, (416,416), (0,0,0),
True, crop=False)
        net.setInput(blob)
        outs=net.forward(output_layers)

    class_ids=[]
    confidences=[]
    boxes=[]
```

```
    for out in outs:
        for detection in out:
            scores=detection[5:]
            class_id=np.argmax(scores)
            confidence=scores[class_id]
            if confidence > 0.5:
                center_x=int(detection[0]*width)
                center_y=int(detection[1]*height)
                w=int(detection[2]*width)
                h=int(detection[3]*height)

                x=int(center_x - w/2)
                y=int(center_y - h/2)

                boxes.append([x,y,w,h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

        indexes=cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
        for i in range(len(boxes)):
            if i in indexes:
                x,y,w,h=boxes[i]
                label=str(classes[class_ids[i]])
                color=(0,255,0) #GReen
                cv2.rectangle(frame, (x,y), (x+w, y+h), color, 2)
                cv2.putText(frame, label, (x,y+ 30),
cv2.FONT_HERSHEY_PLAIN,1,color,2)

        frame_rgb=cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        plt.figure(figsize=(10,10))
        plt.imshow(frame_rgb)
        plt.axis('off')
        display(plt.gcf())
        clear_output(wait=True)
        plt.close()
```

```
finally:
    cap.release()
    print("Stream ended")
```

Output:





Practical 7

Aim: Perform Feature extraction using RANSAC

Theory:

Random sample consensus, or RANSAC, is an iterative method for estimating a mathematical model from a data set that contains outliers. The RANSAC algorithm works by identifying the outliers in a data set and estimating the desired model using data that does not contain outliers.
RANSAC is accomplished with the following steps:

- Randomly selecting a subset of the data set
- Fitting a model to the selected subset
- Determining the number of outliers
- Repeating steps 1-3 for a prescribed number of iterations

In computer vision, RANSAC is used as a robust approach to estimate the fundamental matrix in stereo vision, for finding the commonality between two sets of points for feature-based object detection, and registering sequential video frames for video stabilization.

Advantages and disadvantages of RANSAC:
An advantage of RANSAC is its ability to do robust estimation of the model parameters, i.e., it can estimate the parameters with a high degree of accuracy even when a significant number of outliers are present in the data set. A disadvantage of RANSAC is that there is no upper bound on the time it takes to compute these parameters (except exhaustion). When the number of iterations computed is limited the solution obtained may not be optimal, and it may not even be one that fits the data in a good way. In this way RANSAC offers a trade-off; by computing a greater number of iterations the probability of a reasonable model being produced is increased.

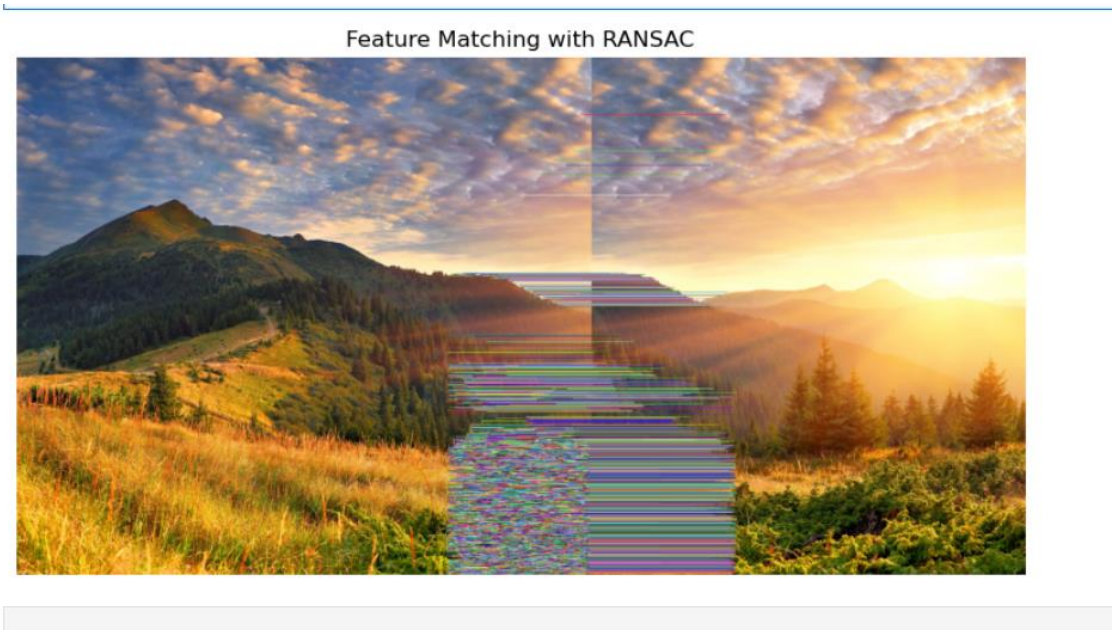
CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


# Load the color images
img1_color = cv2.imread('left1.jpg')  # Query image (color)
img2_color = cv2.imread('right1.jpg')  # Train image (color)


# Convert to grayscale for feature detection
img1 = cv2.cvtColor(img1_color, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2_color, cv2.COLOR_BGR2GRAY)


# Initialize SIFT detector
sift = cv2.SIFT_create()
```

```
# Detect keypoints and compute descriptors
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)

# FLANN parameters and matcher setup
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
search_params = dict(checks=50)

flann = cv2.FlannBasedMatcher(index_params, search_params)

# Match descriptors using KNN
matches = flann.knnMatch(des1, des2, k=2)
# Ratio test to keep good matches
good = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:
        good.append(m)

# Extract location of good matches
points1 = np.zeros((len(good), 2), dtype=np.float32)
points2 = np.zeros((len(good), 2), dtype=np.float32)

for i, match in enumerate(good):
    points1[i, :] = kp1[match.queryIdx].pt
    points2[i, :] = kp2[match.trainIdx].pt
# Find homography using RANSAC
    H, status = cv2.findHomography(points1, points2, cv2.RANSAC)

# Create a new image that puts the two images side by side
    height = max(img1_color.shape[0], img2_color.shape[0])
    width = img1_color.shape[1] + img2_color.shape[1]
    output = np.zeros((height, width, 3), dtype=np.uint8)

# Place both images within this new image
    output[0:img1_color.shape[0], 0:img1_color.shape[1]] = img1_color
    output[0:img2_color.shape[0],
img1_color.shape[1]:img1_color.shape[1]+img2_color.shape[1]] = img2_color
# Draw lines between the matching points
```

```
for i, (m, color) in enumerate(zip(good, np.random.randint(0, 255,
(len(good), 3)))):
    if status[i]:
        pt1 = tuple(map(int, kp1[m.queryIdx].pt))
        pt2 = tuple(map(int, kp2[m.trainIdx].pt))
        pt2 = (pt2[0] + img1_color.shape[1], pt2[1])  # Shift the point for
img2

        cv2.line(output, pt1, pt2, color.tolist(), 2)

# Convert the result to RGB for matplotlib display and show the final image
output_rgb = cv2.cvtColor(output, cv2.COLOR_BGR2RGB)
# Use matplotlib to display the image
plt.figure(figsize=(15, 5))
plt.imshow(output_rgb)
plt.axis('off')  # Turn off axis numbers and ticks
plt.title('Feature Matching with RANSAC')
plt.show()
```

OUTPUT



Practical 8

Aim: Perform Colorization

Theory: Image colorization is the process of taking an input grayscale (black and white) image and then producing an output colorized image that represents the semantic colors and tones of the input.

Challenges in Image Colorization:

- **Loss of Color Information**: A fundamental challenge in image colorization is the loss of color information during the conversion to grayscale. Color images capture the richness of the real world by encoding the full spectrum of light hitting each pixel. This spectrum is typically represented using three channels: red, green, and blue (RGB). Grayscale images, on the other hand, discard this detailed color data and only retain the luminance information, essentially representing how bright or dark each pixel is. This loss of information creates a significant obstacle for colorization algorithms.
- **Ambiguity in Grayscale Images:** Unlike color images that capture the full spectrum of light, grayscale images only represent luminance, or brightness. This ambiguity makes it difficult for colorization algorithms to definitively assign colors to grayscale pixels, requiring them to rely on additional information or make educated guesses to achieve a realistic outcome.
- **Lack of Semantic Understanding:** Grayscale images lack the rich information that allows us to distinguish between a brown bear and a polar bear, for instance. This semantic ambiguity presents difficulties. Colorization algorithms may assign colors based solely on local image features, potentially leading to unrealistic color choices.
- **Perceptual Color Constancy:** Colorization algorithms need to not only predict colors for grayscale pixels but also ensure those predicted colors appear consistent with the perceived lighting in the scene. Imagine a grayscale image of a landscape at dusk. The algorithm must not only assign colors to the sky and trees, but also account for the warm, yellowish light typical of dusk to achieve a realistic and believable colorization.
- **Color Inconsistency (Color Bleeding):** It occurs when the predicted colors for neighboring regions in the image fail to respect clear boundaries. During colorization, color bleeding might cause the red color of the rose to spill over and contaminate the edges of the leaf, resulting in a greenish tinge. This happens because colorization algorithms often rely on local image features and may struggle to differentiate between distinct objects with similar grayscale values.

In this practical, we will create a program to convert a black & white image i.e grayscale image to a colour image. We are going to use the Caffe colourization model for this program. And you should be familiar with basic OpenCV functions and uses like reading an image or how to load a pre-trained model using dnn module etc. Now let us discuss the procedure that we will follow to implement the program. Like RGB, lab colour has 3 channels L, a, and b. But here instead of pixel values, these have different significances i.e:

- **L-channel:** light intensity
- **a channel:** green-red encoding

- **b channel:** blue-red encoding

And in our program, we will use the L channel of our image as input to our model to predict ab channel values and then rejoin it with the L channel to generate our final image.

Steps:

Load the model and the convolution/kernel points

Read and preprocess the image

Generate model predictions using the L channel from our input image

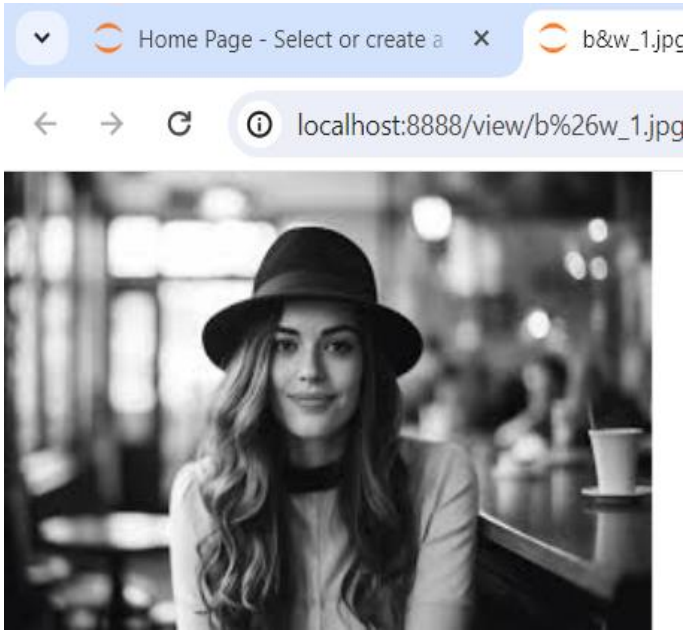
Use the output -> ab channel to create a resulting image

CODE

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

net=cv2.dnn.readNetFromCaffe('colorization_deploy_v2.prototxt',
'colorization_release_v2.caffemodel')

pts_in_hull= np.load('pts_in_hull.npy', allow_pickle=True)

class8=net.getLayerId("class8_ab")
conv8=net.getLayerId("conv8_313_rh")
pts_in_hull=pts_in_hull.transpose().reshape(2,313,1,1)
net.getLayer(class8).blobs= [pts_in_hull.astype(np.float32)]
net.getLayer(conv8).blobs=[np.full([1,313], 2.606, np.float32)]

image=cv2.imread('b&w_1.jpg')

gray_image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

gray_image=cv2.cvtColor(gray_image, cv2.COLOR_GRAY2RGB)

normalized_image=gray_image.astype('float32') / 255.0

lab_image=cv2.cvtColor(normalized_image, cv2.COLOR_RGB2Lab)
```

```
resized_l_channel=cv2.resize(lab_image[:,:,0], (224,224))
resized_l_channel-=50

net.setInput(cv2.dnn.blobFromImage(resized_l_channel))
pred=net.forward()[0,:,:,:].transpose((1,2,0))

pred_resized=cv2.resize(pred,(image.shape[1], image.shape[0]))

colorized_image=np.concatenate((lab_image[:,:,0][:,:, np.newaxis],
pred_resized), axis=2)

colorized_image=cv2.cvtColor(colorized_image, cv2.COLOR_Lab2BGR)

colorized_image-np.clip(colorized_image, 0 ,1)

colorized_image=(255*colorized_image).astype('uint8')
cv2.imwrite('path_to_output/colorized_image.jpg', colorized_image)

#cv2.imshow('Colorized Image', colorized_image)
#cv2.waitKey(0)
#cv2.destroyAllWindows()

colorized_image_rgb= cv2.cvtColor(colorized_image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(8,8))
plt.imshow(colorized_image_rgb)
plt.axis('off')
plt.show()
```

OUTPUT


Home Page - Select or create a
b&w_1.jpg
localhost:8888/view/b%26w_1.jpg

Practical 9

Aim: Perform Text Detection and Recognition

Theory: Computer vision involves extracting information from visual data and allows us to perform complex tasks such as classification, prediction, recognition, and much more. In this practical, we will look at how to detect text using Tesseract in media, a classic optical character recognition application.

Optical character recognition:
Optical character recognition (OCR), is a revolutionary technology that enables machines to interpret and convert images of text into machine-readable formats. It allows us to utilize the potential of printed or handwritten text.

Simply put, the goal of OCR is to convert the human perception of characters and convert them into machine-encoded text.

The concept of optical character recognition is used in text detection, where we aim to identify and recognize the text found within an image or a video.

The Vision API can detect and extract text from images. There are two annotation features that support optical character recognition (OCR):

- TEXT_DETECTION detects and extracts text from any image. For example, a photograph might contain a street sign or traffic sign. The JSON includes the entire extracted string, as well as individual words, and their bounding boxes. In text detection, our goal is to automatically compute the bounding boxes for every region of text in an image.
- DOCUMENT_TEXT_DETECTION also extracts text from an image, but the response is optimized for dense text and documents. The JSON includes page, block, paragraph, word, and break information.

CODE

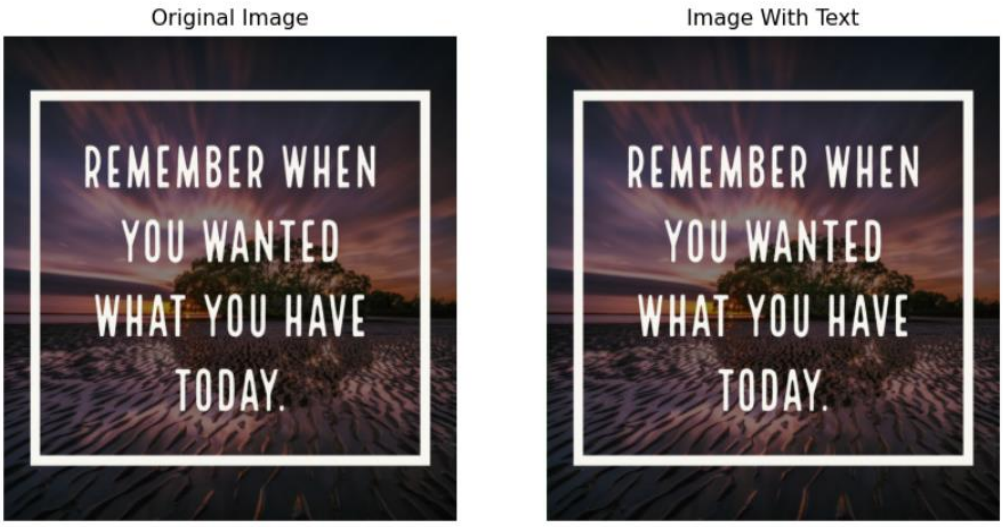
```
import pytesseract
import cv2
from pytesseract import Output
# Specify the tesseract executable path
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'
# Load the image
image = cv2.imread('qoute.jpg')
original_image = image.copy() # Make a copy for displaying the original
later
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

```
# Use Opencv to find text blocks (simple thresholding)
_, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY_INV)
# Dilate to connect text characters
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))
dilate = cv2.dilate(thresh, kernel, iterations = 3)
# Find Contours
contours, _ = cv2.findContours(dilate, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
# Sort Contours by their y-coordinate first, then x-coordinate
lines = sorted(contours, key=lambda ctr:
(cv2.boundingRect(ctr)[1],cv2.boundingRect(ctr)[0]))
# Go through each contour, crop and read the text
for contour in lines:
    x,y,w,h = cv2.boundingRect(contour)
    # Make sure the contour area is a reasonable size
    if w*h > 50:
        roi = image[y:y+h, x:x+w]
        text = pytesseract.image_to_string(roi, lang='eng', config= '--psm
6')
        cv2.putText(image, text, (x,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.3,
(0,255,0), 1)
        cv2.rectangle(image, (x,y), (x+w,y+h), (0,255,0), 1)
        print(text)
import matplotlib.pyplot as plt
original_image_rgb = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
# Display the original image
plt.figure(figsize=(10,10))
plt.subplot(1,2,1)
plt.imshow(original_image_rgb)
plt.title('Original Image')
plt.axis('off')
# Display the image with detected text
plt.subplot(1,2,2)
plt.imshow(image_rgb)
plt.title('Image With Text')
```

```
plt.axis('off')
plt.show()
```

OUTPUT

```
REMEMBER WHEN
YOU WANTED
WHAT YOU HAVE
TODAY.
```



Practical 10

Aim: Perform Image matting and compositing

Theory: Image Matting is the process of accurately estimating the foreground object in images and videos. It is a very important technique in image and video editing applications, particularly in film production for creating visual effects.

In case of image segmentation, we segment the image into foreground and background by labeling the pixels. Image segmentation generates a binary image, in which a pixel either belongs to foreground or background. However, Image Matting is different from the image segmentation, wherein some pixels may belong to foreground as well as background, such pixels are called partial or mixed pixels.

In order to fully separate the foreground from the background in an image, accurate estimation of the alpha values for partial or mixed pixels is necessary.

Compositing is the process or technique of combining visual elements from separate sources into single images, often to create the illusion that all those elements are parts of the same scene.

In the matting process, a foreground element of arbitrary shape is extracted from a background image. The matte extracted by this process describes the opacity of the foreground element at every point. In the compositing process, the foreground element is placed over a new background image, using the matte to hold out those parts of the new background that the foreground element obscures.

CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt


def create_mask_for_foreground(fg_image):

    fg_hsv=cv2.cvtColor(fg_image, cv2.COLOR_BGR2HSV)

    lower_green=np.array([36,25,25])
    upper_green=np.array([86, 255, 255])

    mask=cv2.inRange(fg_hsv, lower_green, upper_green)

    foreground_mask=cv2.bitwise_not(mask)
```

```
    kernel=np.ones((3,3), np.uint8)
    foreground_mask=cv2.morphologyEx(foreground_mask, cv2.MORPH_OPEN,
kernel, iterations=2)
    foreground_mask=cv2.dilate(foreground_mask, kernel, iterations=4)
    return foreground_mask

def composite_fg_with_new_bg(fg_image, bg_image, fg_mask):
    bg_resized=cv2.resize(bg_image,(fg_image.shape[1], fg_image.shape[0]))
    fg_mask_normalized=fg_mask/255.0

    fg_prepared=cv2.bitwise_and(fg_image, fg_image, mask=fg_mask)
    bg_prepared=cv2.bitwise_and(bg_resized, bg_resized,
mask=cv2.bitwise_not(fg_mask))

    composite_image=cv2.add(fg_prepared, bg_prepared)
    return composite_image

foreground_image_path='greenscreen.jpg'
foreground=cv2.imread(foreground_image_path)
if foreground is None:
    raise ValueError("Error loading foreground image")

background_image_path='bg.jpg'
background=cv2.imread(background_image_path)
if background is None:

    background=np.full(foreground.shape, 255, dtype=foreground.dtype)

foreground_mask=create_mask_for_foreground(foreground)

composite_image=composite_fg_with_new_bg(foreground, background,
foreground_mask)

foreground_rgb=cv2.cvtColor(foreground, cv2.COLOR_BGR2RGB)
foreground_mask_rgb=cv2.cvtColor(foreground_mask, cv2.COLOR_GRAY2RGB)
composite_image_rgb=cv2.cvtColor(composite_image, cv2.COLOR_BGR2RGB)
```

```
plt.figure(figsize=(18,6))

plt.subplot(1,3,1)
plt.imshow(foreground_rgb)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1,3,2)
plt.imshow(foreground_mask_rgb, cmap='gray')
plt.title('Foreground Mask')
plt.axis('off')

plt.subplot(1,3,3)
plt.imshow(composite_image_rgb)
plt.title('Composite Image')
plt.axis('off')

plt.show()
```

OUTPUT