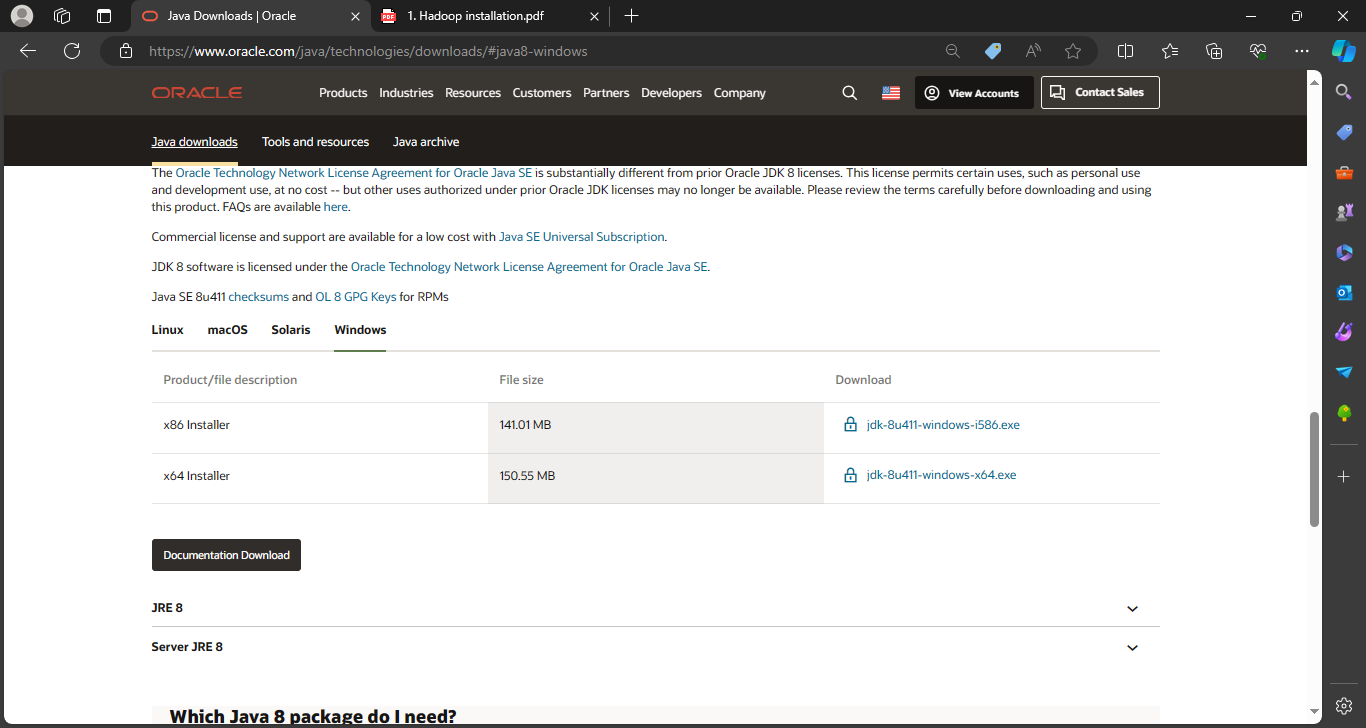
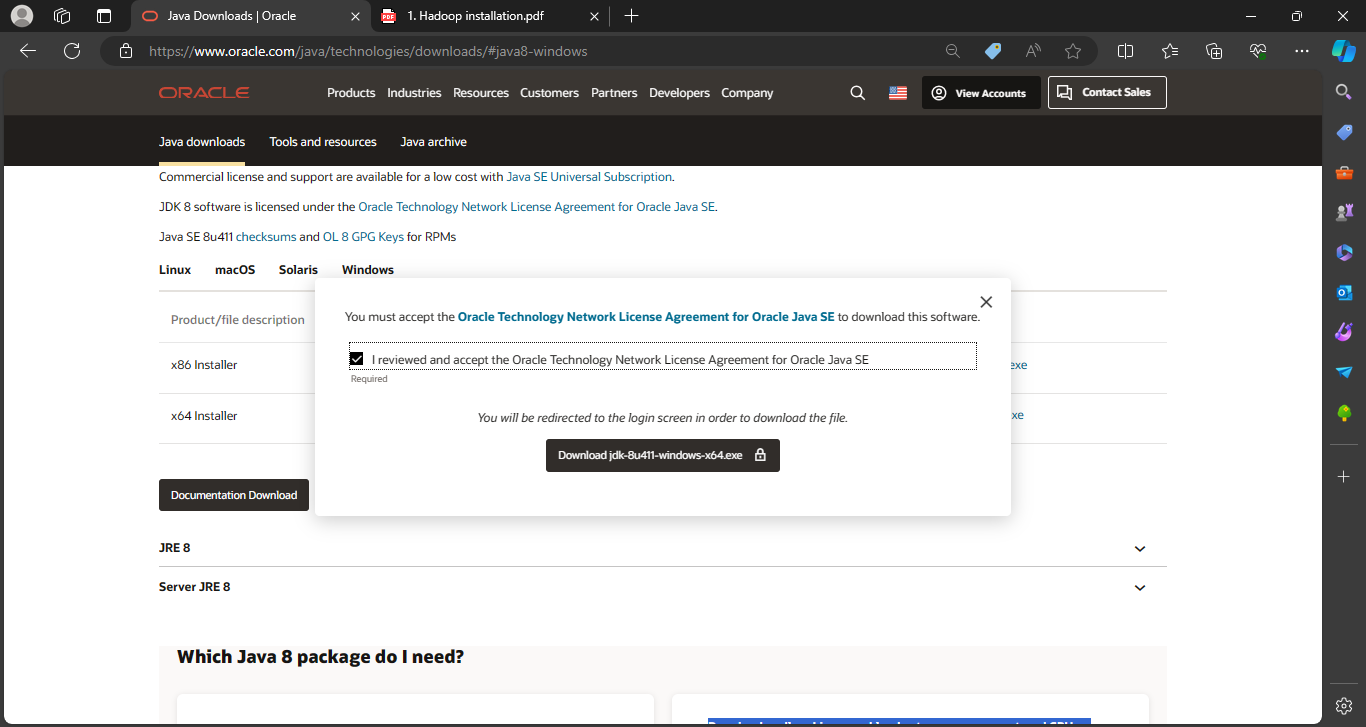
**Practical 1**

**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](https://www.oracle.com/java/technologies/downloads/" \l "java8-windows)





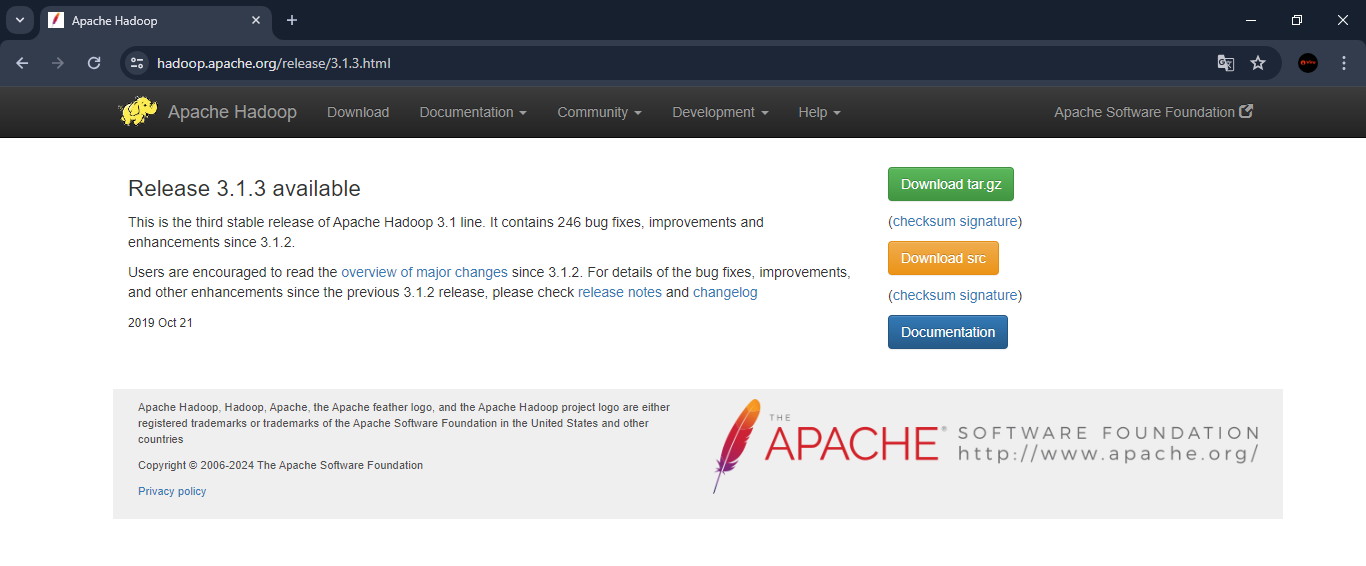
Oracle registration

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

Jdk is downloaded

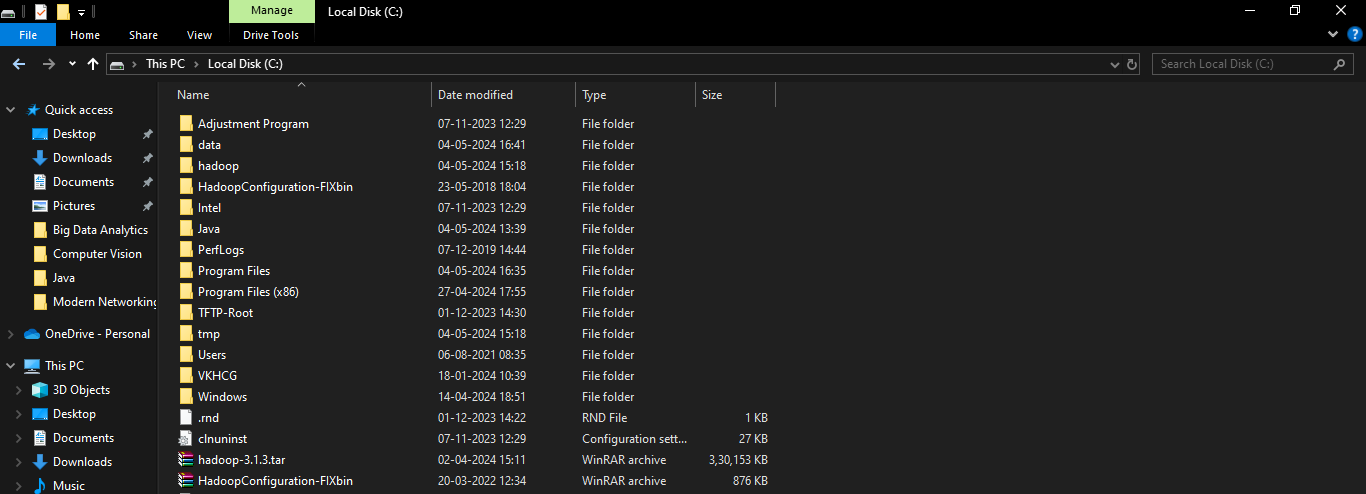
**Step 2: Download Hadoop for the local system**

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

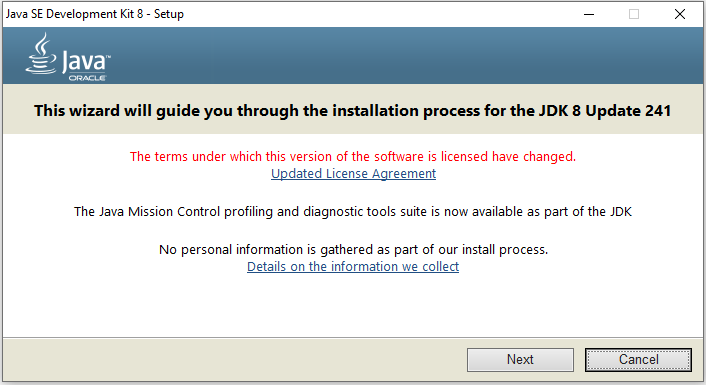


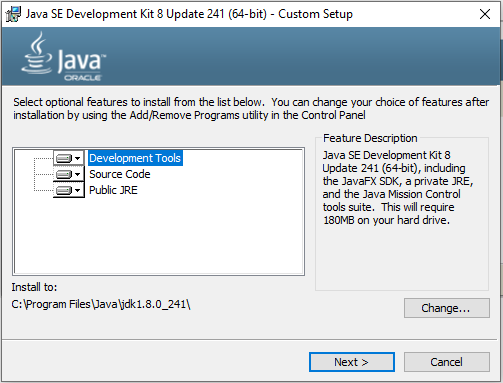
Hadoop downloaded

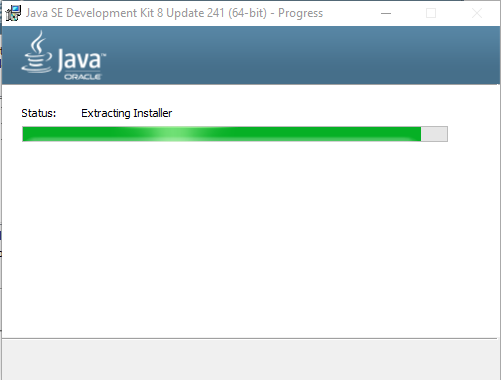
Move files to c drive

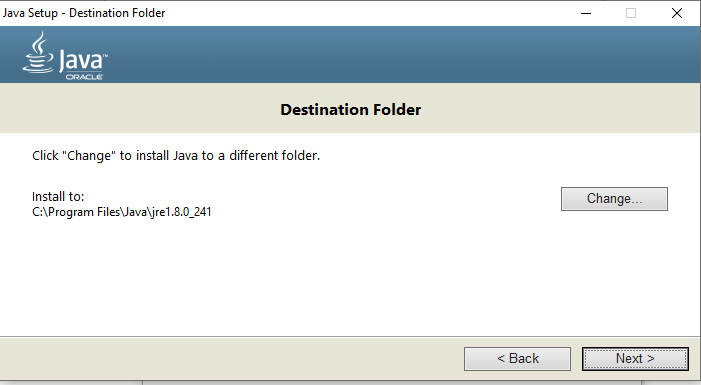


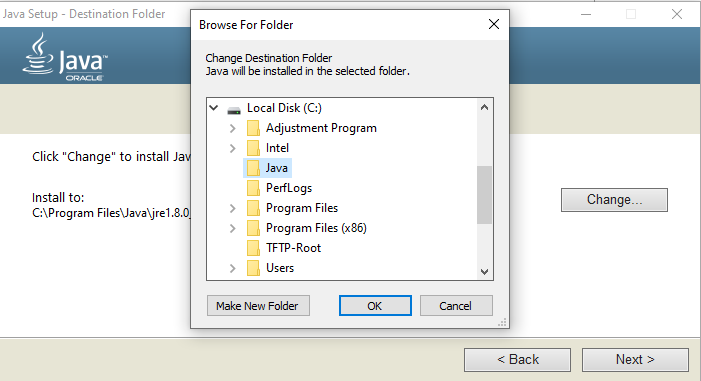
**Step 3: Install Java**

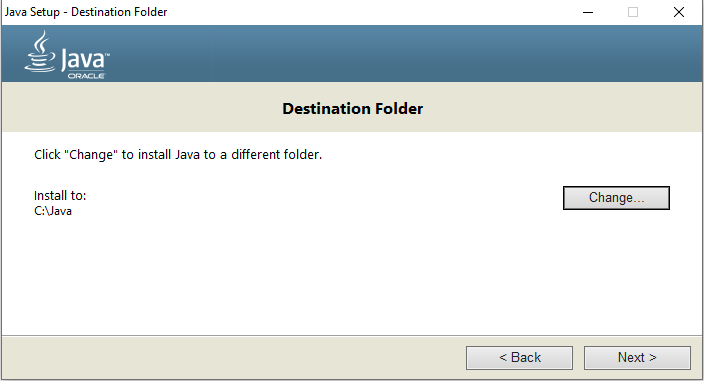


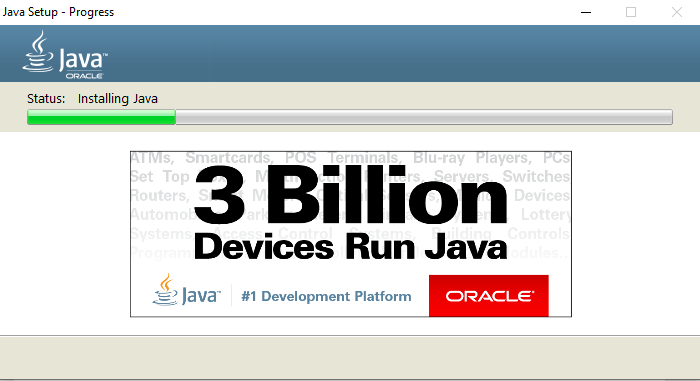


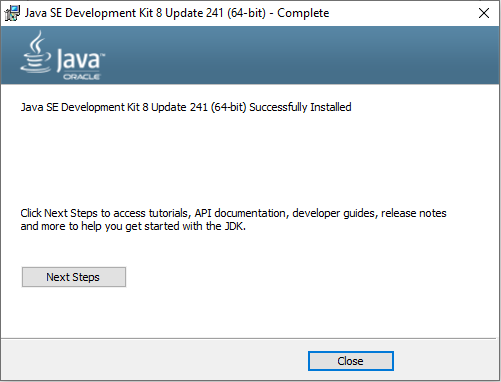


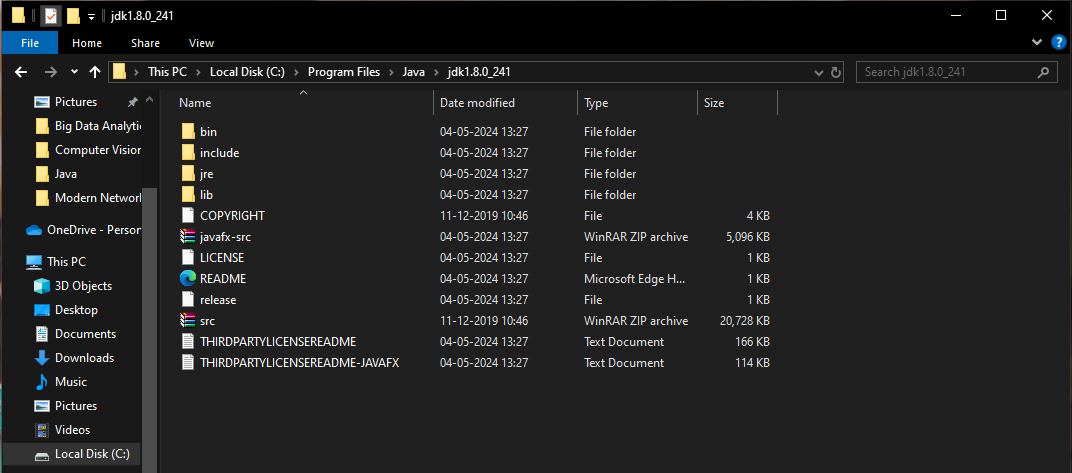


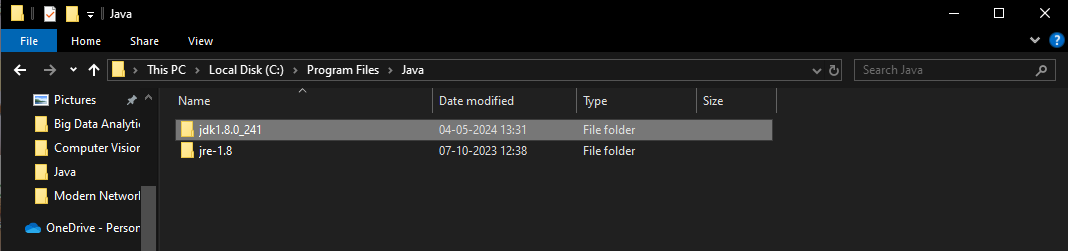




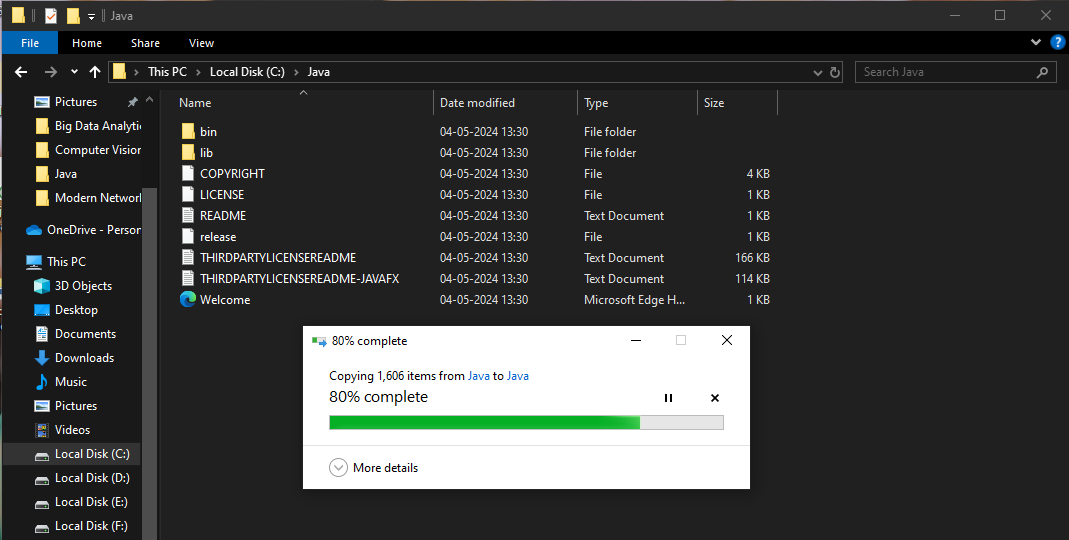


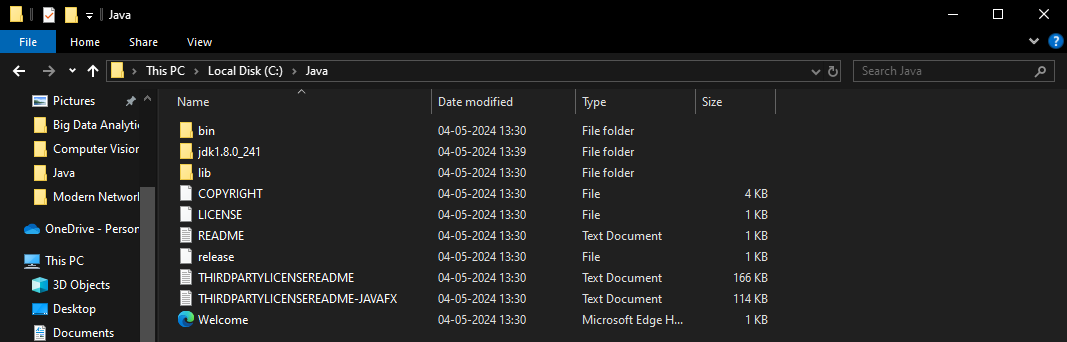






Paste it here.



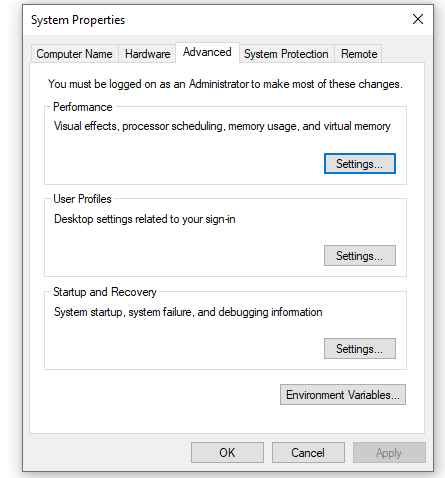


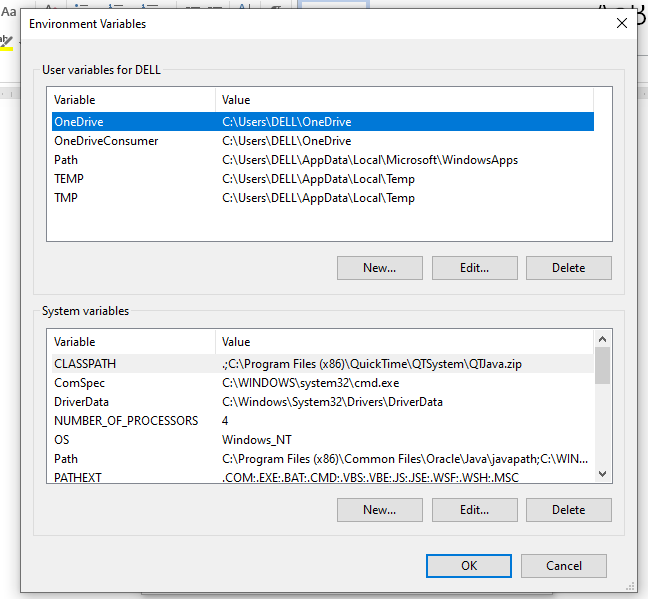
Now java will be available at:

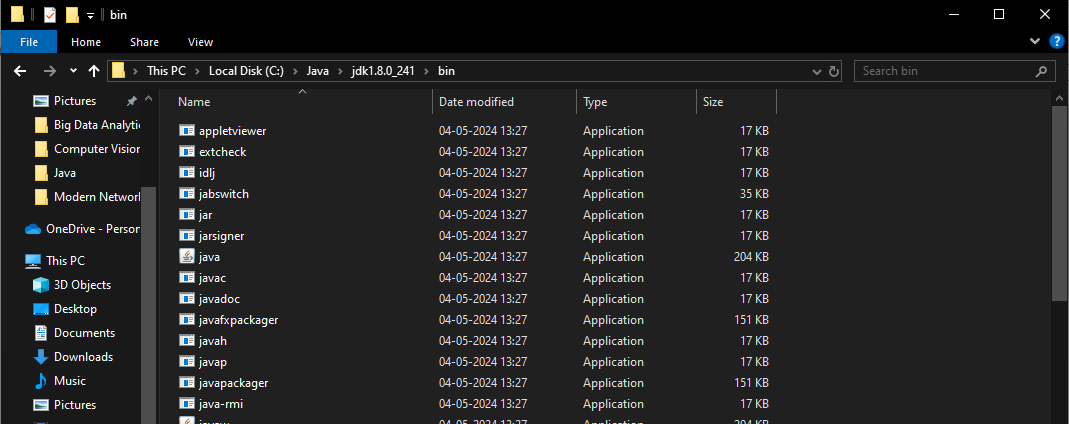
C:\Java

**Step 4: Setting environment variables for java**

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

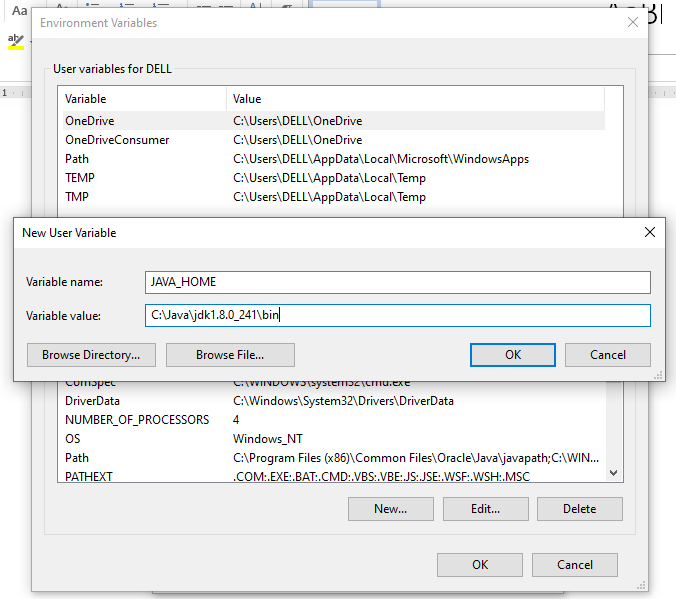






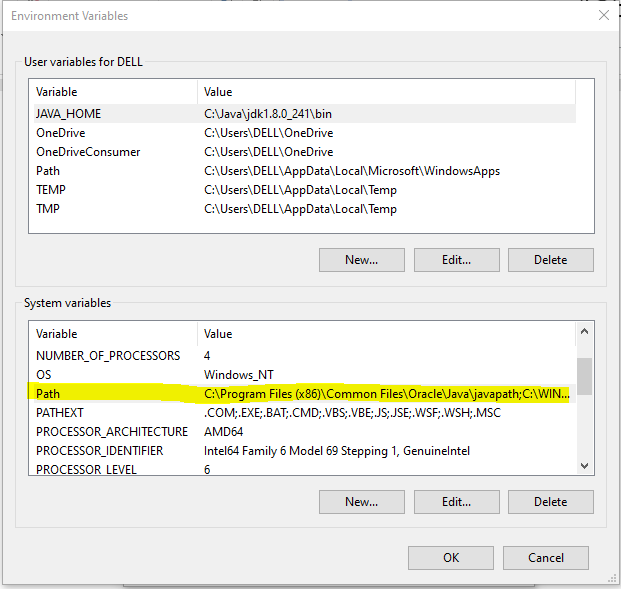
**Set java home and path for java**

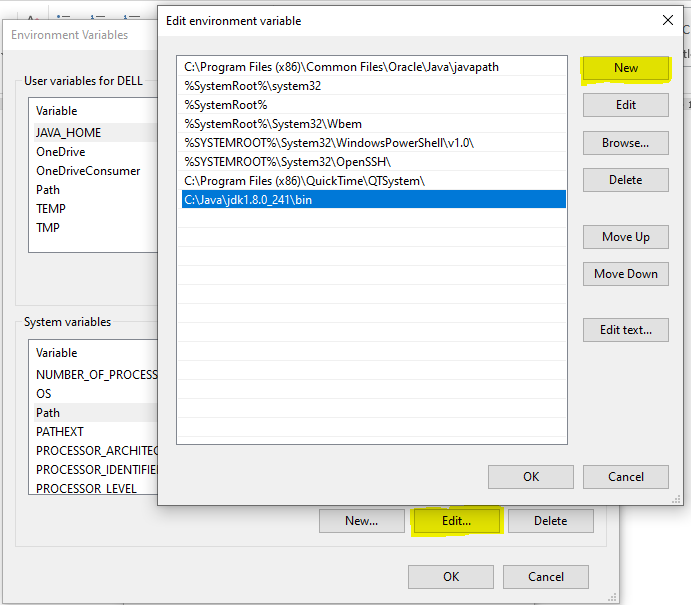
Set user variables



Set system variables

System variable->path->edit->new



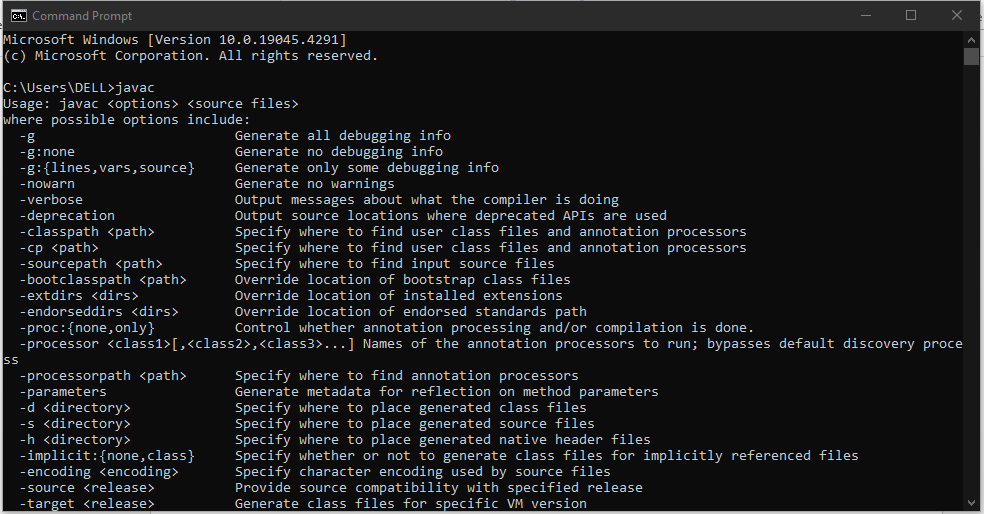


Click Ok-> OK->close

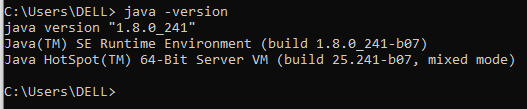
Java successfully installed

**Step 5: Java version checking**

Go to command prompt

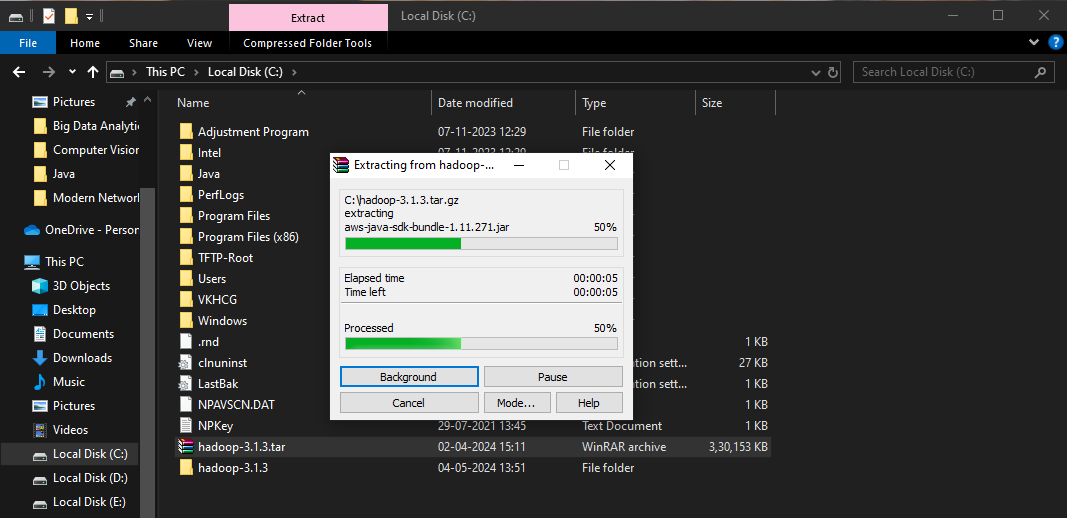


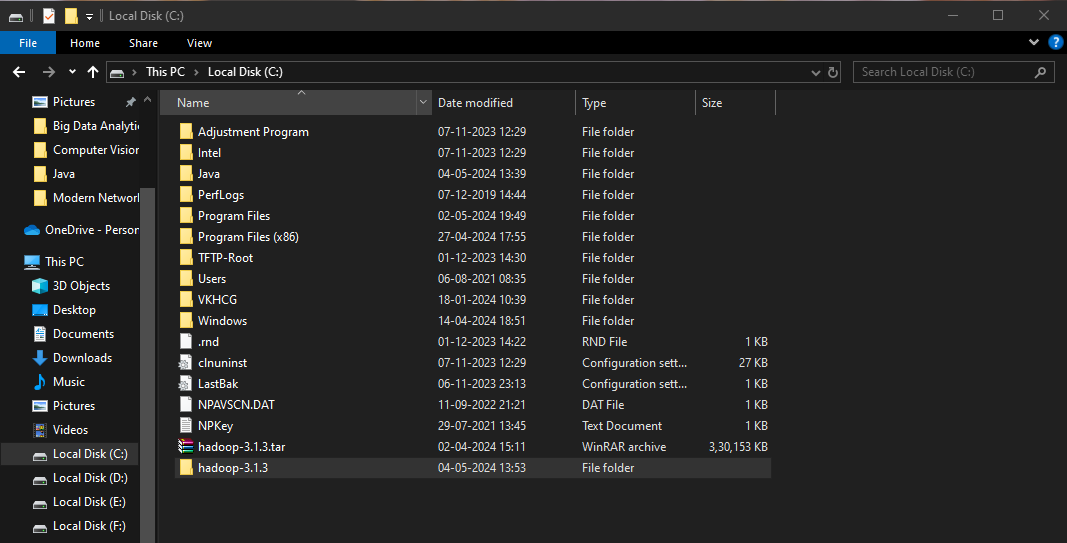
Check java version



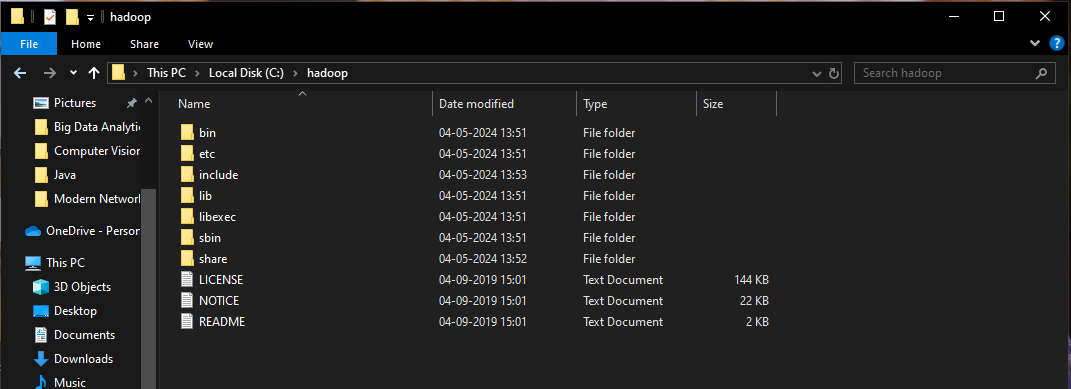
**Step 6: Install Hadoop now to our local system**

Unzip Hadoop setup file

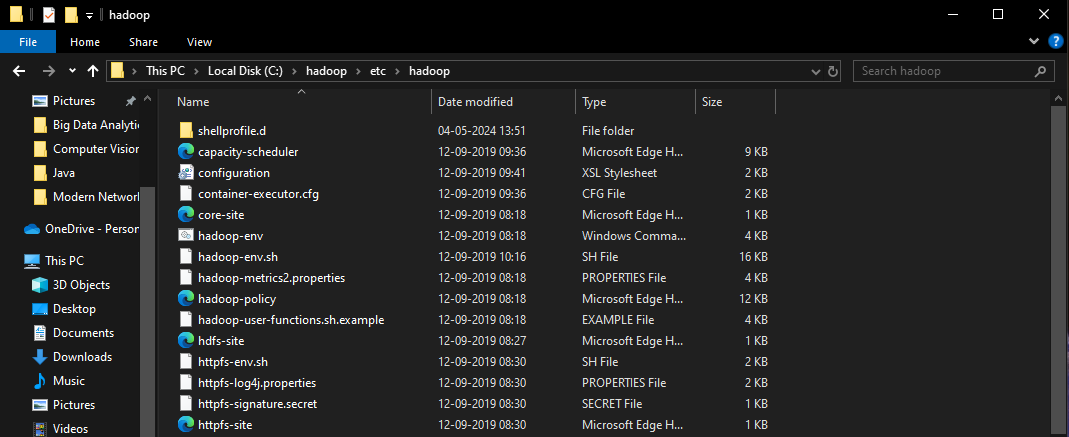




Rename Hadoop-3.1.3 folder as hadoop



**Step 7: Configuration of Hadoop** hadoop->etc-> hadoop



**Important files:**

Core-site.xml

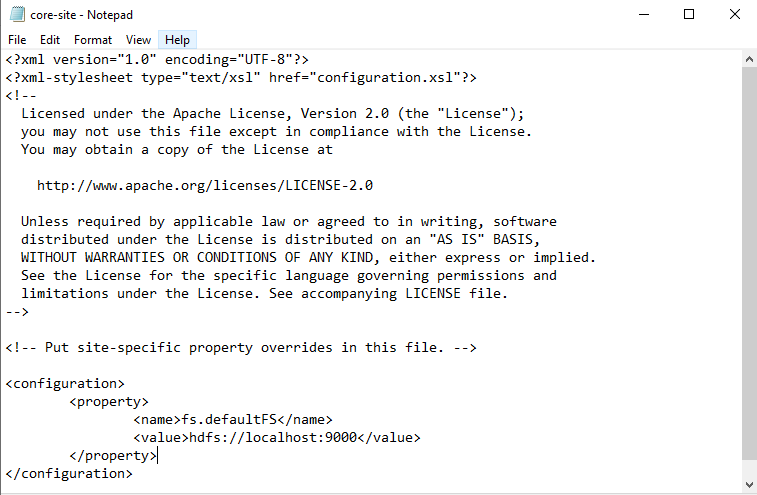
Hdfs-site.xml

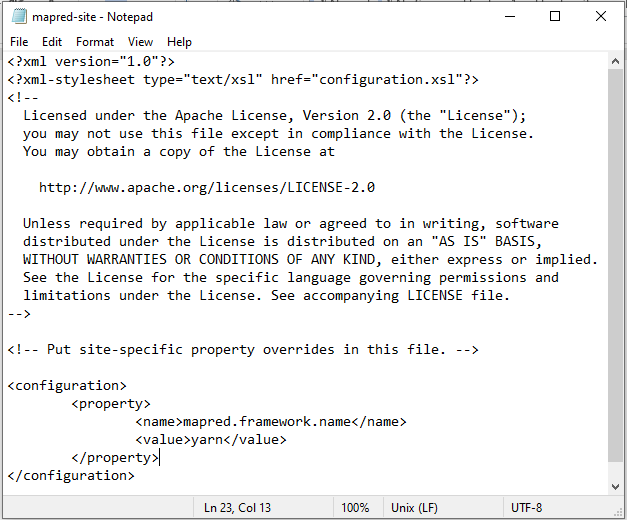
Mapred-site.xml

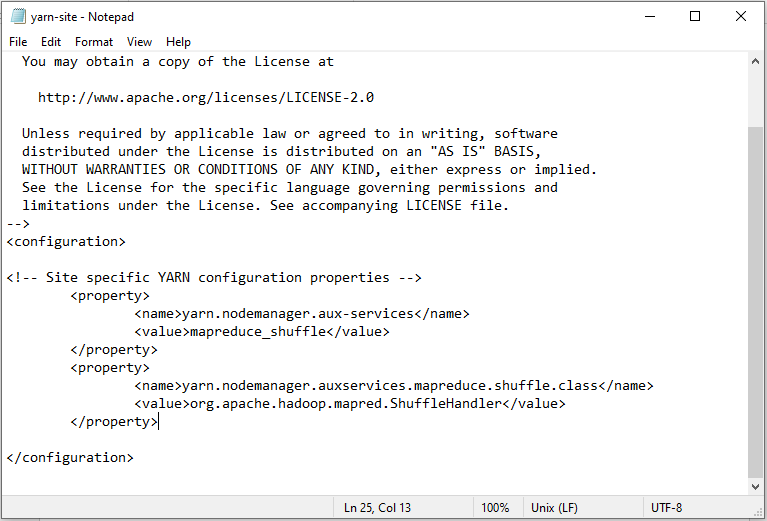
Yarn-site.xml

Hadoop-env windows command prompt file

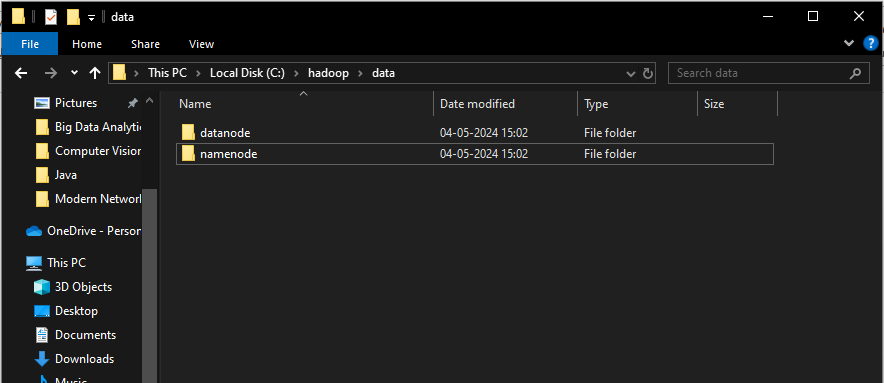
Open all these files in notepad

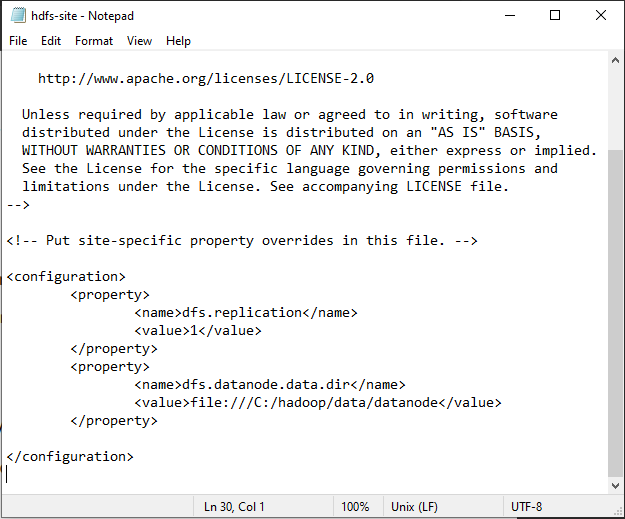


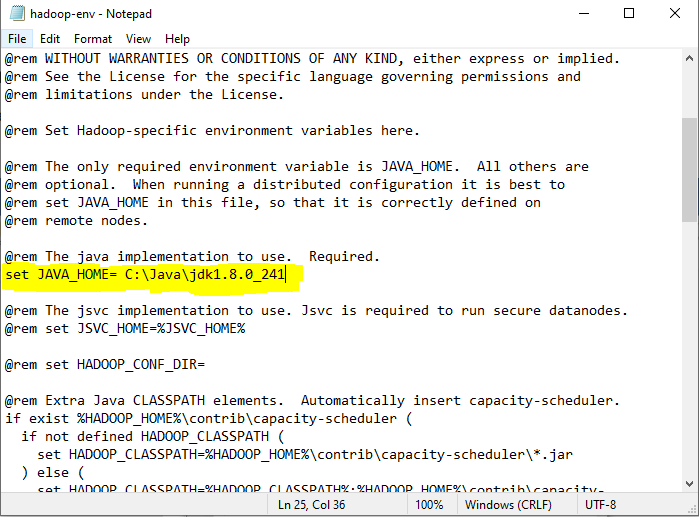




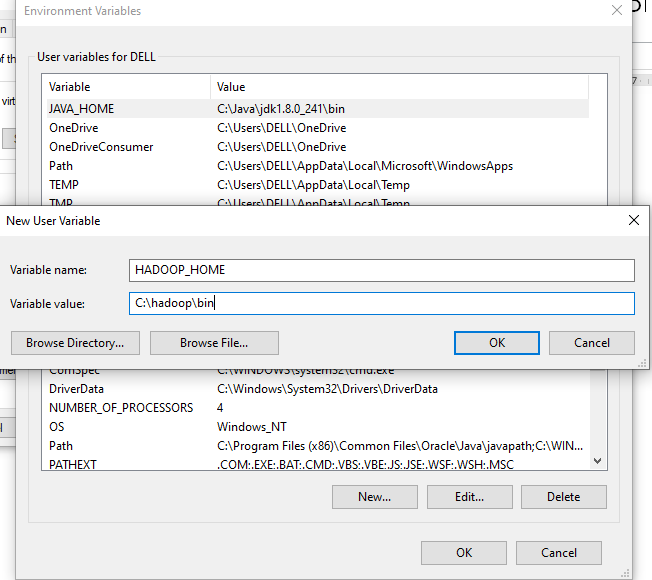
**Create data directory and subdirectories in hadoop**

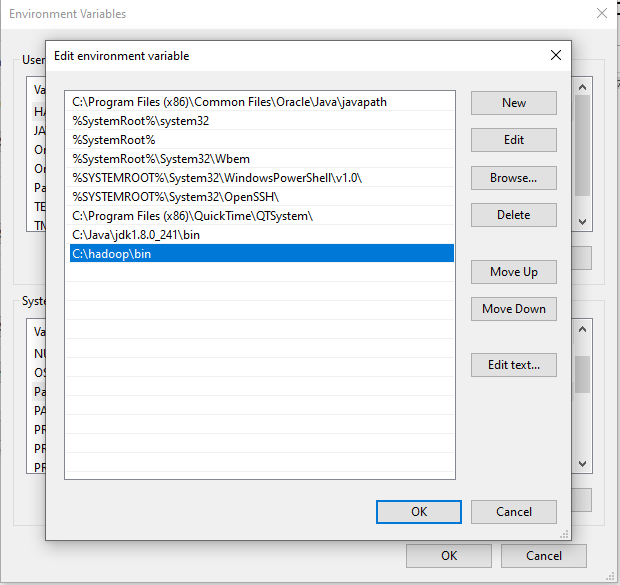


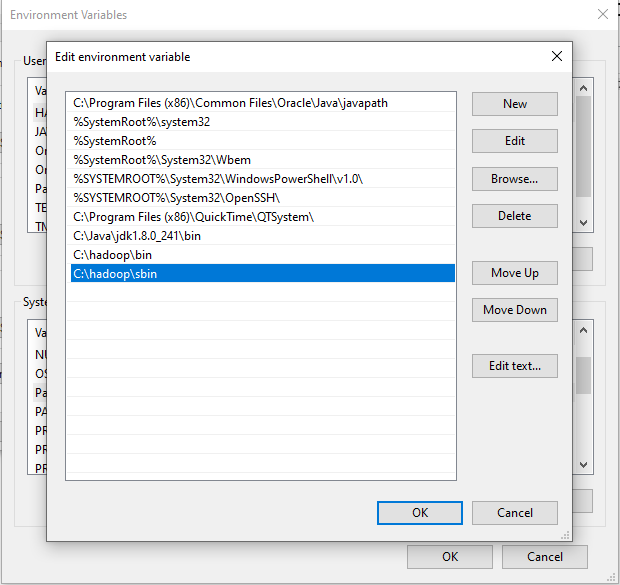




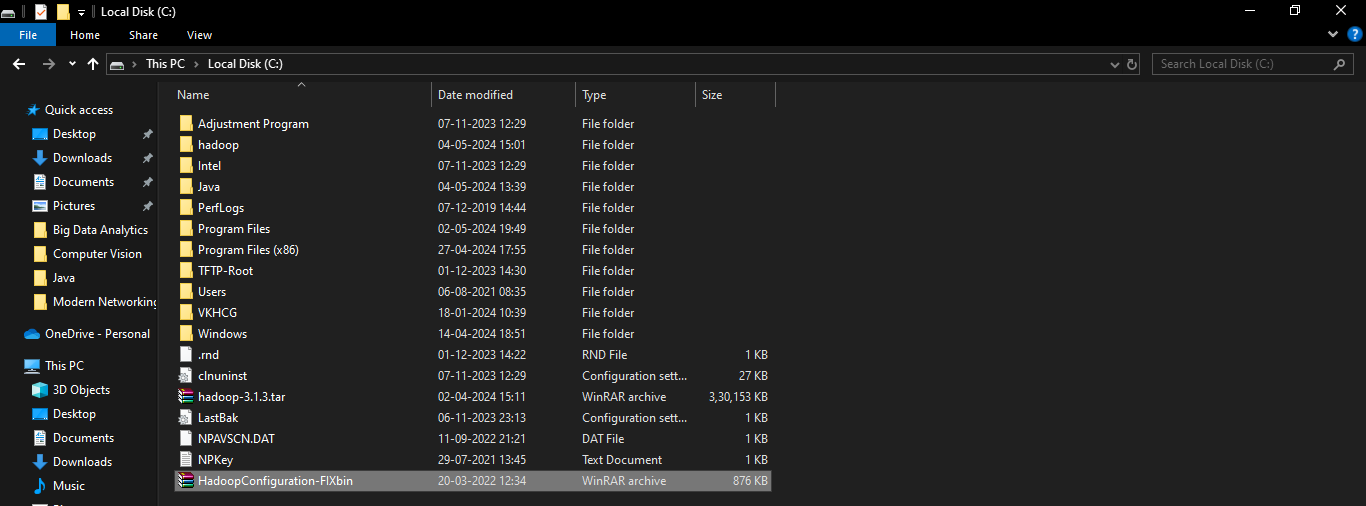
Set home and path for Hadoop

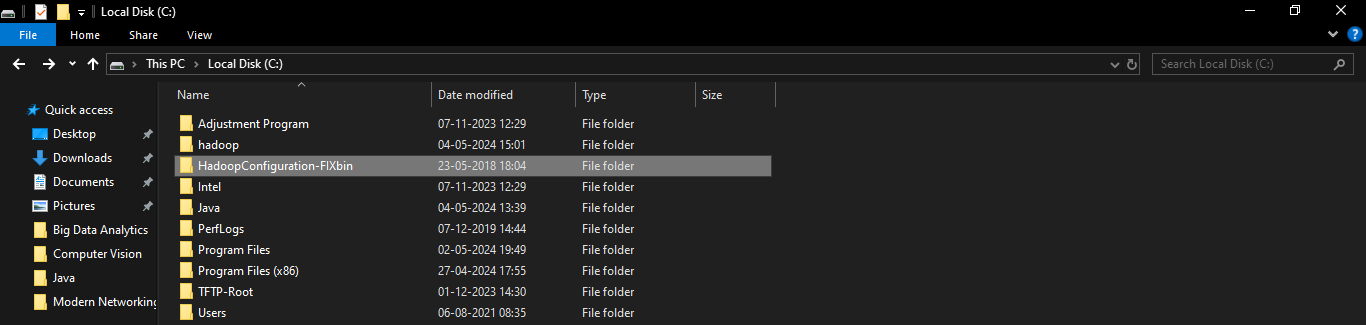




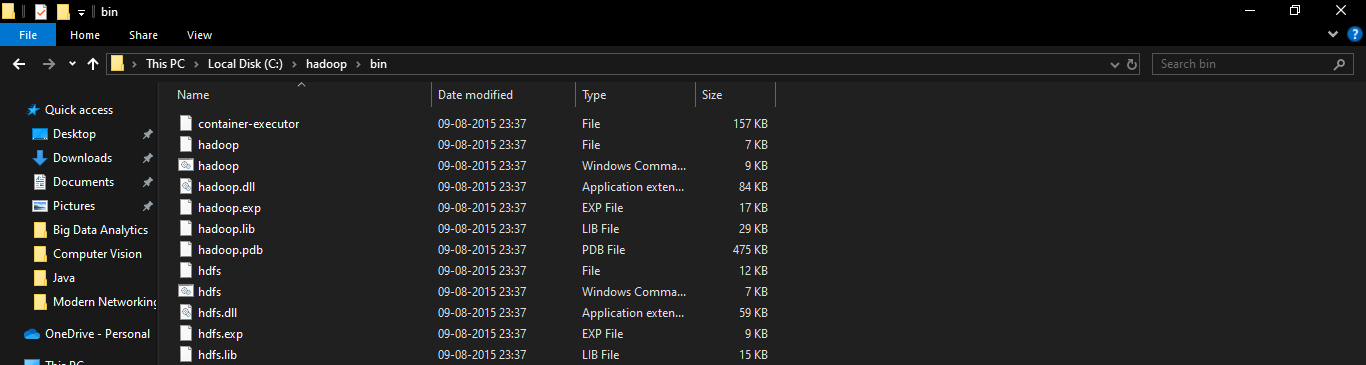


**Other configuration files**





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



**Step 8: Verification of Hadoop installation**

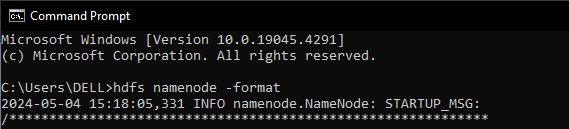
Go to command prompt.

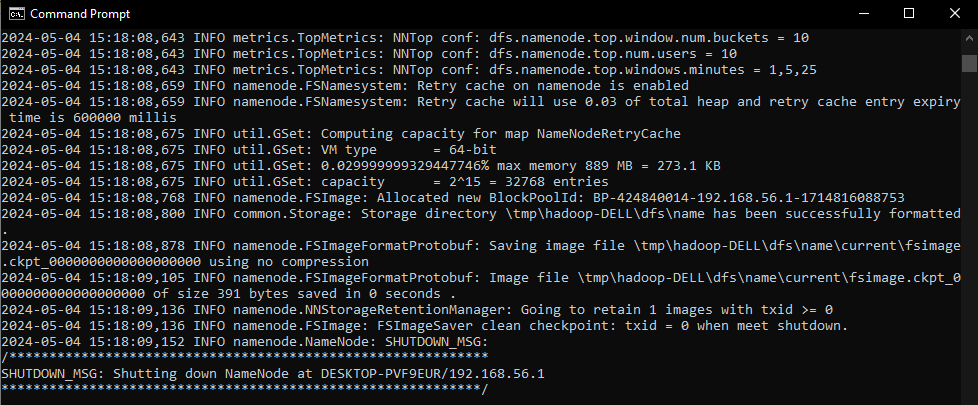
Type:

hdfs namenode -format

set of files pop up on the terminal.

That means successful installation of Hadoop.



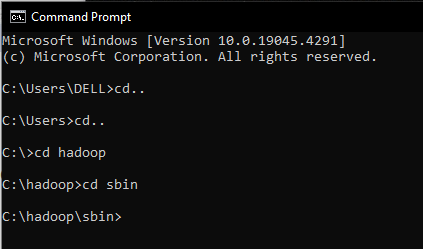


Namenode is successfully started.

Open new terminal and start all Hadoop daemons

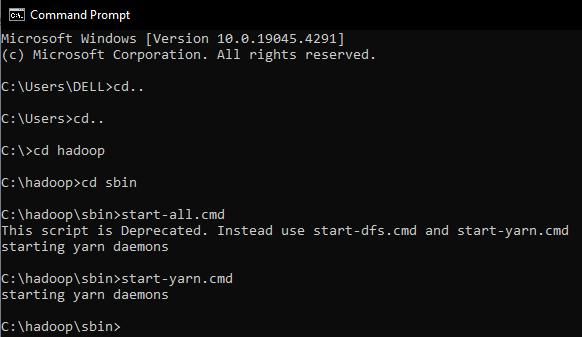
Go to Hadoop location.

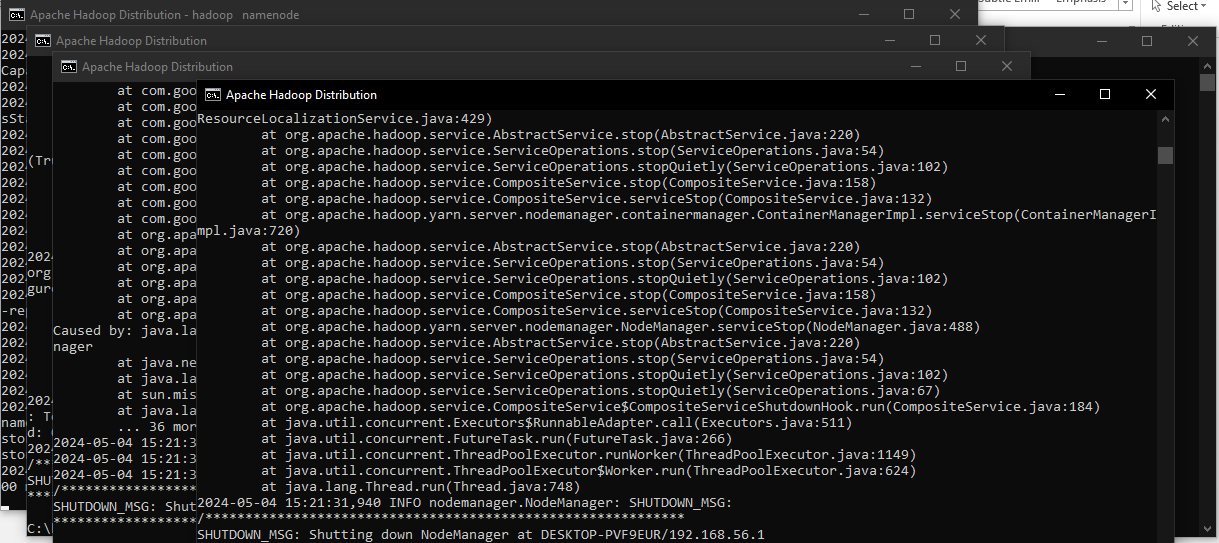
C:\hadoop\sbin



Type the command

start-all.cmd

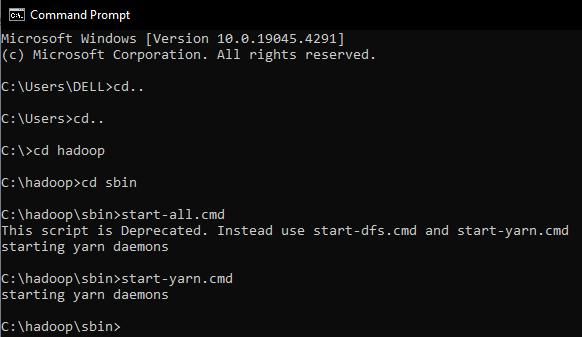


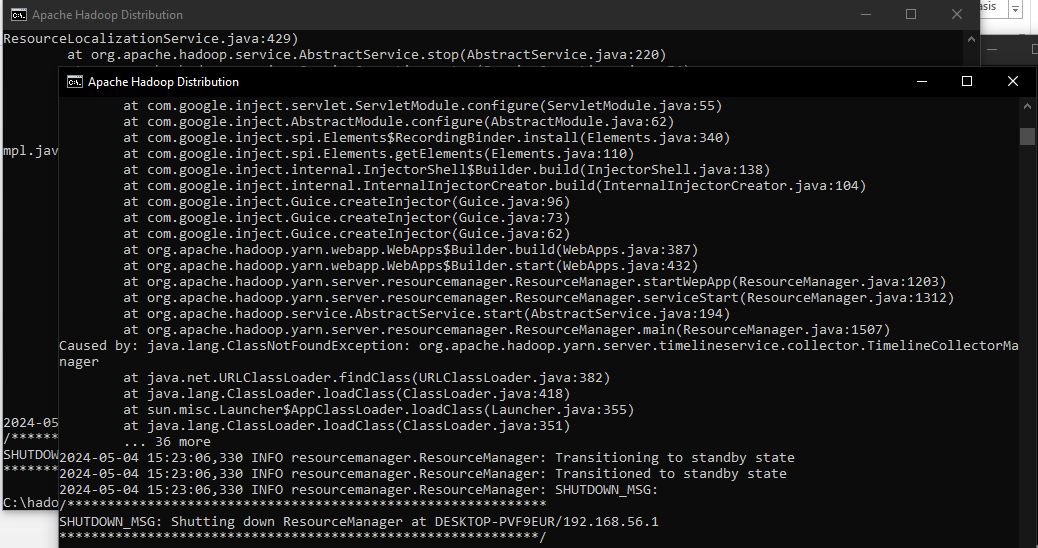


All the nodes will start successfully.

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.

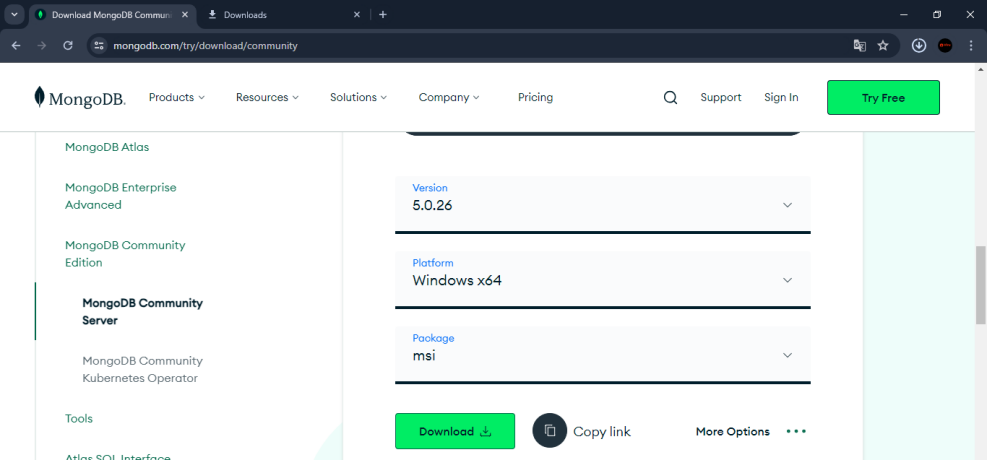
We can also use the dot notation to access MongoDB data. Its easy syntax makes our job a lot easier. Additionally, PyMongo’s rich documentation is always standing there with a helping hand. We will use this library for accessing MongoDB.

**Steps of the installation:**

**Step 1: Download MongoDB**

Go to official website: MongoDB Community server

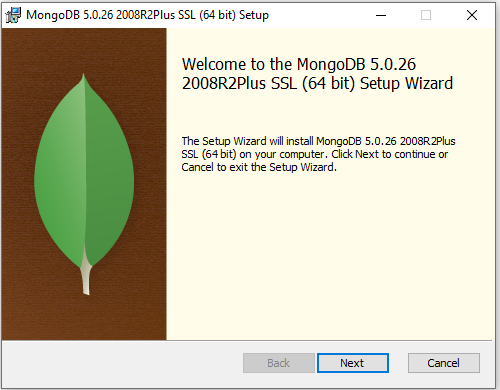
<https://www.mongodb.com/try/download/community>

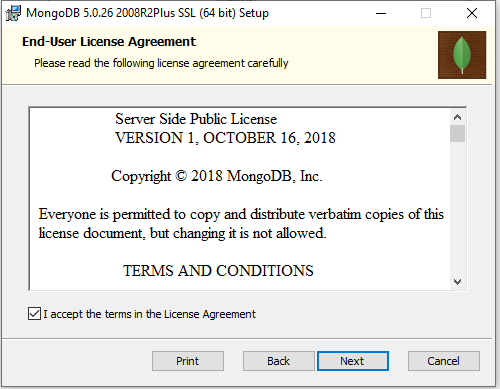


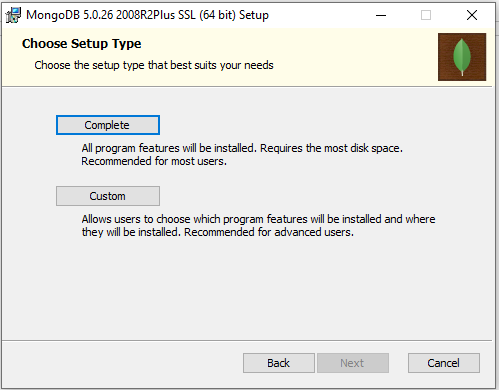
Click on download

**Step 2: Install MongoDB**

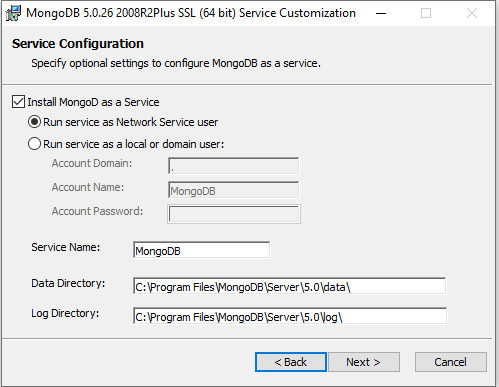
It will download msi file. Click on it and Start the installation.



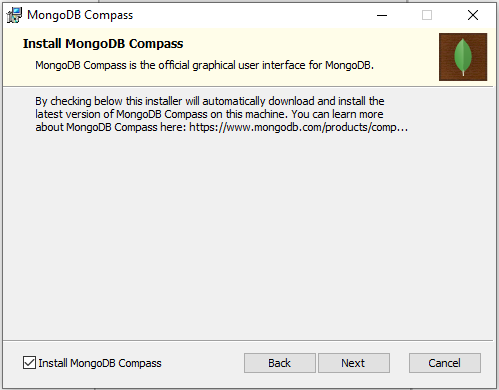




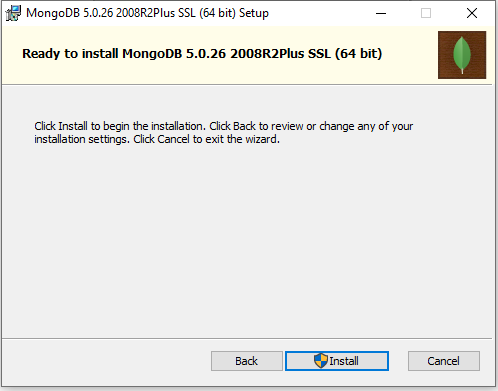
Click on complete option



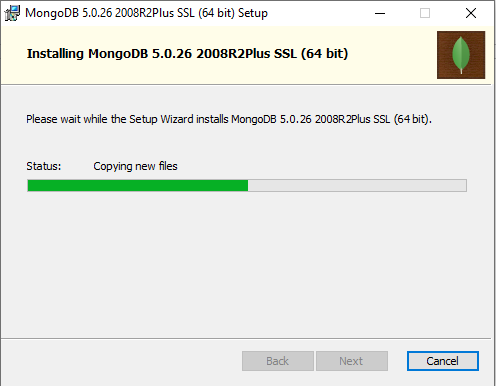
Keep all these setting as it is. Click on next.

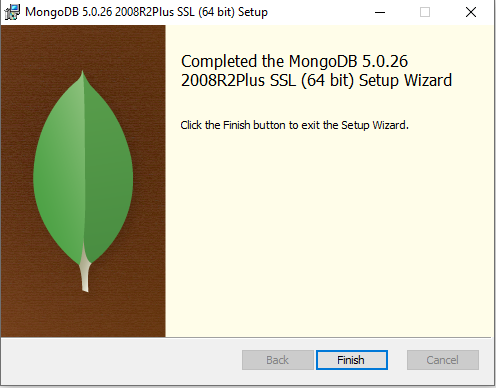


Click on next

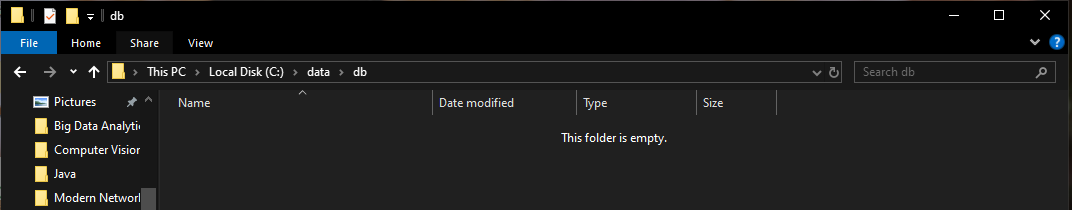


Click on install





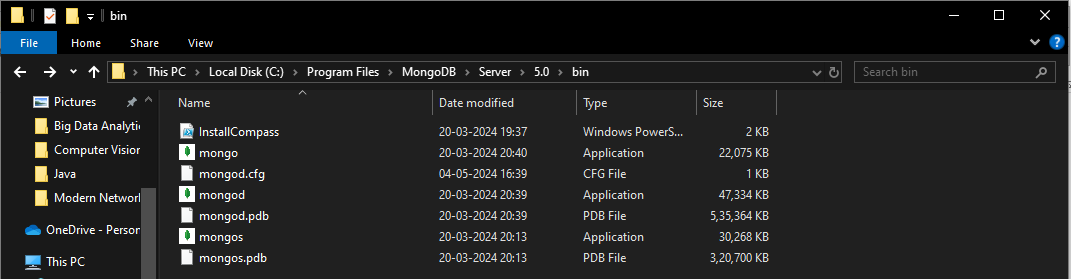
**Step 3: Verify MongoDB Installation**First we will create C:\data\db directory



Now go to

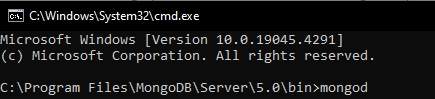
C:\Program Files\MongoDB\Server\5.0\bin

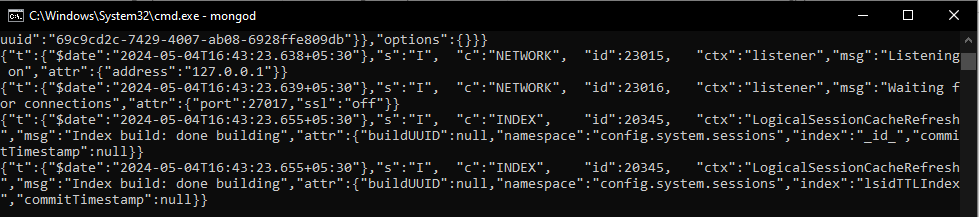
Start command prompt from this location



To start mongo db server

Enter mongod command





Mongo daemon is started now

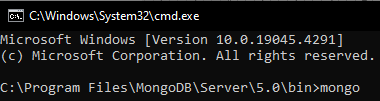
To open the mongo shell

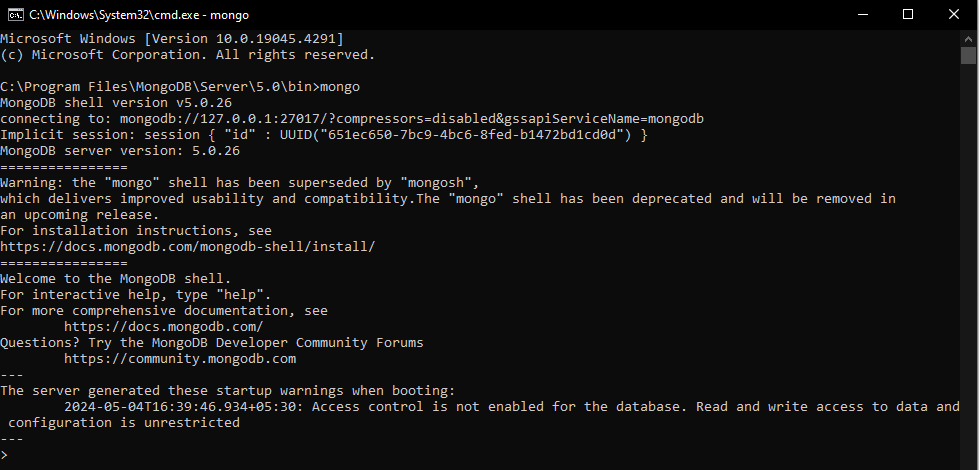
Go to

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

Start command prompt from this location

Fire the command: mongo

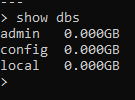




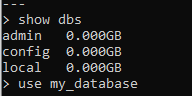
Mongo shell is started

To see all the default databases:

>show dbs

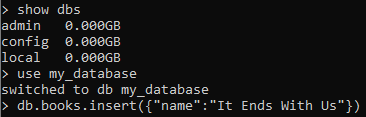


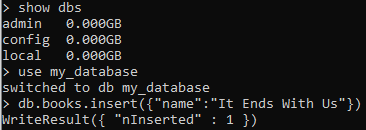
To create new database named my\_database:



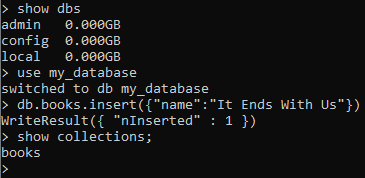
To create collection in the database:

And insert json values into it:

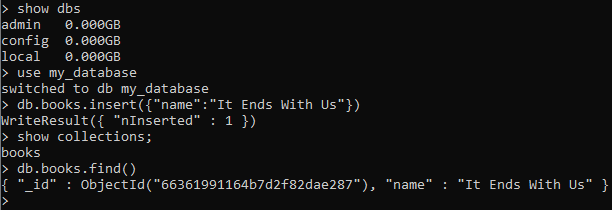




To see entered collection:



To see all the documents in the collection:

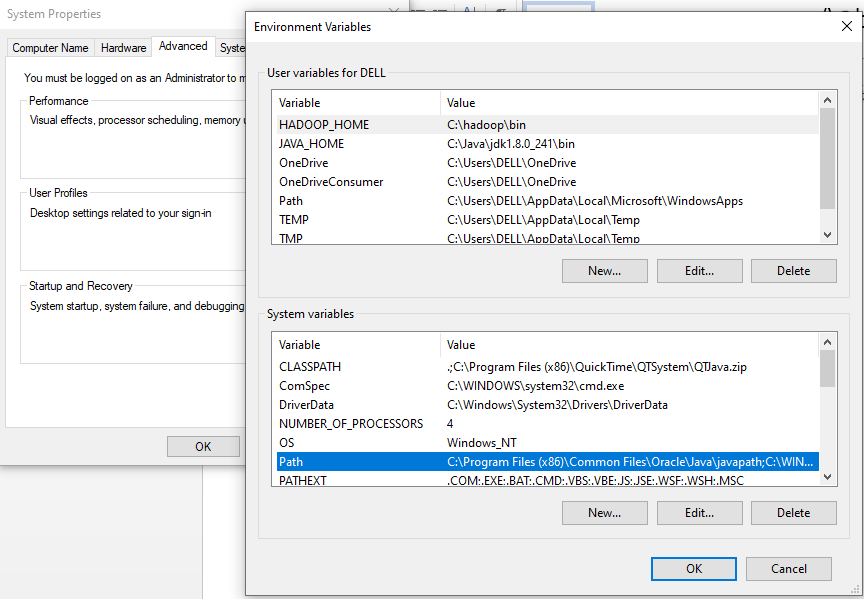


To set path of MongoDB server and shell

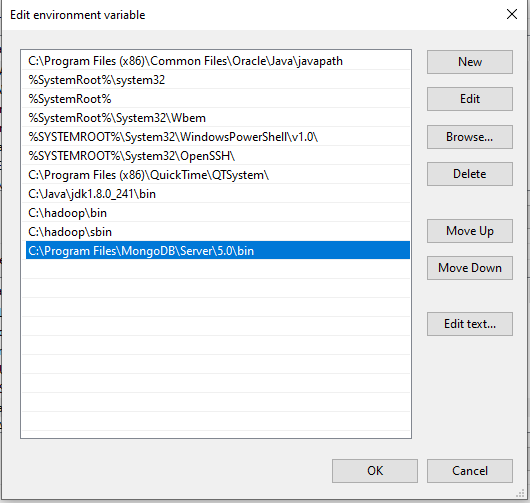
Copy in clipboard: C:\Program Files\MongoDB\Server\5.0\bin

Go to environment variables

Add mongodb path to system path variable



Click on New

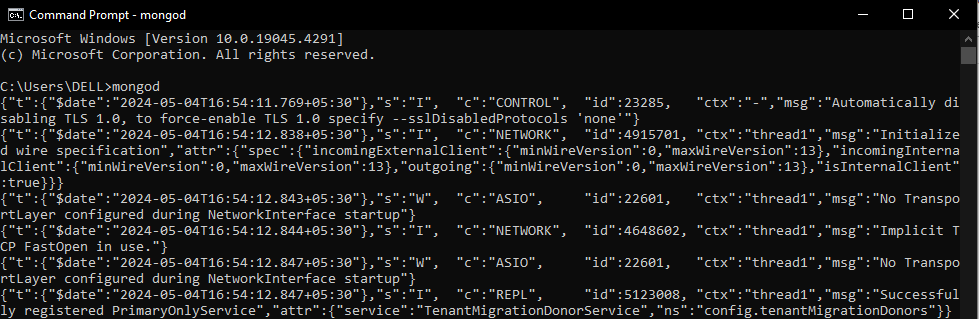


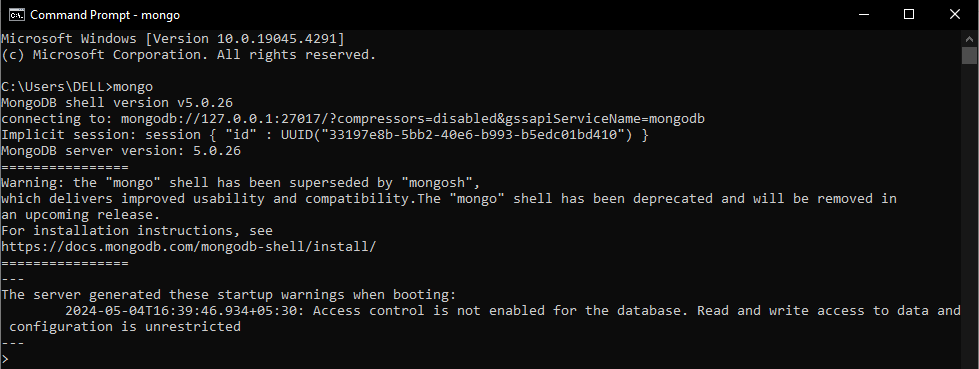
Paste the path

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.

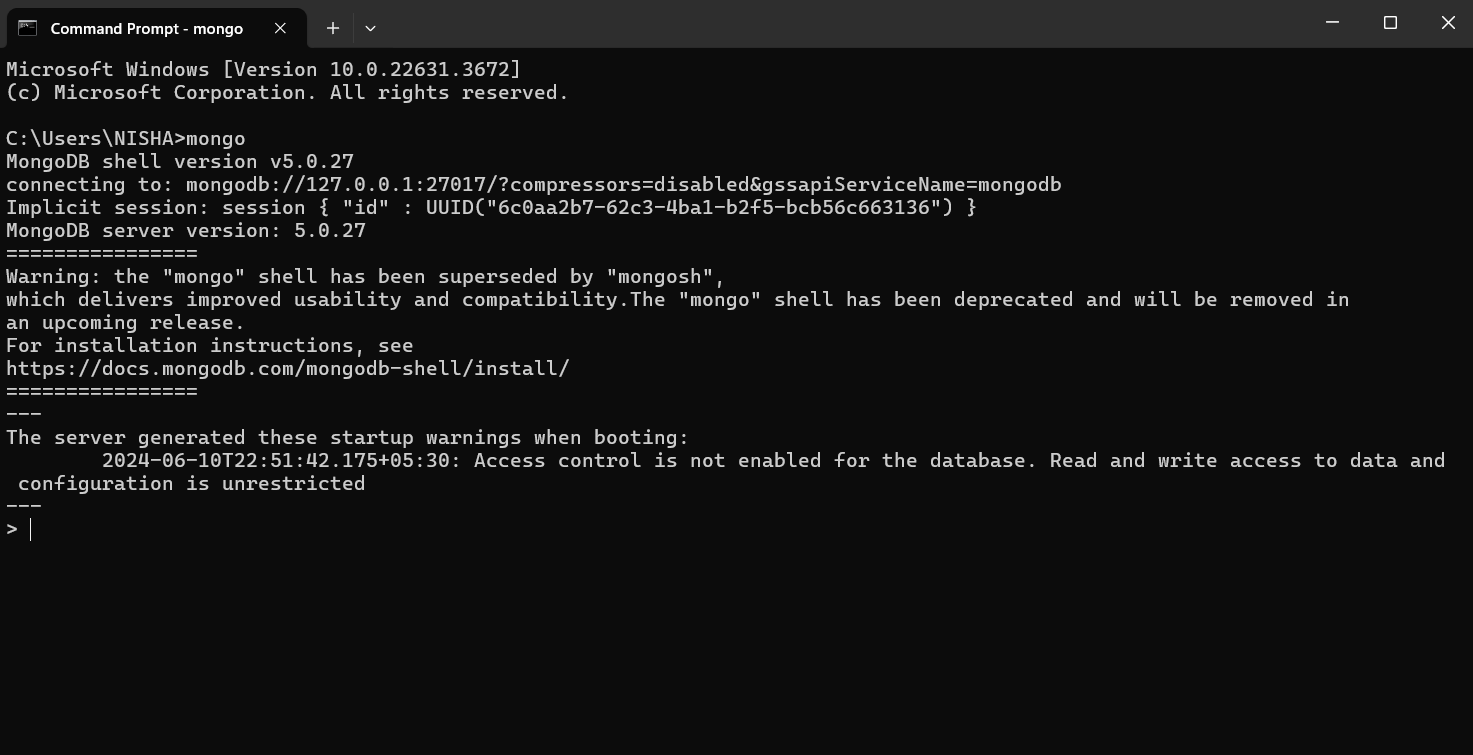
Restarted the machine

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.

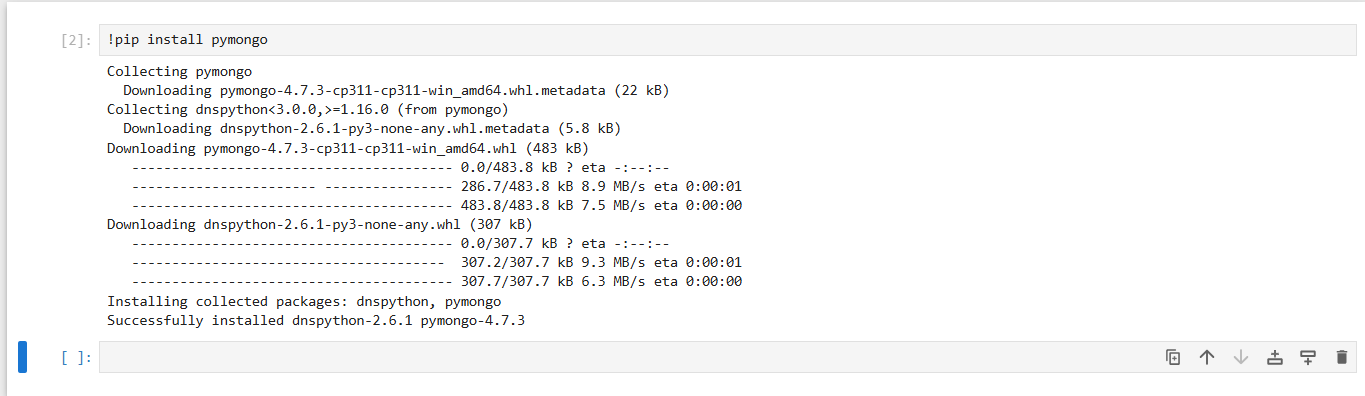


**Step 4: Install MongoDB Python on Windows**

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

the PyMongo library.

Install package to use MongoDBTo 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())

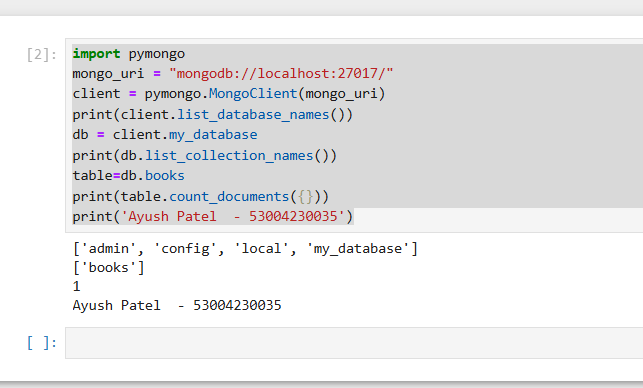
db = client.my\_database

print(db.list\_collection\_names())

table=db.books

print(table.count\_documents({}))

print('Ayush Patel - 53004230035')



**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:**

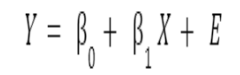
A Regression models are used to describe relationships between variables by fitting a line to the observed data. Regression allows you to estimate how a dependent variable change as the independent variable(s) change.

**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 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()

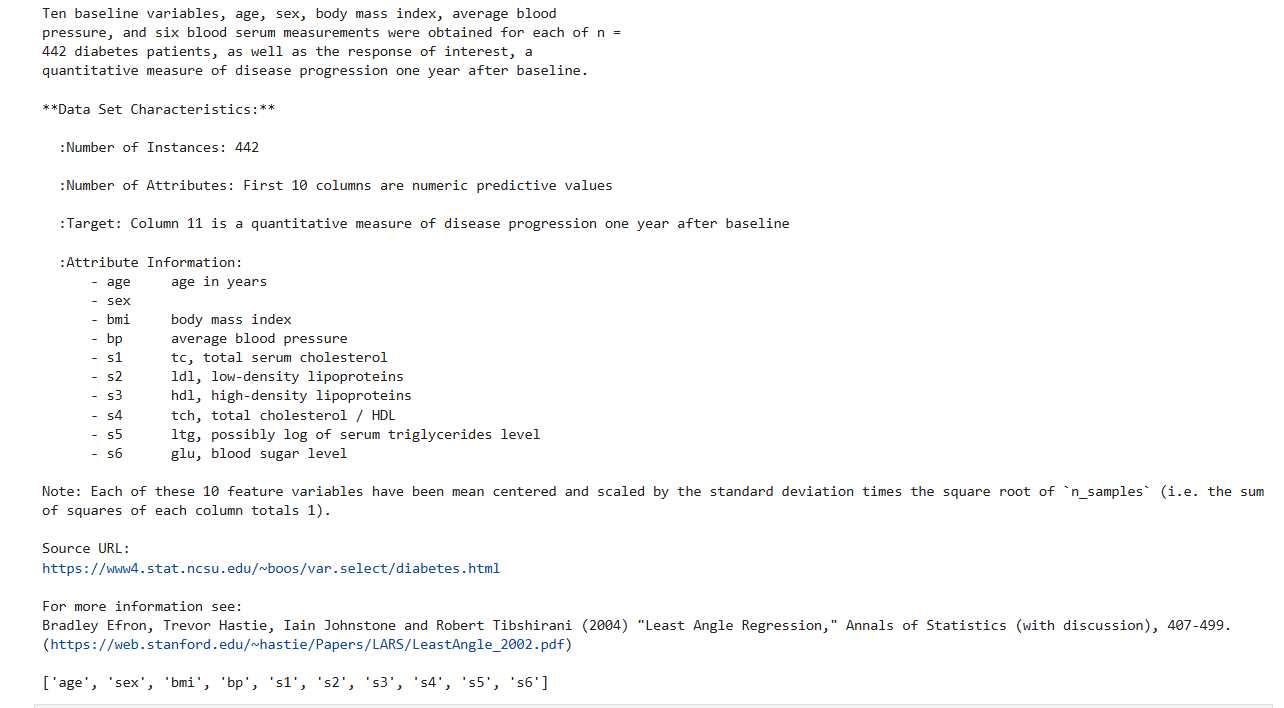
# Load the diabetes dataset

diabetes\_X, diabetes\_y = datasets.load\_diabetes(return\_X\_y = True)

# Description of the dataset

print(diabetes['DESCR'])

print(diabetes.feature\_names)



# 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(())

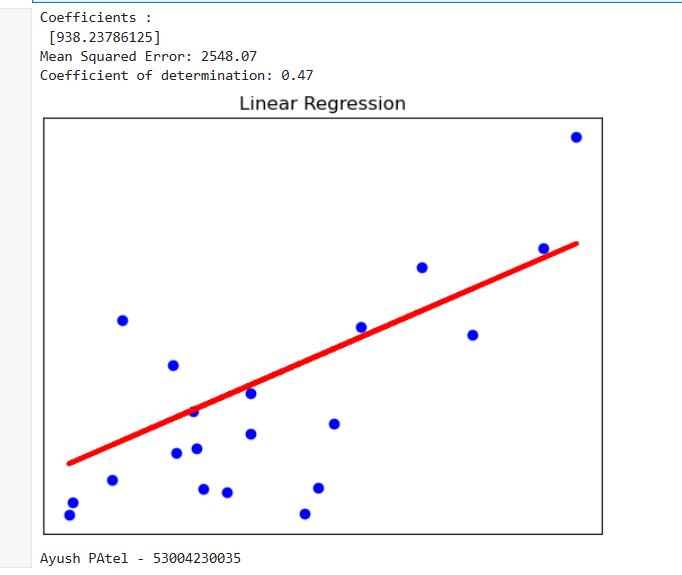
plt.yticks(())

plt.title('Linear Regression')

plt.show()

print('Ayush PAtel - 53004230035')

**OUTPUT**



**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 = 𝛽0 + 𝛽1 𝑋1 + ⋯ + 𝛽𝑛𝑋𝑛 + 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')

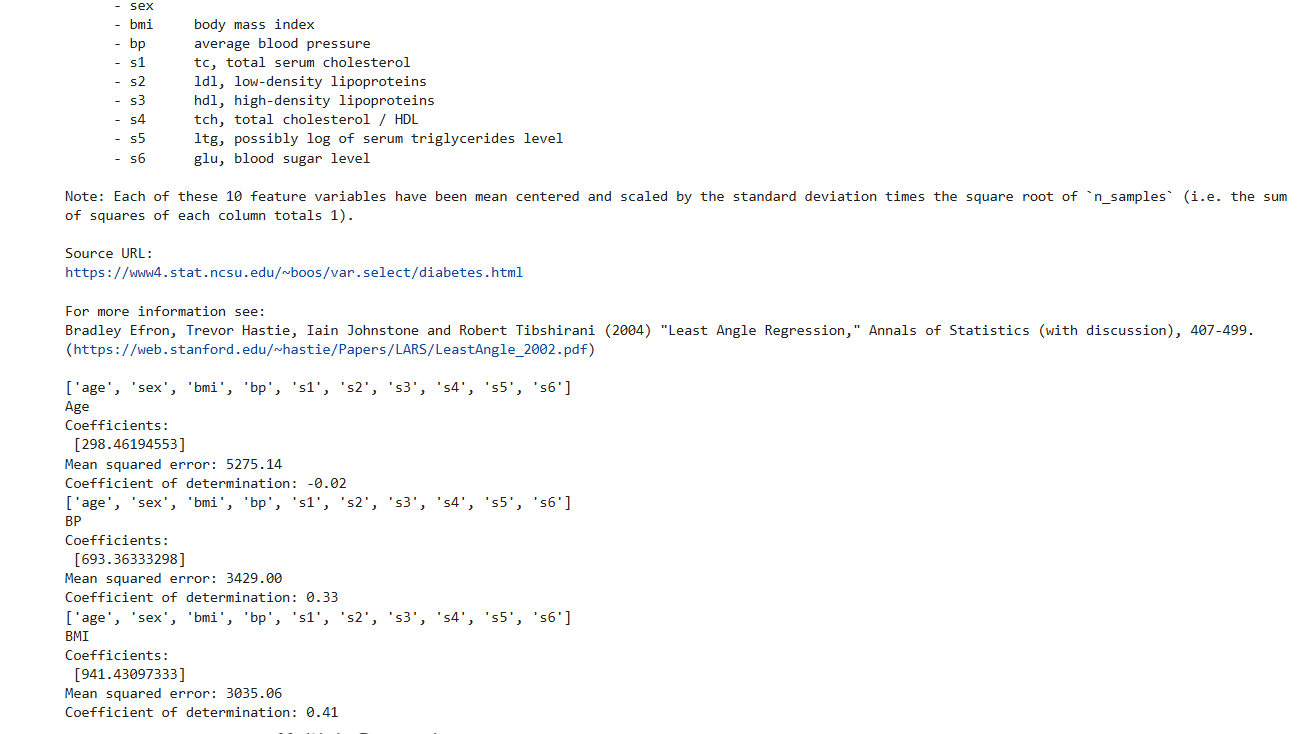
plt.ylabel('Disease Progression')

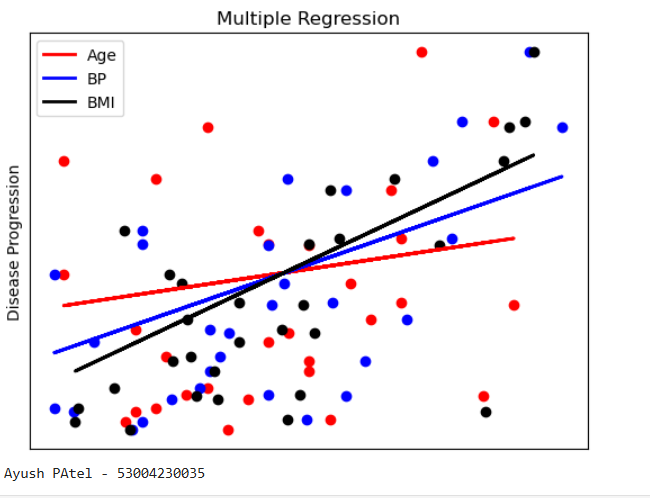
plt.legend()

plt.show()

print('Ayush PAtel - 53004230035')

**OUTPUT**





**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**

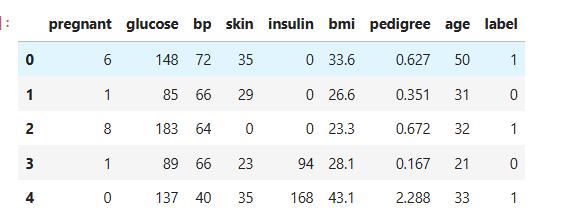
import pandas as pd

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()



# 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



# 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')

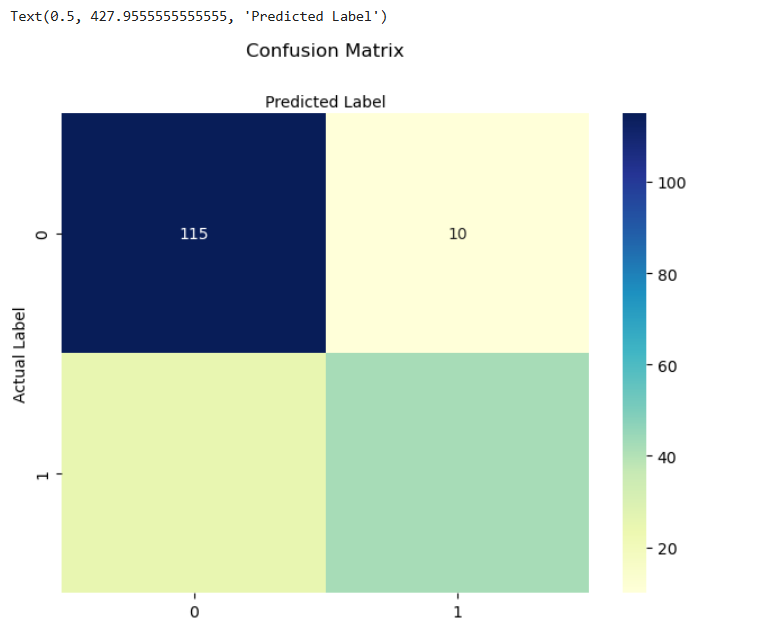
plt.tight\_layout()

plt.title('Confusion Matrix', y=1.1)

plt.ylabel('Actual Label')

plt.xlabel('Predicted Label')

**OUTPUT**

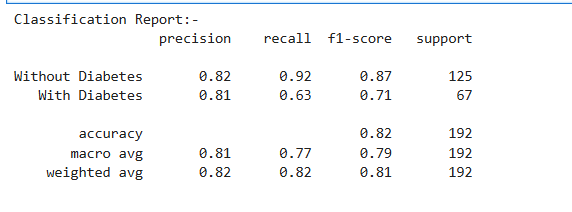


from sklearn.metrics import classification\_report

target\_names = ['Without Diabetes', 'With Diabetes']

print('Classification Report:-')

print(classification\_report(Y\_test,Y\_pred,target\_names=target\_names))



# 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)

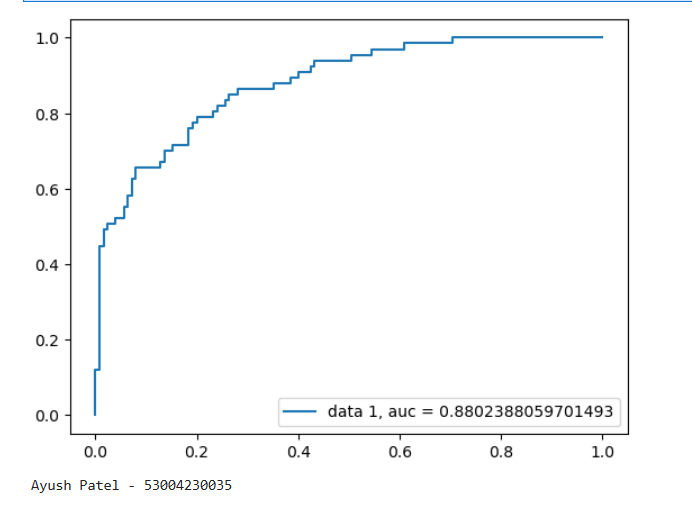
plt.plot(fpr, tpr, label = 'data 1, auc = '+str(auc))

plt.legend(loc=4)

plt.show()

print('Ayush Patel - 53004230035')

**OUTPUT**



**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])

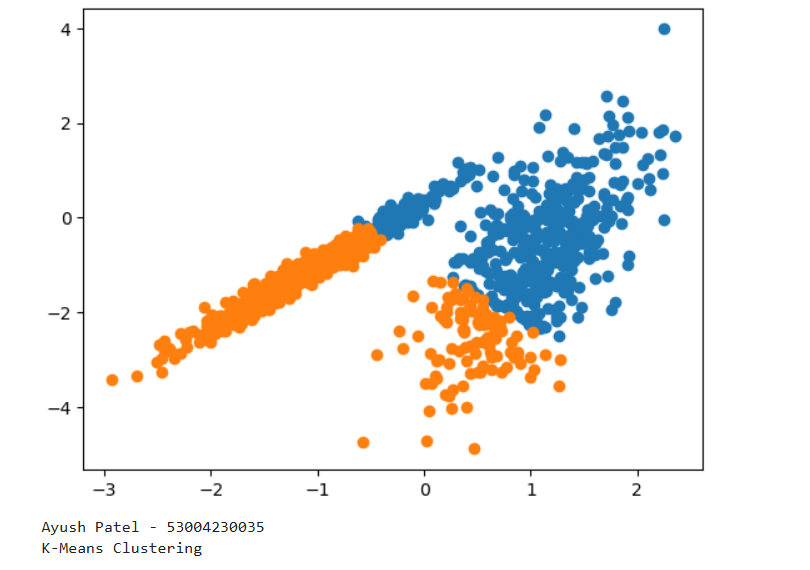
# show the plot

pyplot.show()

print("K-Means Clustering")

print("Ayush Patel - 53004230035")

**OUTPUT**



**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])

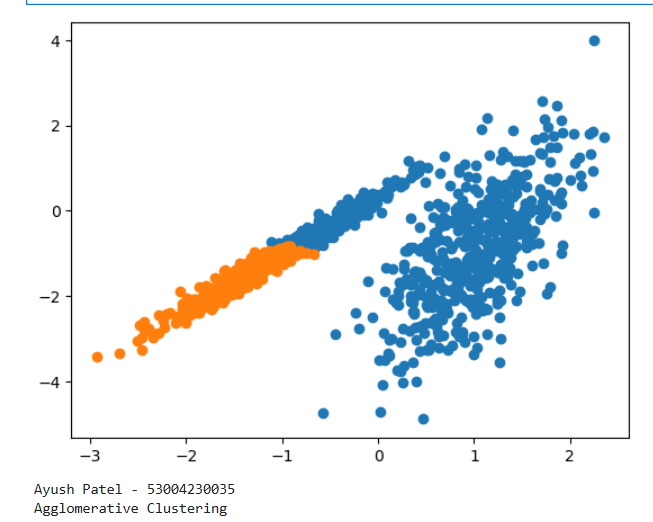
# show the plot

pyplot.show()

print('Ayush Patel - 53004230035')

print('Agglomerative Clustering')

**OUTPUT**



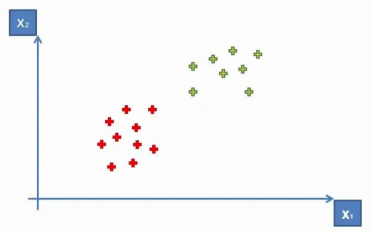
**Practical 7**

**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?

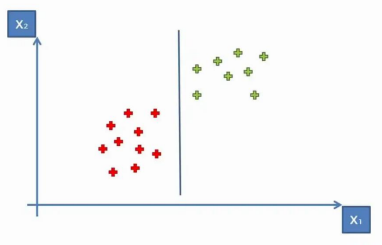
Here, we have two points in two-dimensional space, we have two columns x1 and x2. And we have some observations such as red and green, which are already classified. This is linearly separable data.



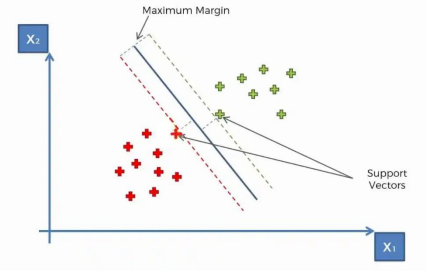
But, now how do we derive a line that separates these points? This means a separation or decision boundary is very important for us when we add new points. So to classify new points, we need to create a boundary between two categories, and when in the future we will add new points and we want to classify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

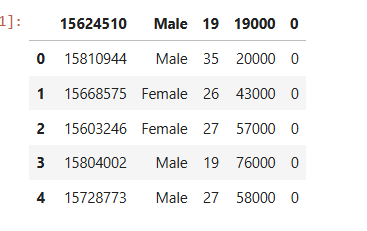
import pandas as pd

import matplotlib.pyplot as plt

# Load Dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

dataset.head()



# 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

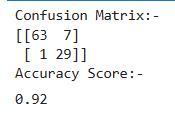
cnf = confusion\_matrix(Y\_test, Y\_pred)

print('Confusion Matrix:-')

print(cnf)

print('Accuracy Score:-')

accuracy\_score(Y\_test, Y\_pred)



# 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')

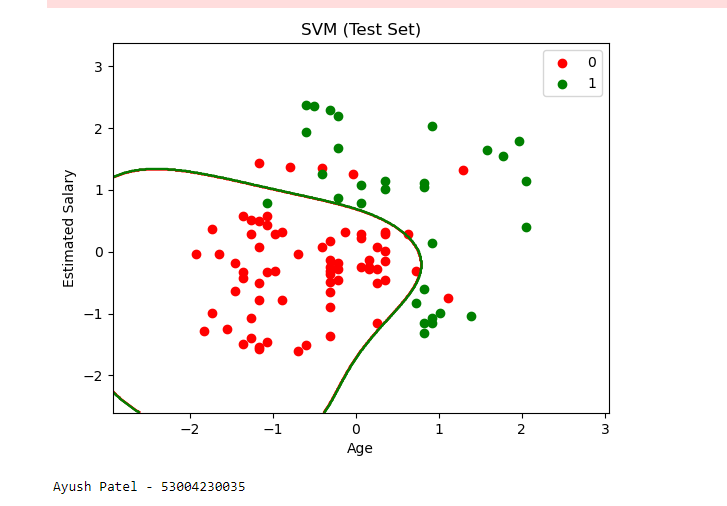
plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

Print(‘Ayush Patel - 53004230035’)

**OUTPUT**

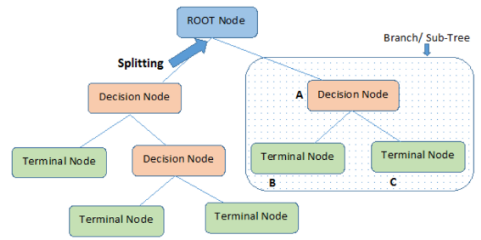


**Practical 8**

**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

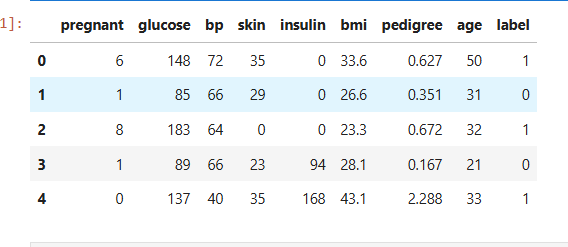
from sklearn.model\_selection import train\_test\_split

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()



# 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('Ayush Patel - 53004230035')



# 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,

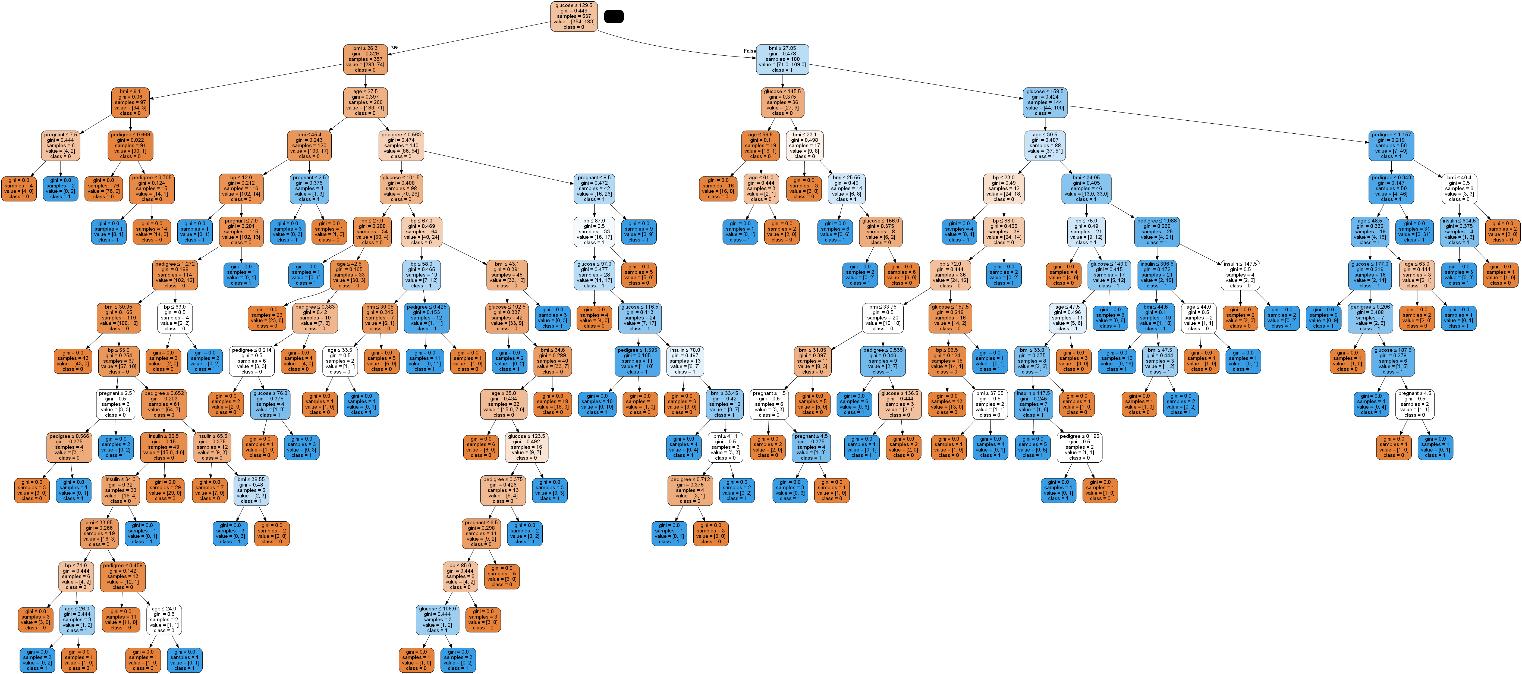
feature\_names=feature\_cols, class\_names=['0','1'])

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(‘Ayush Patel - 53004230035’)

# 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(‘Ayush Patel - 53004230035’)

**OUTPUT**

