Malla Reddy Engineering College (Autonomous)

(An UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad). Accredited 2nd time by NAAC with 'A' Grade, Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad Telangana-500100 www.mrec.ac.in

LABORATORY RECORD

LABORATORY NAME:

LABORATORY CODE:

YEAR/SEMESTER:

REGULATIONS: MR18 (2018-19 ADMITTED BATCH)

DEPARTMENT OF INFORMATION TECHNOLOGY

Malla Reddy Engineering College (Autonomous)

(An UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad). Accredited 2nd time by NAAC with 'A' Grade, Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad Telangana-500100 www.mrec.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

THIS IS TO CERTIFY THAT IT IS BONAFIED RECORD OF LABORATORY WORK DONE BY

MR./MS._____BEARING THE ROLL NUMBER

_____OF_____YEAR/SEMESTER _____

DEPARTMENT IN THE _____LABORATORY

DURING THE ACADEMIC YEAR_____UNDER OUR OBSERVATION.

LAB IN-CHARGE HEAD OF DEPARTMENT

INTERNAL EXAMINER EXTERNAL EXAMINER

Malla Reddy Engineering College (Autonomous)

(An UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad). Accredited 2nd time by NAAC with 'A' Grade, Maisammaguda (H), Medchal-Malkajgiri District, Secunderabad Telangana-500100 www.mrec.ac.in

Department of Information Technology

INDEX

1. Install the python software/Anaconda- python and install useful package for machine learning load the dataset (sample), understand, and visualize the data.

Installing Anaconda on Windows

Installing Anaconda on Windows

For problem solvers, I recommend installing and using the Anaconda distribution of Python. This section details the installation of the Anaconda distribution of Python on Windows 10. I think the Anaconda distribution of Python is the best option for problem solvers who want to use Python. Anaconda is free (although the download is large which can take time) and can be installed on school or work computers where you don't have administrator access or the ability to install new programs. Anaconda comes bundled with about 600 packages pre-installed including **NumPy**, **Matplotlib** and **SymPy**. These three packages are very useful for problem solvers and will be discussed in subsequent chapters.

Follow the steps below to install the Anaconda distribution of Python on Windows.

Steps:

1. Visit Anaconda.com/downloads
2. Select Windows
3. Download the ***.exe*** installer
4. Open and run the ***.exe*** installer
5. Open the **Anaconda Prompt** and run some Python code

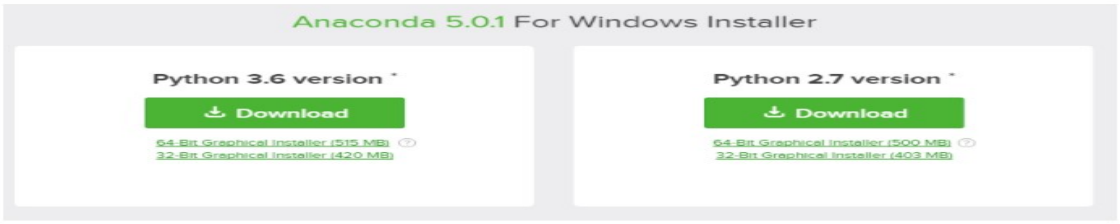
1. Visit the Anaconda downloads page

Go to the following link: Anaconda.com/downloads

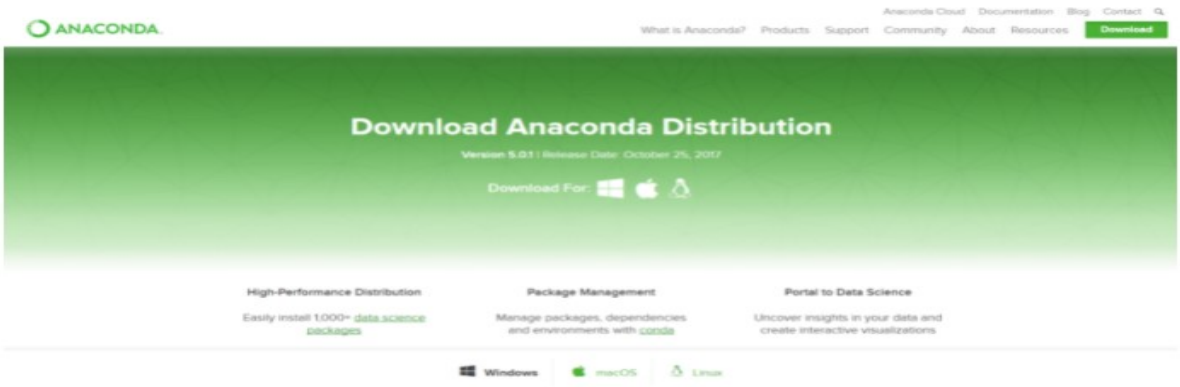
The Anaconda Downloads Page will look something like this:

3. Download

Download the most recent Python 3 release. At the time of writing, the most recent release was the Python 3.6 Version. Python 2.7 is legacy Python. For problem solvers, select the Python 3.6 version. If you are unsure if your computer is running a 64-bit or 32-bit version of Windows, select 64-bit as 64-bit Windows is most common.



You may be prompted to enter your email. You can still download Anaconda if you click `[No Thanks]` and don't enter your Work Email address.

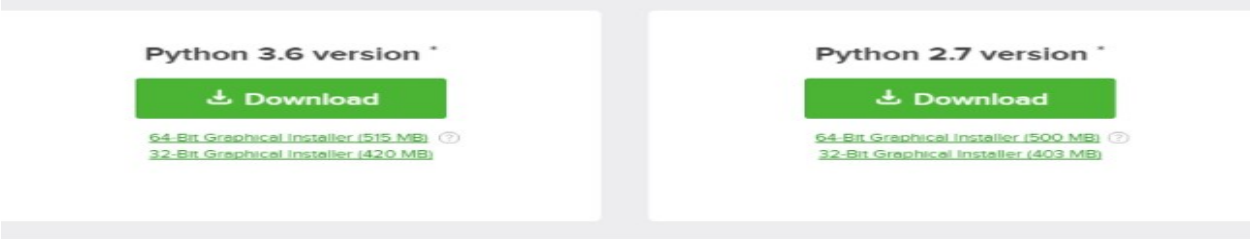


2. Select Windows

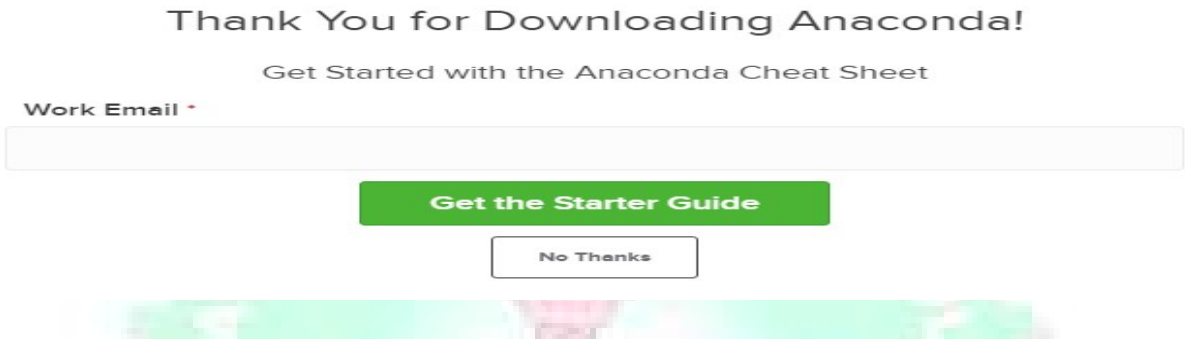
Select Windows where the three operating systems are listed.

3. Download

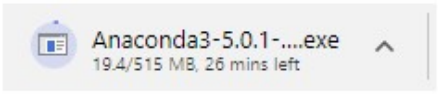
Download the most recent Python 3 release. At the time of writing, the most recent release was the Python 3.6 Version. Python 2.7 is legacy Python. For problem solvers, select the Python 3.6 version. If you are unsure if your computer is running a 64-bit or 32-bit version of Windows, select 64-bit as 64-bit Windows is most common.



u may be prompted to enter your email. You can still download Anaconda if you click [No anks] and don't enter your Work Email address.

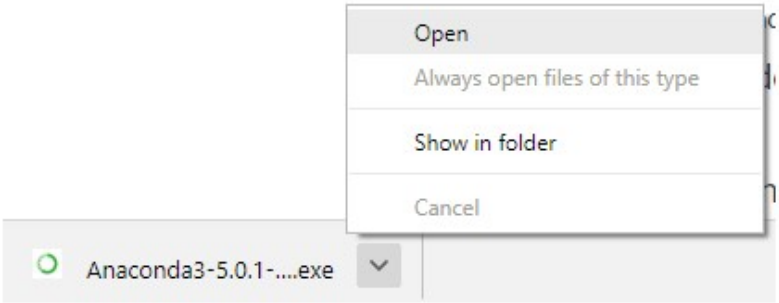


The download is quite large (over 500 MB) so it may take a while to for Anaconda to download

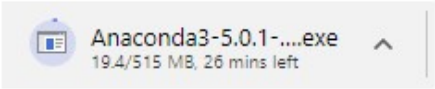


4. Open and run the installer

Once the download completes, open and run the **.exe** installer

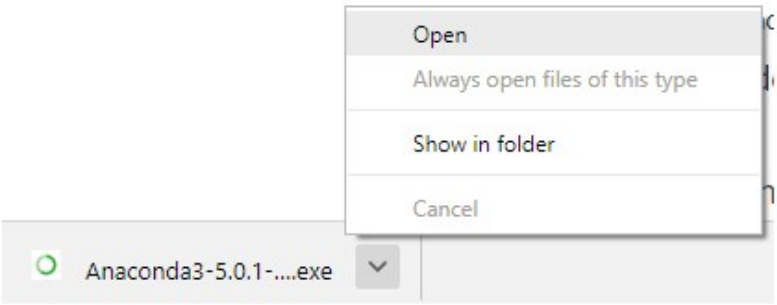


The download is quite large (over 500 MB) so it may take a while to for Anaconda to download



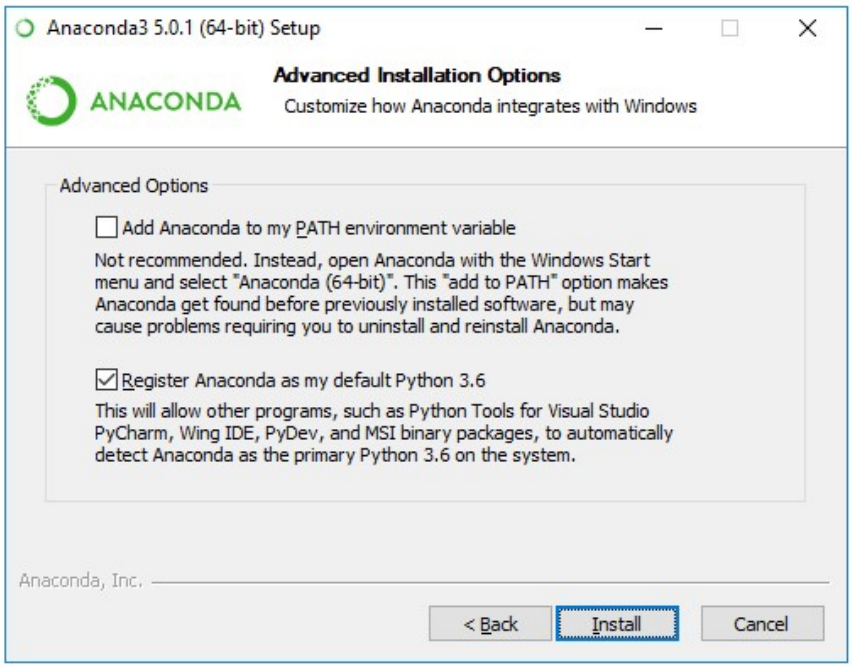
4. Open and run the installer

Once the download completes, open and run the ***.exe*** installer



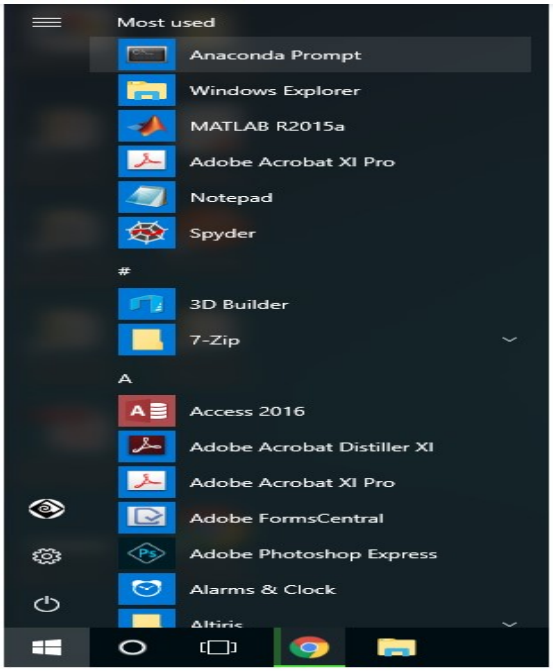
At the beginning of the install, you need to click **Next** to confirm the installation.

At the Advanced Installation Options screen, I recommend that you **do not check** "Add Anaconda to my PATH environment variable"



5. Open the Anaconda Prompt from the Windows start menu

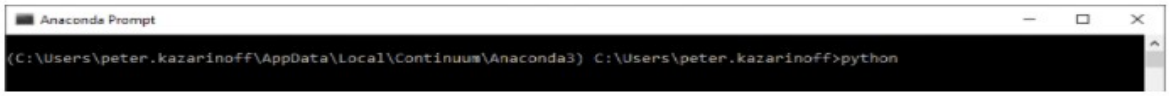
After the installation of Anaconda is complete, you can go to the Windows start menu and select the Anaconda Prompt.



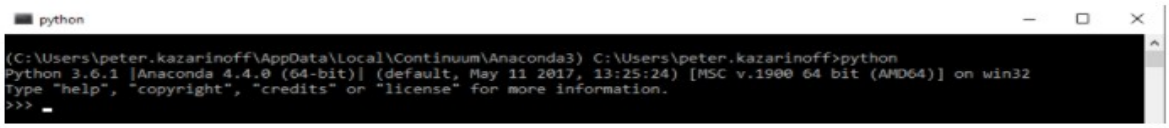
This opens the **Anaconda Prompt**. **Anaconda** is the Python distribution and the **Anaconda Prompt** is a command line shell (a program where you type in commands instead of using a mouse). The black screen and text that makes up the **Anaconda Prompt** doesn't look like much, but it is really helpful for problem solvers using Python.

At the Anaconda prompt, type `python` and hit `[Enter]`. The `python` command starts the Python interpreter, also called the Python REPL (for Read Evaluate Print Loop).

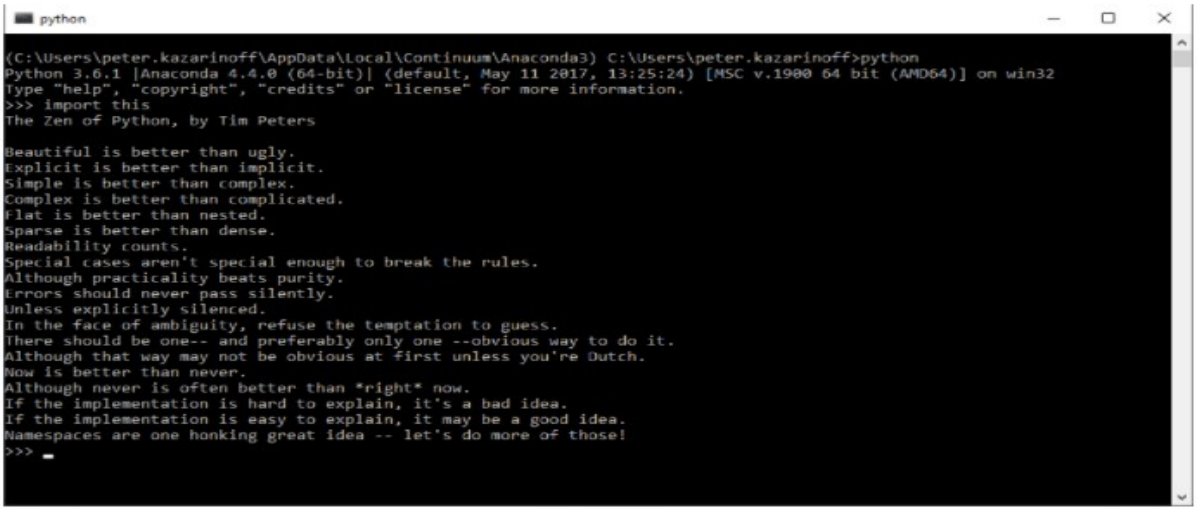
```
> python
```



Note the Python version. You should see something like `Python 3.6.1`. With the interpreter running, you will see a set of greater-than symbols `>>>` before the cursor.



Now you can type Python commands. Try typing `import this`. You should see the ***Zen of Python*** by Tim Peters



To close the Python interpreter, type `exit()` at the prompt `>>>`. Note the double parenthesis at the end of the `exit()` command. The `()` is needed to stop the Python interpreter and get back out to the **Anaconda Prompt**.

To close the **Anaconda Prompt**, you can either close the window with the mouse, or type `exit`, no parenthesis necessary.

When you want to use the Python interpreter again, just click the Windows Start button and select the **Anaconda Prompt** and type `python`.

B) Loading, understanding and visualizing the data set.

```
import pandas as pd
import matplotlib.pyplot as plt
# importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
print(dataset.head())
```

Output:

```
YearsExperience,Salary
1.1,39343.00
1.3,46205.00
1.5,37731.00
2.0,43525.00
2.2,39891.00
2.9,56642.00
3.0,60150.00
3.2,54445.00
3.2,64445.00
3.7,57189.00
3.9,63218.00
4.0,55794.00
4.0,56957.00
4.1,57081.00
4.5,61111.00
4.9,67938.00
5.1,66029.00
5.3,83088.00
5.9,81363.00
6.0,93940.00
6.8,91738.00
7.1,98273.00
7.9,101302.00
8.2,113812.00
8.7,109431.00
```

```
LinearRegression
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/LinearRegression.py
   YearsExperience   Salary
0              1.1  39343.0
1              1.3  46205.0
2              1.5  37731.0
3              2.0  43525.0
4              2.2  39891.0
[ 40748.96184072 122699.62295594  64961.65717022  63099.14214487
 115249.56285456 107799.50275317]
[ 37731. 122391.  57081.  63218. 116969. 109431.]
```

2. Implement simple linear regression

```
import pandas as pd
import matplotlib.pyplot as plt
# importing the dataset
dataset = pd.read_csv('Salary_Data.csv')
print(dataset.head())
# data preprocessing
X = dataset.iloc[:, :-1].values  # independent variable array
y = dataset.iloc[:, 1].values  # dependent variable vector
# splitting the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
# fitting the regression model
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)  # actually produces the linear eqn for the data
# predicting the test set results
y_pred = regressor.predict(X_test)
print(y_pred)
print(y_test)
# visualizing the results
# plot for the TRAIN
plt.scatter(X_train, y_train, color='red')  # plotting the observation line
plt.plot(X_train, regressor.predict(X_train), color='blue')  # plotting the regression line
plt.title("Salary vs Experience (Training set)")  # stating the title of the graph
plt.xlabel("Years of experience")  # adding the name of x-axis
plt.ylabel("Salaries")  # adding the name of y-axis
```

```
plt.show()  # specifies end of graph
# plot for the TEST
plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')  # plotting the regression line
plt.title("Salary vs Experience (Testing set)")
plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```

Output:


```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/LinearRegression.py
   YearsExperience   Salary
0              1.1  39343.0
1              1.3  46205.0
2              1.5  37731.0
3              2.0  43525.0
4              2.2  39891.0
[ 40748.96184072 122699.62295594  64961.65717022  63099.14214487
 115249.56285456 107799.50275317]
[ 37731. 122391.  57081.  63218. 116969. 109431.]
```

3. Implement multivariate linear regression.

```
import pandas

from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]

y = df['CO2']

regr = linear_model.LinearRegression()

regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:

predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

Output:

```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/Regvariant.py
[107.2087328]


Process finished with exit code 0
```

4. Implement simple Logistic Regression and Multivariate Logistic Regression.

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)
print(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
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
# instantiate the model (using the default parameters)
logreg = LogisticRegression()
# fit the model with data
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
# import the metrics class
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
print(cnf_matrix)

#[[trueNeg  falsePos]

# [ falseNeg  truePos]]
```

Output:

```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/LogisticRegression.py
   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
Accuracy: 0.8072916666666666
[[117  13]
 [ 24  38]]
```

5. Implement Decision Trees.

```
# Load libraries

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

# Import Decision Tree Classifier

from sklearn.model_selection import train_test_split

# Import train_test_split function

from sklearn import metrics

#Import scikit-learn metrics module for accuracy calculation

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

# load dataset

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

print(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 dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test

# Create Decision Tree classifier object

#clf = DecisionTreeClassifier()
```

```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# 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))
```

Output:

```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/DecisionTree.py
   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
Accuracy: 0.7705627705627706

Process finished with exit code 0
```

6. Implement any 3 Classification Algorithms.

```
import pandas as pd

import matplotlib.pyplot as plt

# importing the dataset

dataset = pd.read_csv('Salary_Data.csv')

print(dataset.head())

# data preprocessing

X = dataset.iloc[:, :-1].values  # independent variable array

y = dataset.iloc[:, 1].values  # dependent variable vector

# splitting the dataset

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)

# fitting the regression model

from sklearn.linear_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X_train, y_train)  # actually produces the linear eqn for the data

# predicting the test set results

y_pred = regressor.predict(X_test)

print(y_pred)

print(y_test)

# visualizing the results

# plot for the TRAIN

plt.scatter(X_train, y_train, color='red')  # plotting the observation line
```

```
plt.plot(X_train, regressor.predict(X_train), color='blue')  # plotting the regression line

plt.title("Salary vs Experience (Training set)")  # stating the title of the graph

plt.xlabel("Years of experience")  # adding the name of x-axis

plt.ylabel("Salaries")  # adding the name of y-axis

plt.show()  # specifies end of graph

# plot for the TEST

plt.scatter(X_test, y_test, color='red')

plt.plot(X_train, regressor.predict(X_train), color='blue')  # plotting the regression line

plt.title("Salary vs Experience (Testing set)")

plt.xlabel("Years of experience")

plt.ylabel("Salaries")

plt.show()
```

Output:

```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/LinearRegression.py
   YearsExperience   Salary
0              1.1  39343.0
1              1.3  46205.0
2              1.5  37731.0
3              2.0  43525.0
4              2.2  39891.0
[ 40748.96184072 122699.62295594  64961.65717022  63099.14214487
 115249.56285456 107799.50275317]
[ 37731. 122391.  57081.  63218. 116969. 109431.]
```

```
import pandas

from sklearn import linear_model

df = pandas.read_csv("cars.csv")

X = df[['Weight', 'Volume']]

y = df['CO2']

regr = linear_model.LinearRegression()

regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:

predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/Regvariant.py
[107.2087328]


Process finished with exit code 0
```

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
# load dataset
pima = pd.read_csv("diabetes.csv", header=None, names=col_names)
print(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
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
# instantiate the model (using the default parameters)
logreg = LogisticRegression()
# fit the model with data
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
# import the metrics class
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print(cnf_matrix)
```

#[[trueNeg falsePos]

[falseNeg truePos]]

```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/LogisticRegression.py
   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
Accuracy: 0.8072916666666666
[[117  13]
 [ 24  38]]
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/DecisionTree.py
   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
Accuracy: 0.7705627705627706

Process finished with exit code 0
```

7. Implement Random Forests Algorithm

```
#Import scikit-learn dataset library

from sklearn import datasets

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics

#Load dataset

iris = datasets.load_iris()

# print the label species(setosa, versicolor,virginica)

print(iris.target_names)

# print the names of the four features

print(iris.feature_names)

# print the iris data (top 5 records)

print(iris.data[0:5])

# print the iris labels (0:setosa, 1:versicolor, 2:virginica)

print(iris.target)

# Creating a DataFrame of given iris dataset.

data=pd.DataFrame({

    'sepal length':iris.data[:,0],

    'sepal width':iris.data[:,1],

    'petal length':iris.data[:,2],

    'petal width':iris.data[:,3],
```

```
    'species':iris.target

})

data.head()

X=data[['sepal length', 'sepal width', 'petal length', 'petal width']]  # Features

y=data['species']  # Labels

# Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test

#Import Random Forest Model

#Create a Gaussian Classifier

clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)

clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation

# Model Accuracy, how often is the classifier correct?

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output:

```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/randomforest.py
['setosa' 'versicolor' 'virginica']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
Accuracy: 0.9333333333333333

Process finished with exit code 0
```

8. Implement K-Means, KNN algorithms

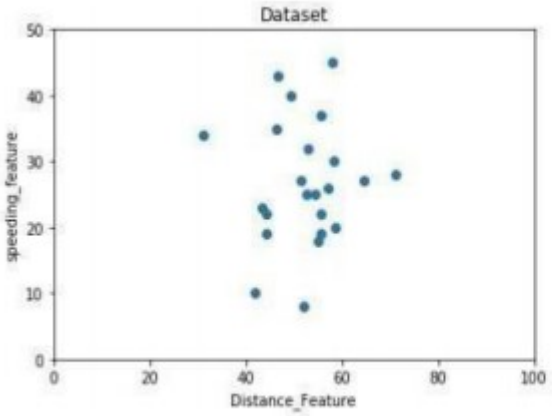
```
#Create artificial data set

from sklearn.datasets import make_blobs

raw_data = make_blobs(n_samples = 200, n_features = 2, centers = 4, cluster_std = 1.8)

#Data imports

import pandas as pd

import numpy as np

#Visualization imports

import matplotlib.pyplot as plt

#Visualize the data

plt.scatter(raw_data[0][:,0], raw_data[0][:,1])

plt.scatter(raw_data[0][:,0], raw_data[0][:,1], c=raw_data[1])

#Build and train the model

from sklearn.cluster import KMeans

model = KMeans(n_clusters=4)

model.fit(raw_data[0])

#See the predictions

print(model.labels_)

print(model.cluster_centers_)

#PLot the predictions against the original data set

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))

ax1.set_title('Our Model')

ax1.scatter(raw_data[0][:,0], raw_data[0][:,1],c=model.labels_)
```

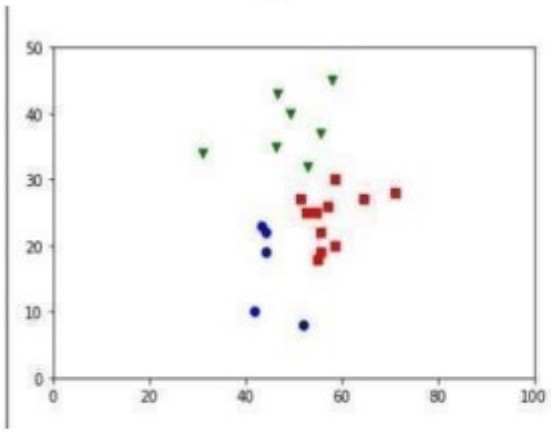
ax2.set_title('Original Data')

ax2.scatter(raw_data[0][:,0], raw_data[0][:,1],c=raw_data[1])

Output:

```
C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/K-means.py
[2 1 0 2 3 1 1 1 0 0 2 2 3 0 2 3 1 0 1 0 3 1 1 3 0 2 0 1 2 3 0 1 0 1 2 3 2
 2 1 1 2 0 0 1 0 3 0 0 3 3 0 1 3 3 2 3 0 2 2 0 2 0 2 2 3 2 0 0 0 1 2 0 2 0
 3 3 0 1 3 3 2 3 0 2 0 2 0 0 2 2 3 2 3 3 2 1 1 2 3 3 2 1 3 1 1 2 2 3 2 0 2
 1 3 2 2 0 2 1 0 3 1 3 3 0 2 3 0 3 1 3 0 1 0 0 2 0 2 1 0 0 3 2 1 1 1 1 1 2
 3 3 2 2 0 1 3 3 1 3 3 0 3 0 2 2 1 3 0 1 0 1 3 1 2 1 1 1 1 2 3 0 1 0 1 1 0
 3 3 2 3 3 3 3 1 1 3 2 3 1 1 3]
[[-9.03095821 -2.38226723]
 [-2.01570905 -4.66737823]
 [-5.57327067  3.63652207]
 [-7.56873301 -6.99238433]]

Process finished with exit code 0
```





```
#Common imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#Import the data set
raw_data = pd.read_csv('classified_data.csv', index_col = 0)
print(raw_data.columns)
#Import standardization functions from scikit-learn
from sklearn.preprocessing import StandardScaler
#Standardize the data set
scaler = StandardScaler()
scaler.fit(raw_data.drop('TARGET CLASS', axis=1))
scaled_features = scaler.transform(raw_data.drop('TARGET CLASS', axis=1))
scaled_data = pd.DataFrame(scaled_features, columns = raw_data.drop('TARGET CLASS', axis=1).columns)
#Split the data set into training data and test data
from sklearn.model_selection import train_test_split
x = scaled_data
y = raw_data['TARGET CLASS']
x_training_data, x_test_data, y_training_data, y_test_data = train_test_split(x, y, test_size = 0.3)
#Train the model and make predictions
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors = 1)
model.fit(x_training_data, y_training_data)
predictions = model.predict(x_test_data)
#Performance measurement
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
print(classification_report(y_test_data, predictions))
print(confusion_matrix(y_test_data, predictions))
#Selecting an optimal K value
error_rates = []
for i in np.arange(1, 101):
new_model = KNeighborsClassifier(n_neighbors = i)
new_model.fit(x_training_data, y_training_data)
new_predictions = new_model.predict(x_test_data)
error_rates.append(np.mean(new_predictions != y_test_data))
plt.figure(figsize=(16,12))
plt.plot(error_rates)
```

Output:

```
#Common imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

C:\Users\User\PycharmProjects\pythonProject\venv\Scripts\python.exe C:/Users/User/PycharmProjects/pythonProject/knearestneighbour.py
Index(['WTT', 'PTI', 'EQW', 'SBI', 'LQE', 'QWG', 'FDJ', 'PJF', 'HQE', 'NXJ',
       'TARGET CLASS'],
      dtype='object')
              precision    recall  f1-score   support

           0       0.95      0.86      0.90       145
           1       0.88      0.96      0.92       155

    accuracy                           0.91       300
   macro avg       0.92      0.91      0.91       300
weighted avg       0.91      0.91      0.91       300

[[124  21]
 [  6 149]]

Process finished with exit code 0
```

9. Implement SVM on any applicable datasets.

```
#Import scikit-learn dataset library

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn import svm

from sklearn import metrics

#Load dataset

cancer = datasets.load_breast_cancer()

# print the names of the 13 features

print("Features: ", cancer.feature_names)

# print the label type of cancer('malignant' 'benign')

print("Labels: ", cancer.target_names)

# print data(feature)shape

cancer.data.shape

# print the cancer data features (top 5 records)

print(cancer.data[0:5])

# print the cancer labels (0:malignant, 1:benign)

print(cancer.target)

# Split dataset into training set and test set

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_size=0.3,random_state=109) # 70% training and 30% test

#Create a svm Classifier

clf = svm.SVC(kernel='linear') # Linear Kernel
```

#Train the model using the training sets

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))

Model Precision: what percentage of positive tuples are labeled as such?

print("Precision:",metrics.precision_score(y_test, y_pred))

Model Recall: what percentage of positive tuples are labelled as such?

print("Recall:",metrics.recall_score(y_test, y_pred))

Output:

```
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Labels:  ['malignant' 'benign']
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
 [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
  1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
  1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
  1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
  2.364e-01 7.678e-02]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
Accuracy: 0.9649122807017544
Precision: 0.9811320754716981
Recall: 0.9629629629629629

Process finished with exit code 0
```

10. Implement Neural Networks

```
import numpy as np

class NeuralNetwork():

    def __init__(self):

  # seeding for random number generation

np.random.seed(1)

        # converting weights to a 3 by 1 matrix with values from -1 to 1 and mean of 0

self.synaptic_weights = 2 * np.random.random((3, 1)) - 1

    def sigmoid(self, x):

        # applying the sigmoid function

        return 1 / (1 + np.exp(-x))

    def sigmoid_derivative(self, x):

        # computing derivative to the Sigmoid function

        return x * (1 - x)

    def train(self, training_inputs, training_outputs, training_iterations):

        # training the model to make accurate predictions while adjusting weights continually

        for iteration in range(training_iterations):

            # siphon the training data via  the neuron

            output = self.think(training_inputs)

            # computing error rate for back-propagation

            error = training_outputs - output

            # performing weight adjustments

            adjustments = np.dot(training_inputs.T, error * self.sigmoid_derivative(output))
```

```
self.synaptic_weights += adjustments
    def think(self, inputs):
        # passing the inputs via the neuron to get output
        # converting values to floats
        inputs = inputs.astype(float)
      output = self.sigmoid(np.dot(inputs, self.synaptic_weights))
        return output
                if __name__ == "__main__":
    # initializing the neuron class
neural_network = NeuralNetwork()
print("Beginning Randomly Generated Weights: ")
    print(neural_network.synaptic_weights)
    # training data consisting of 4 examples--3 input values and 1 output
training_inputs = np.array([[0, 0, 1],
                            [1, 1, 1],
                            [1, 0, 1],
                            [0, 1, 1]])
training_outputs = np.array([[0, 1, 1, 0]]).T
    # training taking place
neural_network.train(training_inputs, training_outputs, 15000)
print("Ending Weights After Training: ")
    print(neural_network.synaptic_weights)
user_input_one = str(input("User Input One: "))
```

```
user_input_two = str(input("User Input Two: "))

user_input_three = str(input("User Input Three: "))

print("Considering New Situation: ", user_input_one, user_input_two, user_input_three)

print("New Output data: ")

print(neural_network.think(np.array([user_input_one, user_input_two, user_input_three])))

print("Wow, we did it!")
```

Output:

```
Beginning Randomly Generated Weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Ending Weights After Training:
[[10.08740896]
 [-0.20695366]
 [-4.83757835]]
Considering New Situation:  1 0 0
New Output data:
[0.9999584]
Wow, we did it!
User Input One: 1
User Input Two: 0
User Input Three: 0
```

11. Implement PCA.

```
# import all libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

#import the breast _cancer dataset

from sklearn.datasets import load_breast_cancer

data=load_breast_cancer()

data.keys()

# Check the output classes

print(data['target_names'])

# Check the input attributes

print(data['feature_names'])

# construct a dataframe using pandas

df1=pd.DataFrame(data['data'],columns=data['feature_names'])

# Scale data befor applying PCA

scaling=StandardScaler()

# Use fit and transform method

scaling.fit(df1)

Scaled_data=scaling.transform(df1)
```

```
# Set the n_components=3

principal=PCA(n_components=3)

principal.fit(Scaled_data)

x=principal.transform(Scaled_data)

# Check the dimensions of data after PCA

print(x.shape)

# Check the values of eigen vectors

# prodeced by principal components

principal.components_

plt.figure(figsize=(10,10))

plt.scatter(x[:,0],x[:,1],c=data['target'],cmap='plasma')

plt.xlabel('pc1')

plt.ylabel('pc2')

# immport relevant libraries for 3d graph

from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10,10))

# choose projection 3d for creating a 3d graph

axis = fig.add_subplot(111, projection='3d')

# x[:,0]is pc1,x[:,1] is pc2 while x[:,2] is pc3

axis.scatter(x[:,0],x[:,1],x[:,2], c=data['target'],cmap='plasma')

axis.set_xlabel("PC1", fontsize=10)

axis.set_ylabel("PC2", fontsize=10)

axis.set_zlabel("PC3", fontsize=10)
```

check how much variance is explained by each principal component

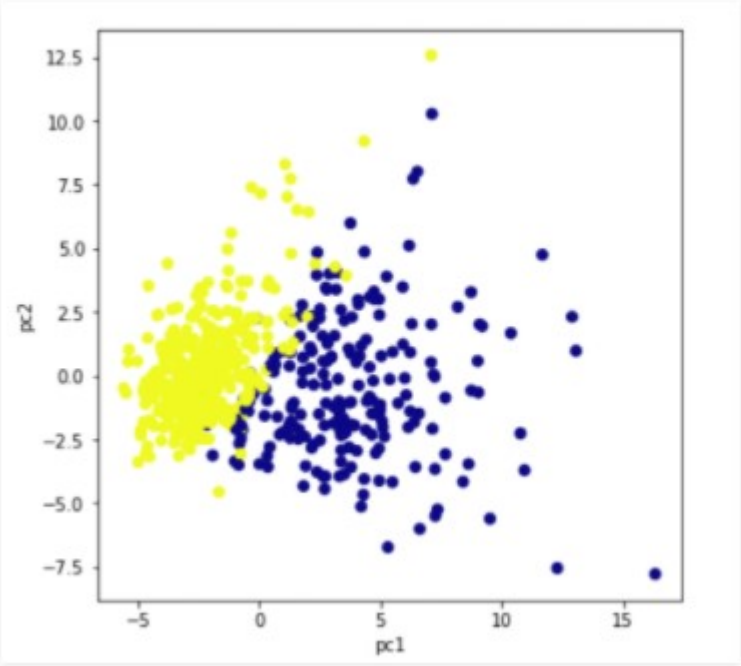
print(principal.explained_variance_ratio_)

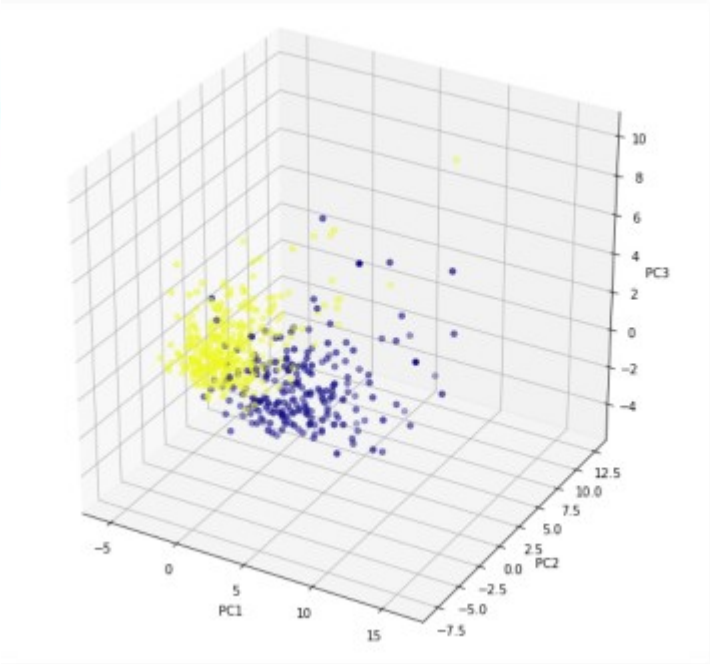
Output:

```
['malignant' 'benign']
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

(569,3)

```
array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
         0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
         0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
         0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
         0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
         0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
       [-0.23385714, -0.05970607, -0.21518137, -0.23107671,  0.18611299,
         0.15189162,  0.06016539, -0.03476746,  0.19034873,  0.36657553,
        -0.10555215,  0.08997966, -0.08945722, -0.15229264,  0.20443046,
         0.23271583,  0.19720726,  0.13032159,  0.18384804,  0.280092  ,
        -0.21986639, -0.04546731, -0.19987843, -0.21935187,  0.17230433,
         0.14359315,  0.09796412, -0.0082572 ,  0.14188334,  0.27533949],
       [-0.0085311 ,  0.06454953, -0.00931412,  0.02869958, -0.10429131,
        -0.07409179,  0.00273334, -0.02556417, -0.04023923, -0.02257508,
         0.26848142,  0.37463403,  0.26664519,  0.2160067 ,  0.30883881,
         0.15478074,  0.17646406,  0.22465714,  0.28858361,  0.21150428,
        -0.04750683, -0.04229772, -0.04854641, -0.01190219, -0.25979723,
        -0.23607529, -0.17305746, -0.17034465, -0.27131251, -0.23279161]])
```


12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
pc2
−5
0
5
10
15
pc1


10
8
6
4
2
0
−2
−4
PC3
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
PC2
−5
0
5
10
15
PC1

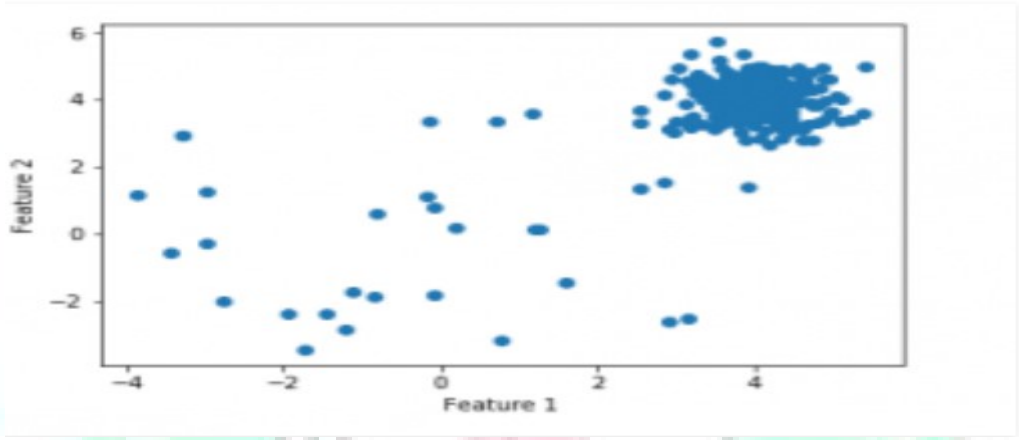
12. Implement anomaly detection and recommendation

```
# import pandas library

import pandas as pd

 # Get the data

column_names = ['user_id', 'item_id', 'rating', 'timestamp']

 path = 'https://media.geeksforgeeks.org/wp-content/uploads/file.tsv'

df = pd.read_csv(path, sep='\t', names=column_names)

# Check the head of the data

df.head()

# Check out all the movies and their respective IDs

movie_titles=pd.read_csv('https://media.geeksforgeeks.org/wp-
content/uploads/Movie_Id_Titles.csv')

movie_titles.head()

data = pd.merge(df, movie_titles, on='item_id')

data.head()

# Calculate mean rating of all movies

data.groupby('title')['rating'].mean().sort_values(ascending=False).head()

# Calculate count rating of all movies

data.groupby('title')['rating'].count().sort_values(ascending=False).head()

# creatingdataframe with 'rating' count values

ratings = pd.DataFrame(data.groupby('title')['rating'].mean())

ratings['num of ratings'] = pd.DataFrame(data.groupby('title')['rating'].count())

ratings.head()
```

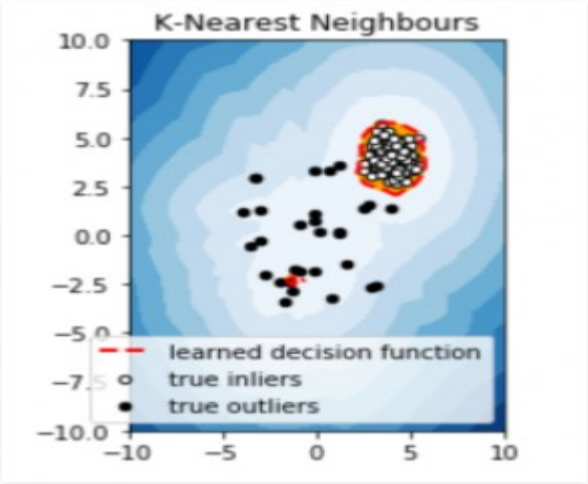
```
import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('white')

%matplotlib inline

 # plot graph of 'num of ratings column'

plt.figure(figsize =(10, 4))

ratings['num of ratings'].hist(bins = 70)

# plot graph of 'ratings' column

plt.figure(figsize =(10, 4))

ratings['rating'].hist(bins = 70)

# Sorting values according to

# the 'num of rating column'

moviemat = data.pivot_table(index ='user_id', columns ='title', values ='rating')

moviemat.head()

ratings.sort_values('num of ratings', ascending = False).head(10)

# analysing correlation with similar movies

starwars_user_ratings = moviemat['Star Wars (1977)']

liarliar_user_ratings = moviemat['Liar Liar (1997)']

starwars_user_ratings.head()

# analysing correlation with similar movies

similar_to_starwars = moviemat.corrwith(starwars_user_ratings)

similar_to_liarliar = moviemat.corrwith(liarliar_user_ratings)
```

```
corr_starwars = pd.DataFrame(similar_to_starwars, columns =['Correlation'])

corr_starwars.dropna(inplace = True)

corr_starwars.head()
```

Output:




```
The number of prediciton errors are 1
```



Anomaly recommendation

```
# import pandas library

import pandas as pd

 # Get the data

column_names = ['user_id', 'item_id', 'rating', 'timestamp']

  path = 'https://media.geeksforgeeks.org/wp-content/uploads/file.tsv'

 df = pd.read_csv(path, sep='\t', names=column_names)

 # Check the head of the data

df.head()

# Check out all the movies and their respective IDs

movie_titles = pd.read_csv('https://media.geeksforgeeks.org/wp-content/uploads/Movie_Id_Titles.csv')

movie_titles.head()

data = pd.merge(df, movie_titles, on='item_id')

data.head()

# Calculate mean rating of all movies

data.groupby('title')['rating'].mean().sort_values(ascending=False).head()

# Calculate count rating of all movies

data.groupby('title')['rating'].count().sort_values(ascending=False).head()

# creating dataframe with 'rating' count values

ratings = pd.DataFrame(data.groupby('title')['rating'].mean())

  ratings['num of ratings'] = pd.DataFrame(data.groupby('title')['rating'].count())

  ratings.head()
```

```
import matplotlib.pyplot as plt

import seaborn as sns

  sns.set_style('white')

%matplotlib inline

 # plot graph of 'num of ratings column'

plt.figure(figsize =(10, 4))

  ratings['num of ratings'].hist(bins = 70)# plot graph of 'ratings' column

plt.figure(figsize =(10, 4))

  ratings['rating'].hist(bins = 70)
```

Output:

	user_id	item_id	rating	timestamp
0	0	50	5	881250949
1	0	172	5	881250949
2	0	133	1	881250949
3	196	242	3	881250949
4	186	302	3	891717742

	item_id	title
0	1	Toy Story (1995)
1	2	GoldenEye (1995)
2	3	Four Rooms (1995)
3	4	Get Shorty (1995)
4	5	Copycat (1995)

	user_id	item_id	rating	timestamp	title_x	title_y	title_x	title_y	title_x	title_y	title_x	title_y	title_x	title_y	title
0	0	50	5	881250949	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)
1	290	50	5	880473582	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)
2	79	50	4	891271545	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)
3	2	50	5	888552084	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)
4	8	50	5	879362124	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)	Star Wars (1977)

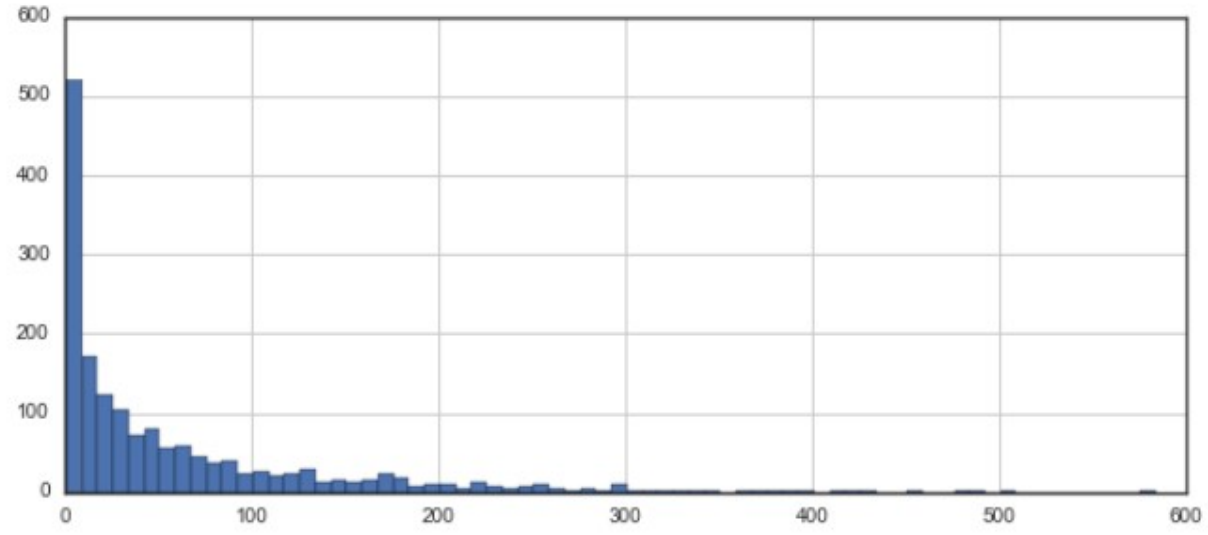
```
title
Marlene Dietrich: Shadow and Light (1996)     5.0
Prefontaine (1997)                            5.0
Santa with Muscles (1996)                     5.0
Star Kid (1997)                               5.0
Someone Else's America (1995)                 5.0
Name: rating, dtype: float64
```

```
title
Star Wars (1977)                 584
Contact (1997)                   509
Fargo (1996)                     508
Return of the Jedi (1983)        507
Liar Liar (1997)                 485
Name: rating, dtype: int64
```

Out[159]:

title	rating	num of ratings
'Til There Was You (1997)	2.333333	9
1-900 (1994)	2.600000	5
101 Dalmatians (1996)	2.908257	109
12 Angry Men (1957)	4.344000	125
187 (1997)	3.024390	41

```
<matplotlib.axes._subplots.AxesSubplot at 0x1258f8780>
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x125d12908>
```