Experiment: 1

Aim: To perform linear regression on Housing Data

Theory:

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

A linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables.

About the dataset:

Based on the different features of the Boston dataset we are predicting the value of house.

Code:

Introduction to regression

[2]:
```
# Boston housing data
import pandas as pd
import numpy as np
boston=pd.read_csv("boston.csv")
boston.head()
```

[2]:

	CRIM	ZN	INDUS	CHAS	NX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

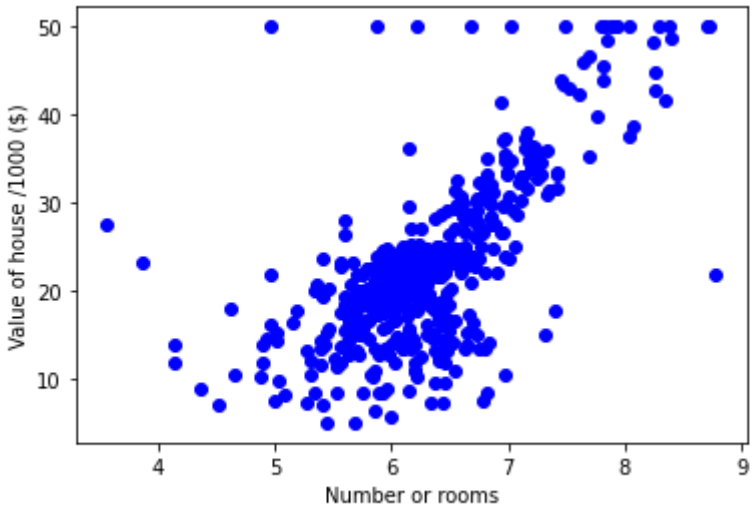
[3]:
```
# creating feature and target arrays

X=boston.drop("MEDV",axis=1).values  # values attributes return
y=boston["MEDV"].values               # numpy arrays
```

[4]:
```
# predicting house value from a single feature
X_rooms=X[:,5]
type(X_rooms), type(y)
```

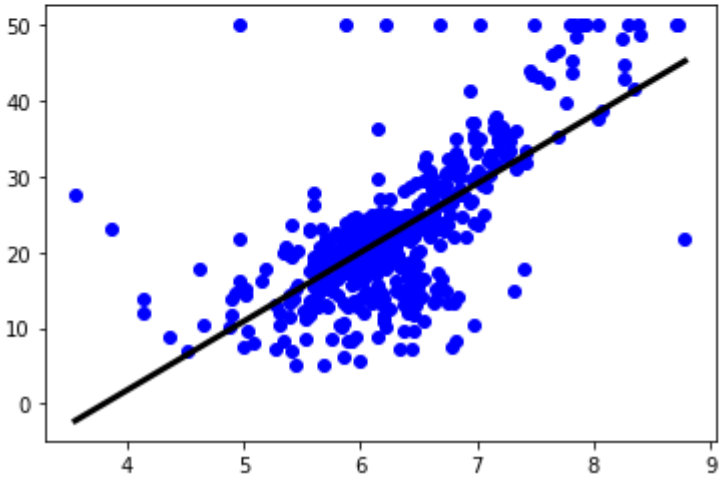
[4]:
```
(numpy.ndarray, numpy.ndarray)
```

[5]:
```
y=y.reshape(-1,1)
X_rooms=X_rooms.reshape(-1,1)
```

[7]:
```
# plotting house value vs number of rooms
import matplotlib.pyplot as plt
plt.scatter(X_rooms,y, color= "blue")
plt.ylabel("Value of house /1000 ($)")
plt.xlabel("Number or rooms")
plt.show()
```



```
# fitting a regression model

import numpy as np
from sklearn.linear_model import LinearRegression

reg= LinearRegression()
reg.fit(X_rooms, y)
prediction_space= np.linspace(min(X_rooms),max(X_rooms)).reshape(-1,1)

plt.scatter(X_rooms, y , color= "blue")
plt.plot(prediction_space, reg.predict(prediction_space),
        color="black", linewidth= 3)
plt.show()
```




```
# import mean_squared_error for checking the model performance

from sklearn.metrics import mean_squared_error
y_pred = reg.predict(X_rooms)
mse = mean_squared_error(y,y_pred)
rmse= np.sqrt(mse)
print("Root Mean Squared Error {}".format(rmse))
```

```
Root Mean Squared Error 6.603071389222561
```

```
# Linear regression on all features
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X_train, X_test , y_train, y_test= train_test_split(X,y, test_size=0.3, random_state=42)
reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
y_pred= reg_all.predict(X_test)
reg_all.score(X_test, y_test) # R^2-score for checking the model performance
```

```
0.7112260057484894
```

```
# import mean_squared_error for checking the model performance

from sklearn.metrics import mean_squared_error
rmse= np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error {}".format(rmse))
```

```
Root Mean Squared Error 4.638689926172851
```

```
# cross validation in scikit-learn

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
cv_results= cross_val_score(reg,X,y, cv=5)
# it returns the R^2 values of 5 folds
print(cv_results)

# mean of the results
print(np.mean(cv_results))
```

```
[ 0.63919994  0.71386698  0.58702344  0.07923081 -0.25294154]
0.35327592439588307
```

Result:

The performance of the linear regression model on all the features of the Boston data is 0.35 which is Poor.

Experiment: 2

Aim: To perform logistic regression on the gender classification dataset.

Theory:

Logistic regression is a process of modelling the probability of a discrete outcome given an input variable.

It is a classification model, which is very easy to realize and achieves very good performance with linearly separable classes. It is an extensively employed algorithm for classification in industry.

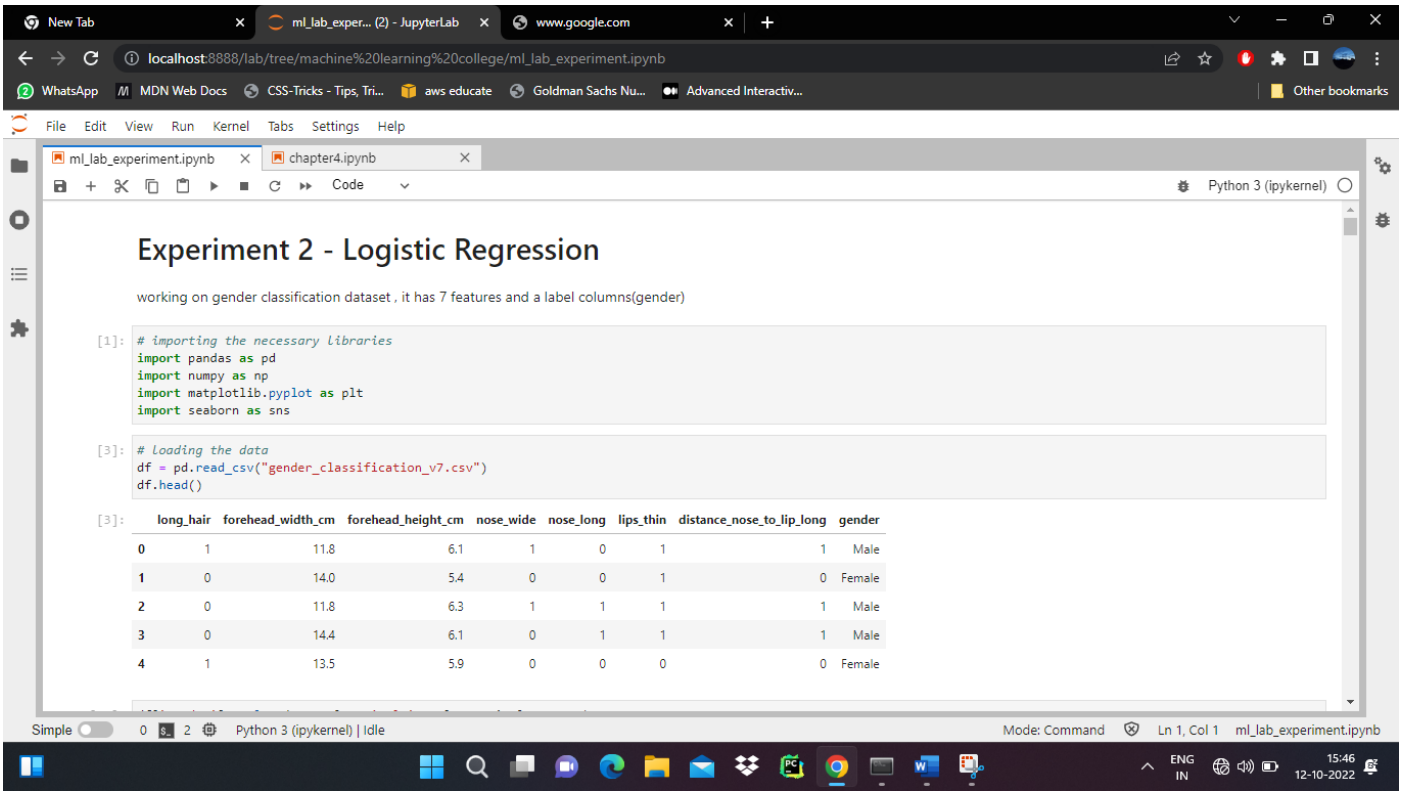
The most common model is a binary outcome; something that can take two values such as true/false, yes/no, and so on. Multinomial logistic regression can model scenarios where there are more than two possible discrete outcomes.

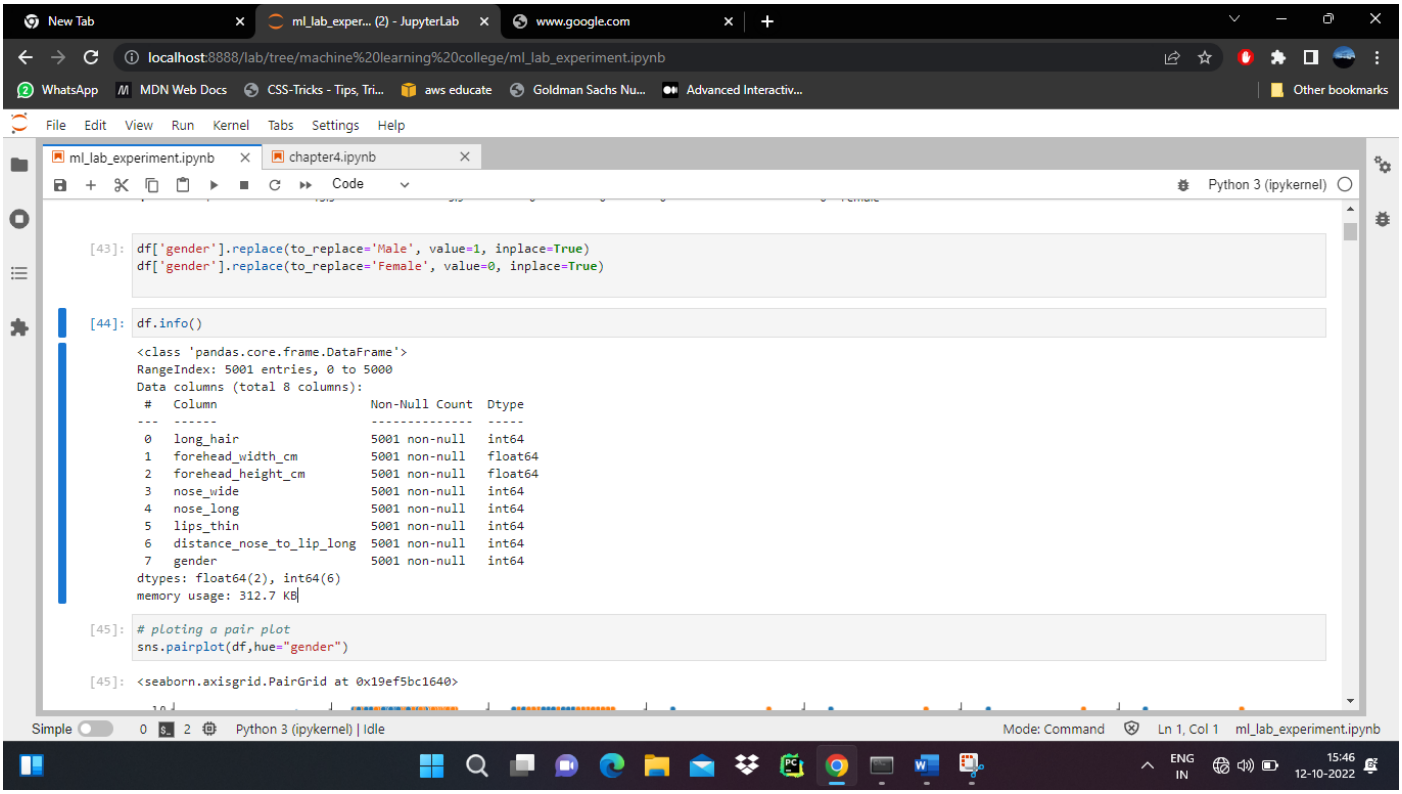
About the dataset:

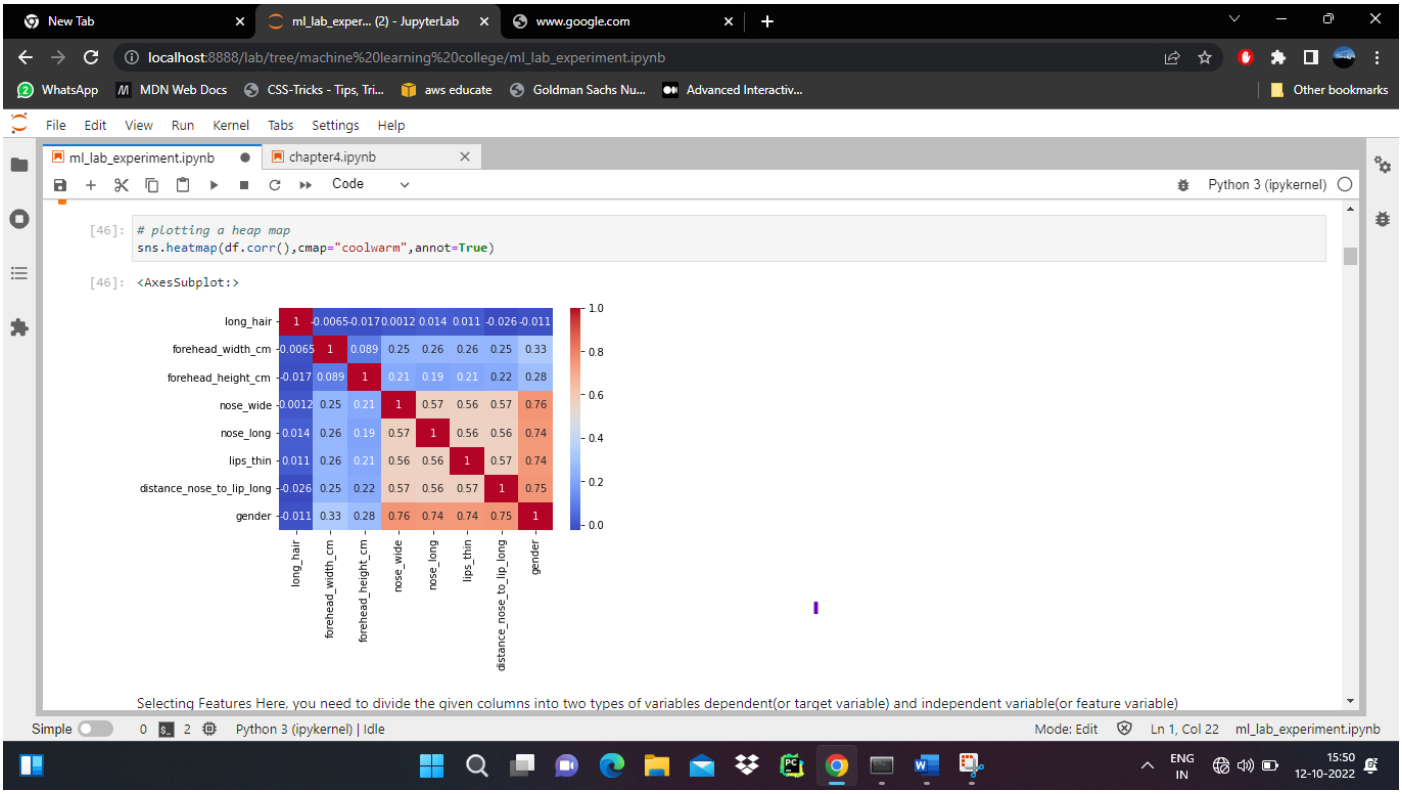
This dataset contains 7 features and a label column.

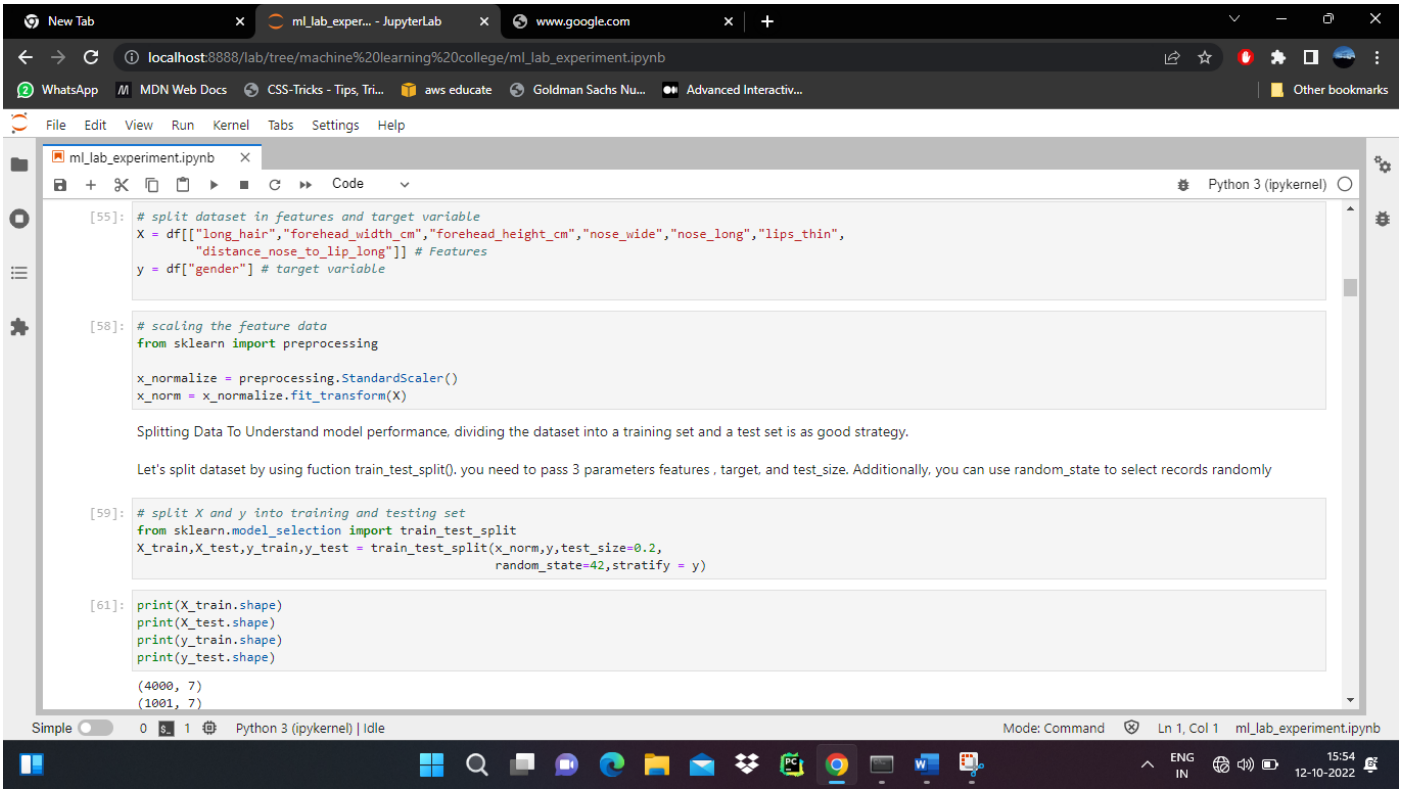
long*hair* - This column contains 0's and 1's where 1 is "long hair" and 0 is "not long hair".
forehead*width*cm - This column is in CM's. This is the width of the forehead.
forehead*height*cm - This is the height of the forehead and it's in Cm's.
nose*wide* - This column contains 0's and 1's where 1 is "wide nose" and 0 is "not wide nose".
nose*long* - This column contains 0's and 1's where 1 is "Long nose" and 0 is "not long nose".
lips*thin* - This column contains 0's and 1's where 1 represents the "thin lips" while 0 is "Not thin lips".
distance*nose*to*lip*long - This column contains 0's and 1's where 1 represents the "long distance between nose and lips" while 0 is "short distance between nose and lips".
gender - This is either "Male" or "Female".

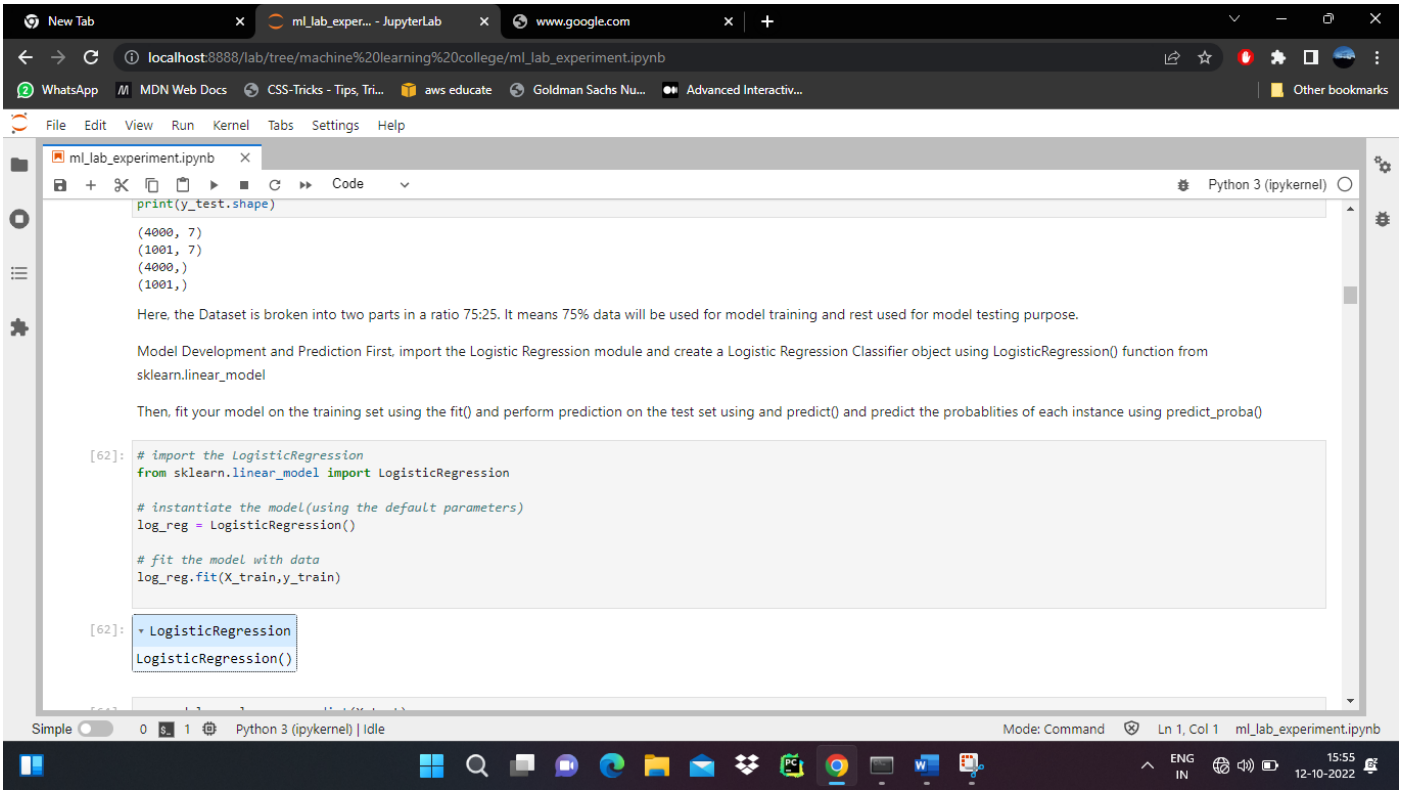
Code:




New Tab
ml_lab_exper... (2) - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
chapter4.ipynb
Code
Python 3 (ipykernel)
[43]: df['gender'].replace(to_replace='Male', value=1, inplace=True)
df['gender'].replace(to_replace='Female', value=0, inplace=True)
[44]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
Column Non-Null Count Dtype
0 long_hair 5001 non-null int64
1 forehead_width_cm 5001 non-null float64
2 forehead_height_cm 5001 non-null float64
3 nose_wide 5001 non-null int64
4 nose_long 5001 non-null int64
5 lips_thin 5001 non-null int64
6 distance_nose_to_lip_long 5001 non-null int64
7 gender 5001 non-null int64
dtypes: float64(2), int64(6)
memory usage: 312.7 KB
[45]: # ploting a pair plot
sns.pairplot(df,hue="gender")
[45]: <seaborn.axisgrid.PairGrid at 0x19ef5bc1640>
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
ml_lab_experiment.ipynb


New Tab
ml_lab_exper... (2) - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
chapter4.ipynb
Code
Python 3 (ipykernel)
[46]: # plotting a heap map
sns.heatmap(df.corr(),cmap="coolwarm",annot=True)
[46]: <AxesSubplot:>
long_hair 1 -0.0065 -0.017 0.0012 0.014 0.011 -0.026 -0.011
forehead_width_cm -0.0065 1 0.089 0.25 0.26 0.26 0.25 0.33
forehead_height_cm -0.017 0.089 1 0.21 0.19 0.21 0.22 0.28
nose_wide 0.0012 0.25 0.21 1 0.57 0.56 0.57 0.76
nose_long 0.014 0.26 0.19 0.57 1 0.56 0.56 0.74
lips_thin 0.011 0.26 0.21 0.56 0.56 1 0.57 0.74
distance_nose_to_lip_long -0.026 0.25 0.22 0.57 0.56 0.57 1 0.75
gender -0.011 0.33 0.28 0.76 0.74 0.74 0.75 1
1.0
0.8
0.6
0.4
0.2
0.0
long_hair
forehead_width_cm
forehead_height_cm
nose_wide
nose_long
lips_thin
distance_nose_to_lip_long
gender
Selecting Features Here, you need to divide the given columns into two types of variables dependent(or target variable) and independent variable(or feature variable)
Simple
Python 3 (ipykernel) | Idle
Mode: Edit
Ln 1, Col 22
ml_lab_experiment.ipynb


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Code
Python 3 (ipykernel)
[55]:
# split dataset in features and target variable
X = df[["long_hair","forehead_width_cm","forehead_height_cm","nose_wide","nose_long","lips_thin",
"distance_nose_to_lip_long"]] # Features
y = df["gender"] # target variable
[58]:
# scaling the feature data
from sklearn import preprocessing
x_normalize = preprocessing.StandardScaler()
x_norm = x_normalize.fit_transform(X)
Splitting Data To Understand model performance, dividing the dataset into a training set and a test set is as good strategy.
Let's split dataset by using fuction train_test_split(). you need to pass 3 parameters features , target, and test_size. Additionally, you can use random_state to select records randomly
[59]:
# split X and y into training and testing set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x_norm,y,test_size=0.2,
random_state=42,stratify = y)
[61]:
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
(4000, 7)
(1001, 7)
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
ml_lab_experiment.ipynb
ENG
IN
15:54
12-10-2022


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Code
Python 3 (ipykernel)
print(y_test.shape)
(4000, 7)
(1001, 7)
(4000,)
(1001,)
Here, the Dataset is broken into two parts in a ratio 75:25. It means 75% data will be used for model training and rest used for model testing purpose.
Model Development and Prediction First, import the Logistic Regression module and create a Logistic Regression Classifier object using LogisticRegression() function from sklearn.linear_model
Then, fit your model on the training set using the fit() and perform prediction on the test set using and predict() and predict the probablities of each instance using predict_proba()
[62]:
# import the LogisticRegression
from sklearn.linear_model import LogisticRegression
# instantiate the model(using the default parameters)
log_reg = LogisticRegression()
# fit the model with data
log_reg.fit(X_train,y_train)
[62]:
LogisticRegression
LogisticRegression()
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
ml_lab_experiment.ipynb
ENG
IN
15:55
12-10-2022

```
[64]: y_pred_log = log_reg.predict(X_test)

[65]: from sklearn.metrics import classification_report
      _report1 =classification_report( y_test , y_pred_log)
      print(_report1)
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       501
           1       0.96      0.96      0.96       500

    accuracy                           0.96      1001
   macro avg       0.96      0.96      0.96      1001
weighted avg       0.96      0.96      0.96      1001
```

Model Evaluation using the Confusion Matrix A confusion matrix is a table that is used the performance of a classification model.you can also visualize the performance of an algorithm. The fundamental of a confusion matrix is the number of correct and incorrect predictions are summed up class wise

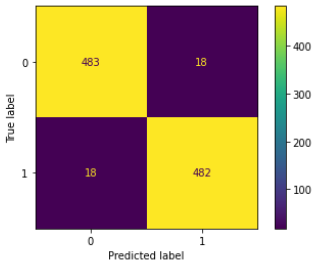
```
[66]: # import the metrics class
      from sklearn.metrics import confusion_matrix
      cnf_matrix = confusion_matrix(y_test,y_pred_log)
      cnf_matrix
```

```
[66]: array([[483,  18],
           [ 18, 482]], dtype=int64)
```

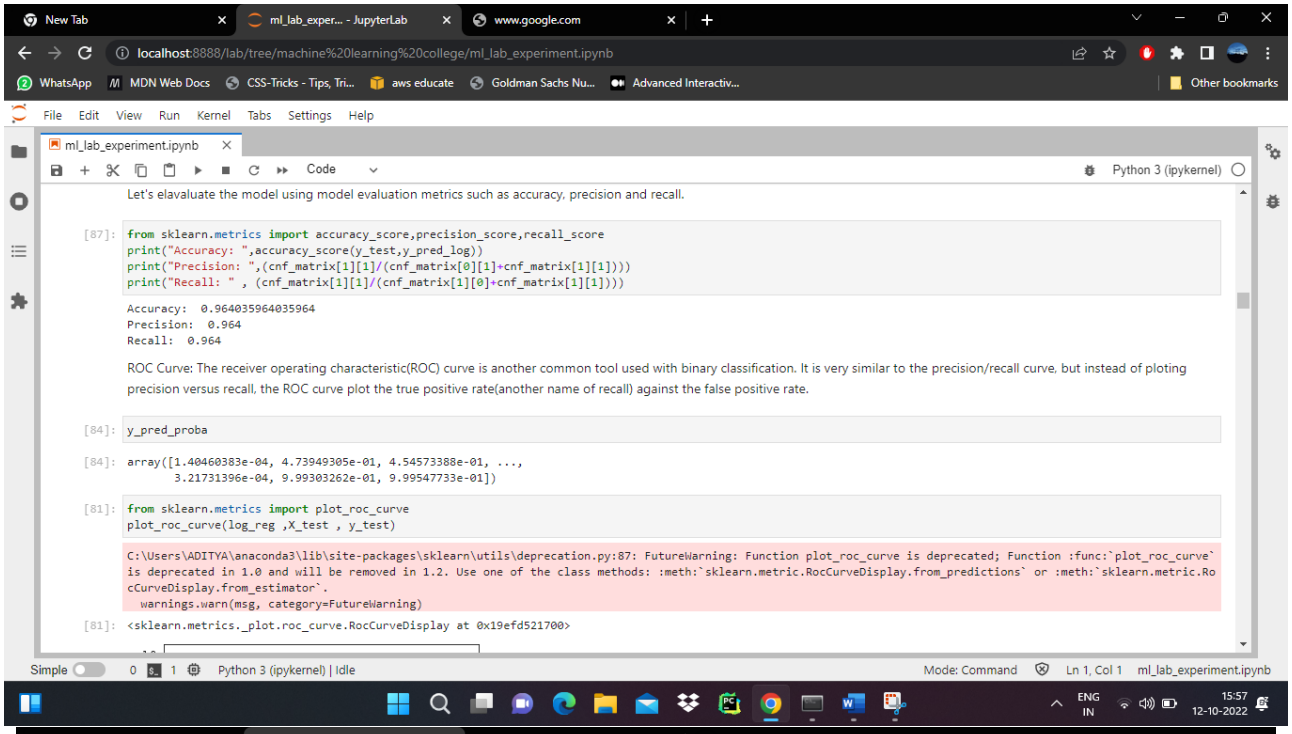
```
[72]: # plotting the confusion matrix
      from sklearn.metrics import plot_confusion_matrix
      plot_confusion_matrix(log_reg , X_test, y_test)
```

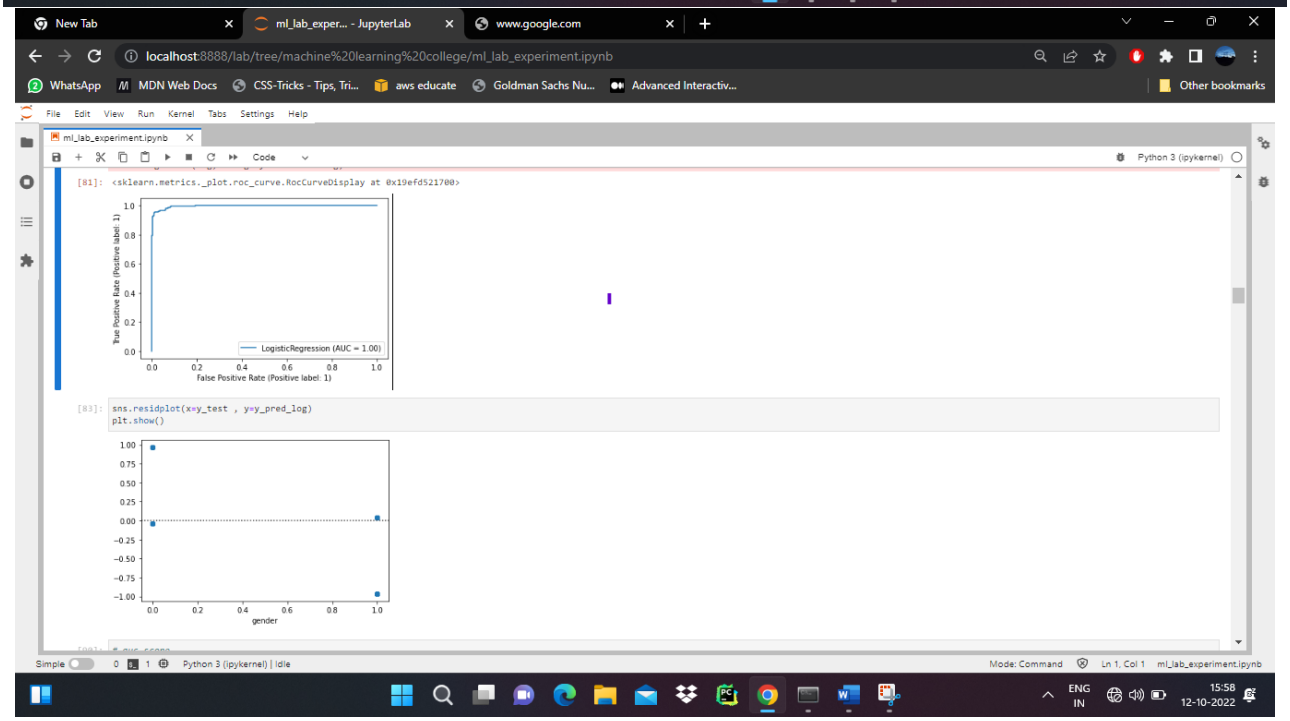
```
C:\Users\ADITYA\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

```
[72]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19efd90f0a0>
```



Here, you can see the confusion matrix in the form of array object. The dimension of the matrix is 2*2 because this model is binary classification. you have two classes 0 and 1.





Result:

The given dataset predicts gender of a person based on the input parameters.
The accuracy, precision, and recall are:

Accuracy: 0.964035964035964
Precision: 0.964
Recall: 0.964

Conclusion:

- The accuracy of the model is 96.40% for the given dataset.

Experiment: 3

Aim: To perform decision tree algorithm on the gender classification dataset.

Theory:

Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**

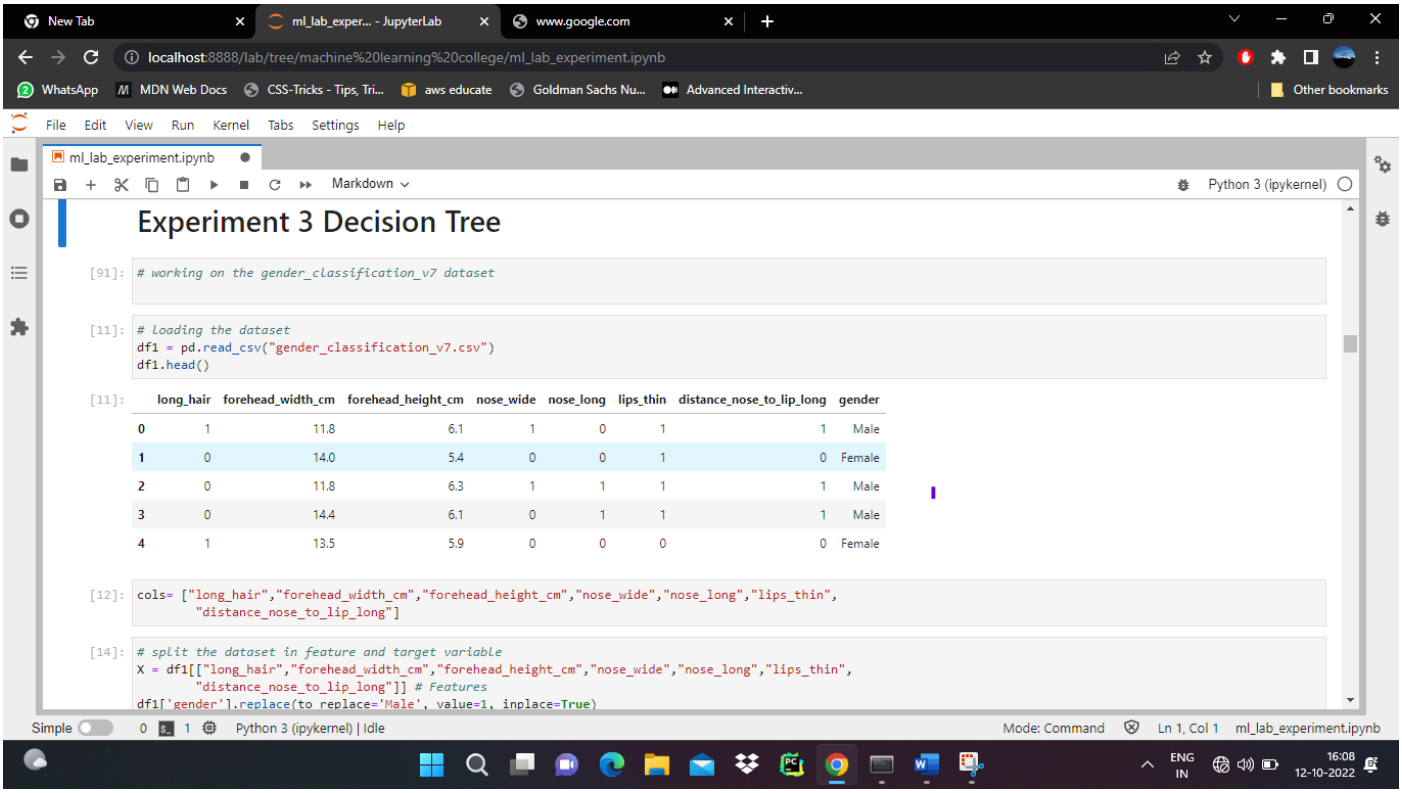
In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

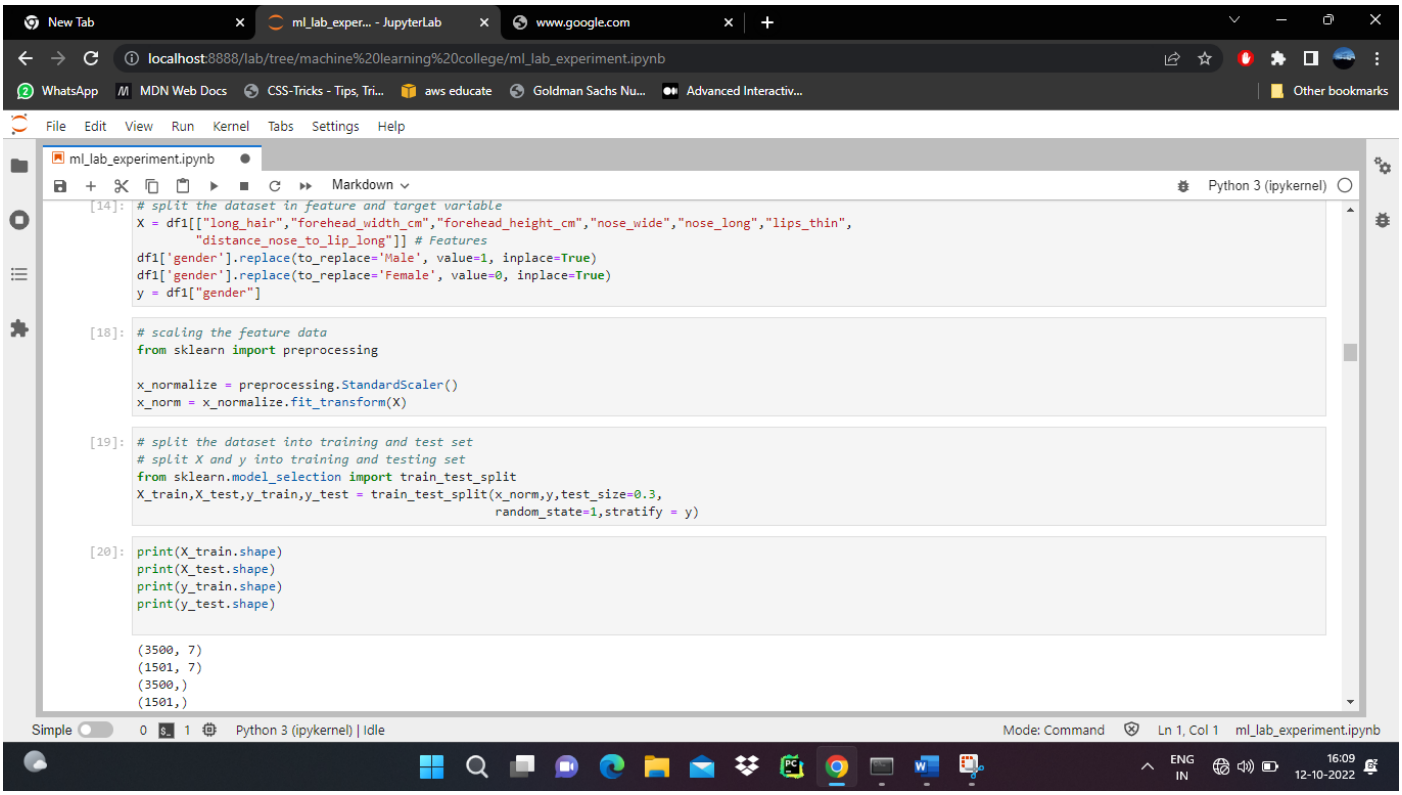
About the dataset:

This dataset contains 7 features and a label column.

long*hair* - This column contains 0's and 1's where 1 is "long hair" and 0 is "not long hair".
foreheadwidth*cm* - This column is in CM's. This is the width of the forehead.
foreheadheight*cm* - This is the height of the forehead and it's in Cm's.
nosewide - This column contains 0's and 1's where 1 is "wide nose" and 0 is "not wide nose".
nose*long* - This column contains 0's and 1's where 1 is "Long nose" and 0 is "not long nose".
lipsthin - This column contains 0's and 1's where 1 represents the "thin lips" while 0 is "Not thin lips".
distance*nose*to*lip*long - This column contains 0's and 1's where 1 represents the "long distance between nose and lips" while 0 is "short distance between nose and lips".
gender - This is either "Male" or "Female".

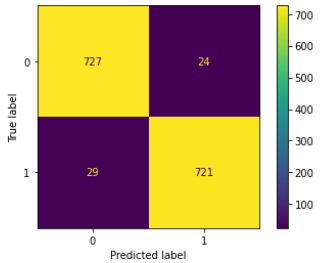
Code:


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Markdown
Python 3 (ipykernel)
Experiment 3 Decision Tree
[91]: # working on the gender_classification_v7 dataset
[11]: # loading the dataset
df1 = pd.read_csv("gender_classification_v7.csv")
df1.head()
[11]:
long_hair forehead_width_cm forehead_height_cm nose_wide nose_long lips_thin distance_nose_to_lip_long gender
0 1 11.8 6.1 1 0 1 1 Male
1 0 14.0 5.4 0 0 1 0 Female
2 0 11.8 6.3 1 1 1 1 Male
3 0 14.4 6.1 0 1 1 1 Male
4 1 13.5 5.9 0 0 0 0 Female
[12]: cols= ["long_hair","forehead_width_cm","forehead_height_cm","nose_wide","nose_long","lips_thin",
"distance_nose_to_lip_long"]
[14]: # split the dataset in feature and target variable
X = df1[["long_hair","forehead_width_cm","forehead_height_cm","nose_wide","nose_long","lips_thin",
"distance_nose_to_lip_long"]] # Features
df1['gender'].replace(to_replace='Male', value=1, inplace=True)
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
ml_lab_experiment.ipynb
ENG IN
16:08
12-10-2022


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Markdown
Python 3 (ipykernel)
[14]: # split the dataset in feature and target variable
X = df1[["long_hair","forehead_width_cm","forehead_height_cm","nose_wide","nose_long","lips_thin",
"distance_nose_to_lip_long"]] # Features
df1['gender'].replace(to_replace='Male', value=1, inplace=True)
df1['gender'].replace(to_replace='Female', value=0, inplace=True)
y = df1["gender"]
[18]: # scaling the feature data
from sklearn import preprocessing
x_normalize = preprocessing.StandardScaler()
x_norm = x_normalize.fit_transform(X)
[19]: # split the dataset into training and test set
# split X and y into training and testing set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x_norm,y,test_size=0.3,
random_state=1,stratify = y)
[20]: print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
(3500, 7)
(1501, 7)
(3500,)
(1501,)
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
ml_lab_experiment.ipynb
ENG IN
16:09
12-10-2022

```
(1501,)

[21]: from sklearn.tree import DecisionTreeClassifier

      # create the instance of the Decision Tree classifier
      tree_clf = DecisionTreeClassifier()

      # Train the Decision Tree Classifier
      tree_clf.fit(X_train,y_train)

[21]: DecisionTreeClassifier
      DecisionTreeClassifier()

[22]: y_pred_tree = tree_clf.predict(X_test)

[23]: # model Accucary , how often is the classifier is correct
      from sklearn.metrics import accuracy_score
      print("Accuracy: " , accuracy_score(y_test,y_pred_tree))

      Accuracy:  0.9646902065289806

[24]: # classification report
      from sklearn.metrics import classification_report
      _report3 = classification_report( y_test , y_pred_tree)
      print(_report3)

                   precision    recall  f1-score   support
```

```
                   0       0.96      0.97      0.96       751
                   1       0.97      0.96      0.96       750

            accuracy                           0.96      1501
           macro avg       0.96      0.96      0.96      1501
        weighted avg       0.96      0.96      0.96      1501


[42]: # confusion matrix
      from sklearn.metrics import plot_confusion_matrix
      plot_confusion_matrix(tree_clf , X_test, y_test)

[42]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x247e5667670>
```



New Tab | ml_lab_exper... - JupyterLab | www.google.com

localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb

WhatsApp | MDN Web Docs | CSS-Tricks - Tips, Tri... | aws educate | Goldman Sachs Nu... | Advanced Interactiv... | Other bookmarks

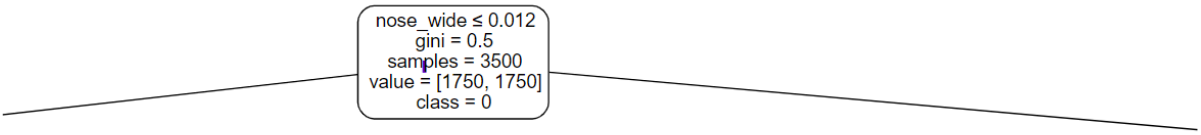
File Edit View Run Kernel Tabs Settings Help

ml_lab_experiment.ipynb

Code | Python 3 (ipykernel)

[36]:
```
import os
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
from graphviz import Source

dot_data = StringIO()
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "decision_trees"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)
export_graphviz(tree_clf,
                out_file=os.path.join(IMAGES_PATH, "gender_tree.dot"),
                filled=True,rounded = True,
                special_characters=True,feature_names=cols,class_names=["0","1"])
Source.from_file(os.path.join(IMAGES_PATH, "gender_tree.dot"))
```

[36]:



Simple | 0 | 1 | Python 3 (ipykernel) | Idle | Mode: Command | Ln 1, Col 1 | ml_lab_experiment.ipynb

ENG IN 16:12 12-10-2022

New Tab | ml_lab_exper... - JupyterLab | www.google.com

localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb

WhatsApp | MDN Web Docs | CSS-Tricks - Tips, Tri... | aws educate | Goldman Sachs Nu... | Advanced Interactiv... | Other bookmarks

File Edit View Run Kernel Tabs Settings Help

ml_lab_experiment.ipynb

Code | Python 3 (ipykernel)

[37]:
```
# creating a new decison with criterion as entropy an max_depth = 3
tree_clf = None
# create decision tree classifier object
tree_clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)
# train the DTC
tree_clf.fit(X_train,y_train)
```

[37]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

[39]:
```
# predict the response for test dataset
y_pred = tree_clf.predict(X_test)

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

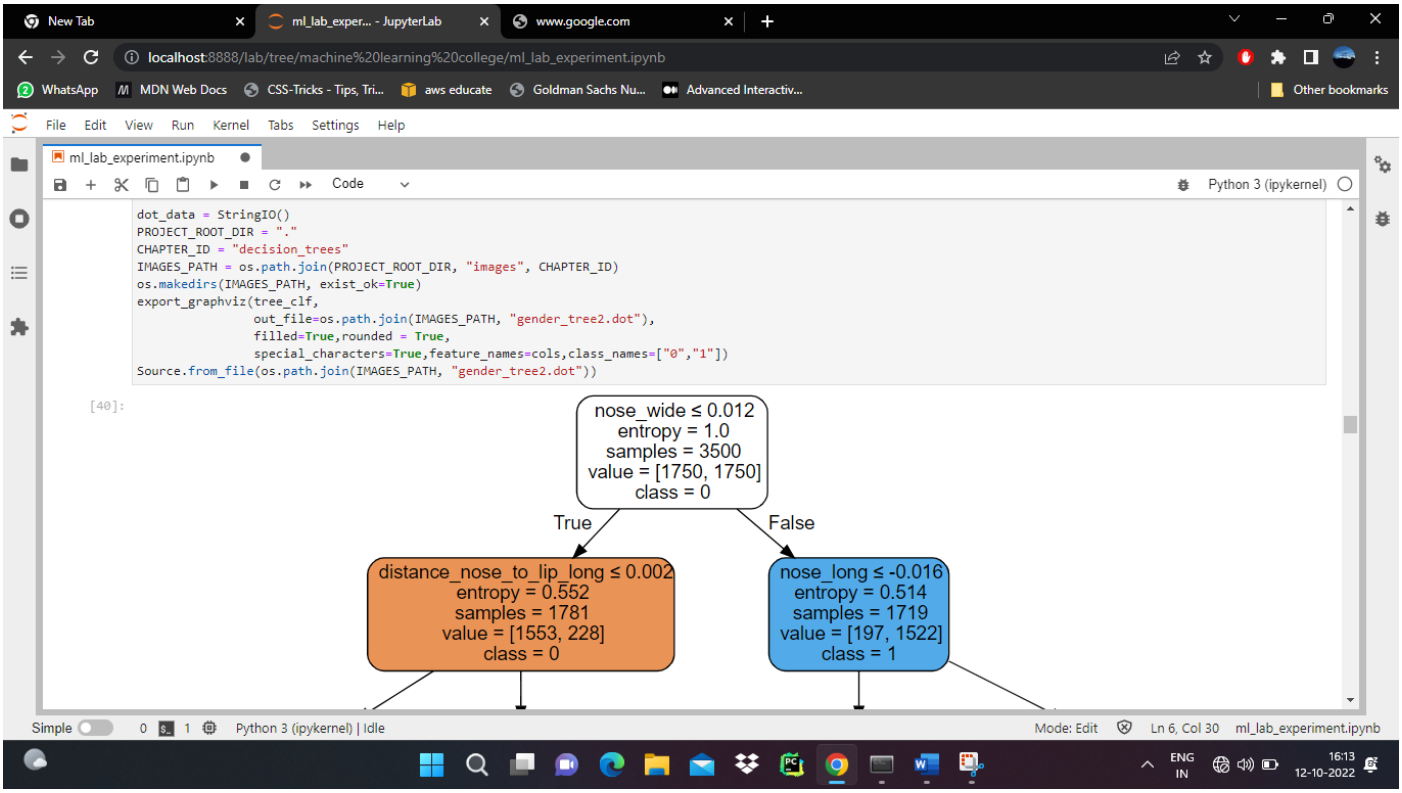
```
Accuracy:  0.9626915389740173
```

Visualizing the decision tree

[40]:
```
import os
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from six import StringIO
from IPython.display import Image
import pydotplus
from graphviz import Source

dot_data = StringIO()
```

Simple | 0 | 1 | Python 3 (ipykernel) | Idle | Mode: Edit | Ln 6, Col 30 | ml_lab_experiment.ipynb

ENG IN 16:13 12-10-2022



Result:

The given dataset predicts the gender of a person based on the input parameters.
The accuracy, precision, and recall are :

Accuracy: 0.9646902065289806

Conclusion:

- The nose_width, lips_thin, forehead_width, and distance_nose_to_lip_long are the important features for the prediction of stroke in the given database.

- The accuracy of the model is 96.46% for the given dataset.

Classification Algorithm	Accuracy
• Logistic Regression	96.40
• Decision Tree	96.46

Experiment: 4

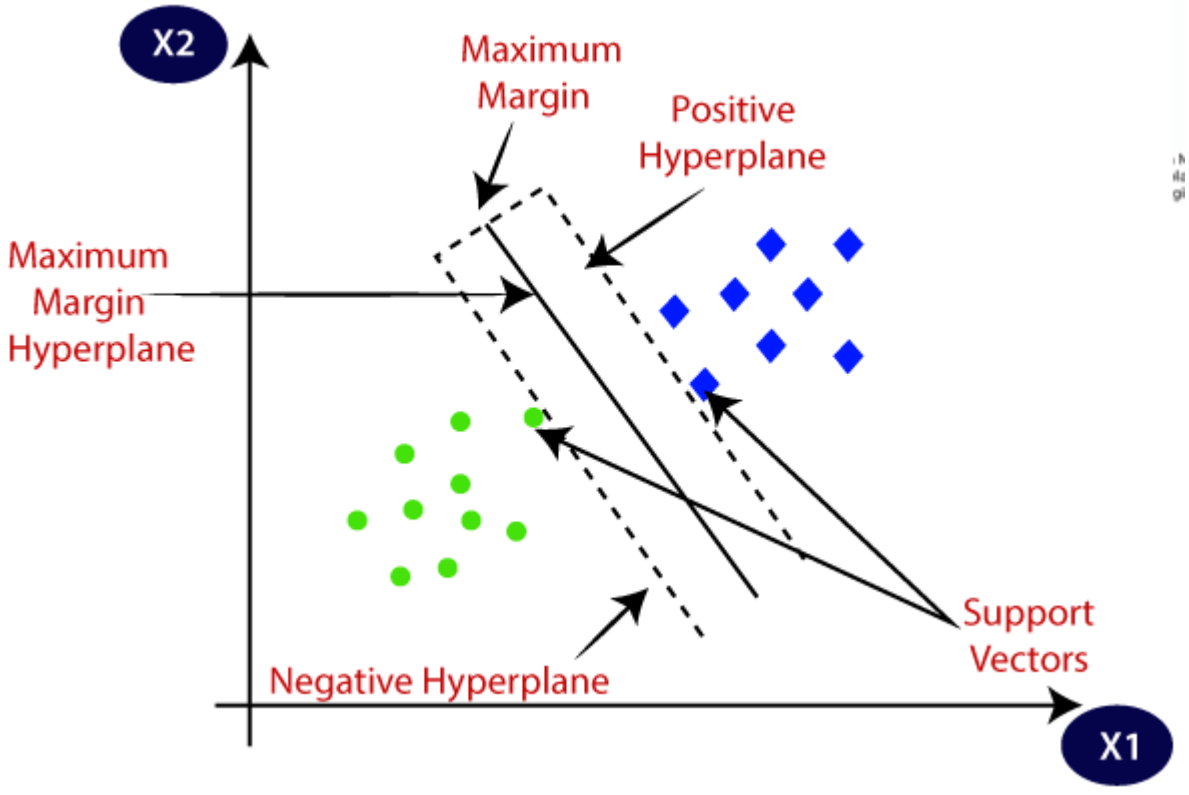
Aim: To perform support vector machine algorithm on the gender classification dataset.

Theory:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

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



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about the different features of cats and dogs, and then we test it with this strange creature. So as the support vector creates a decision boundary between these two data (cat and dog) and chooses

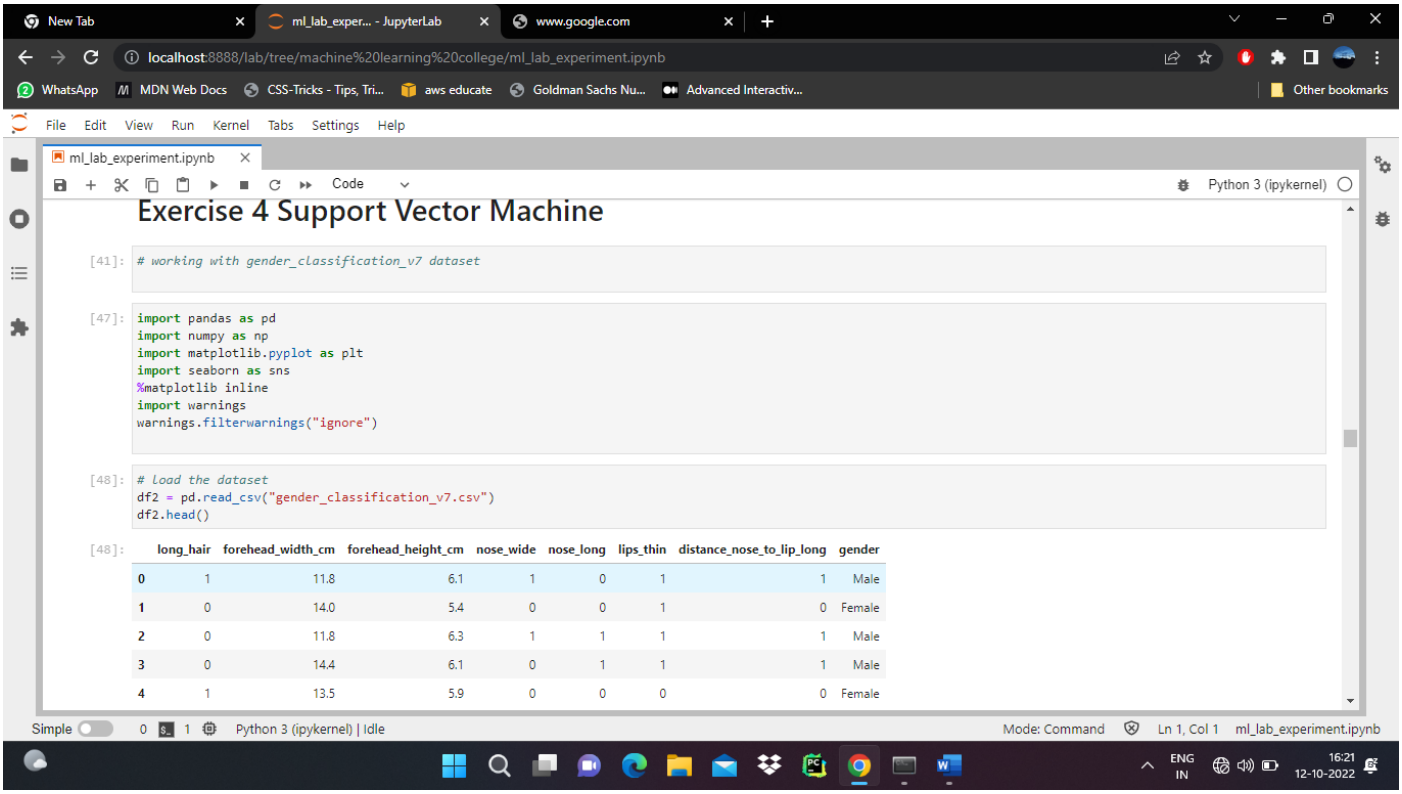
extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat.

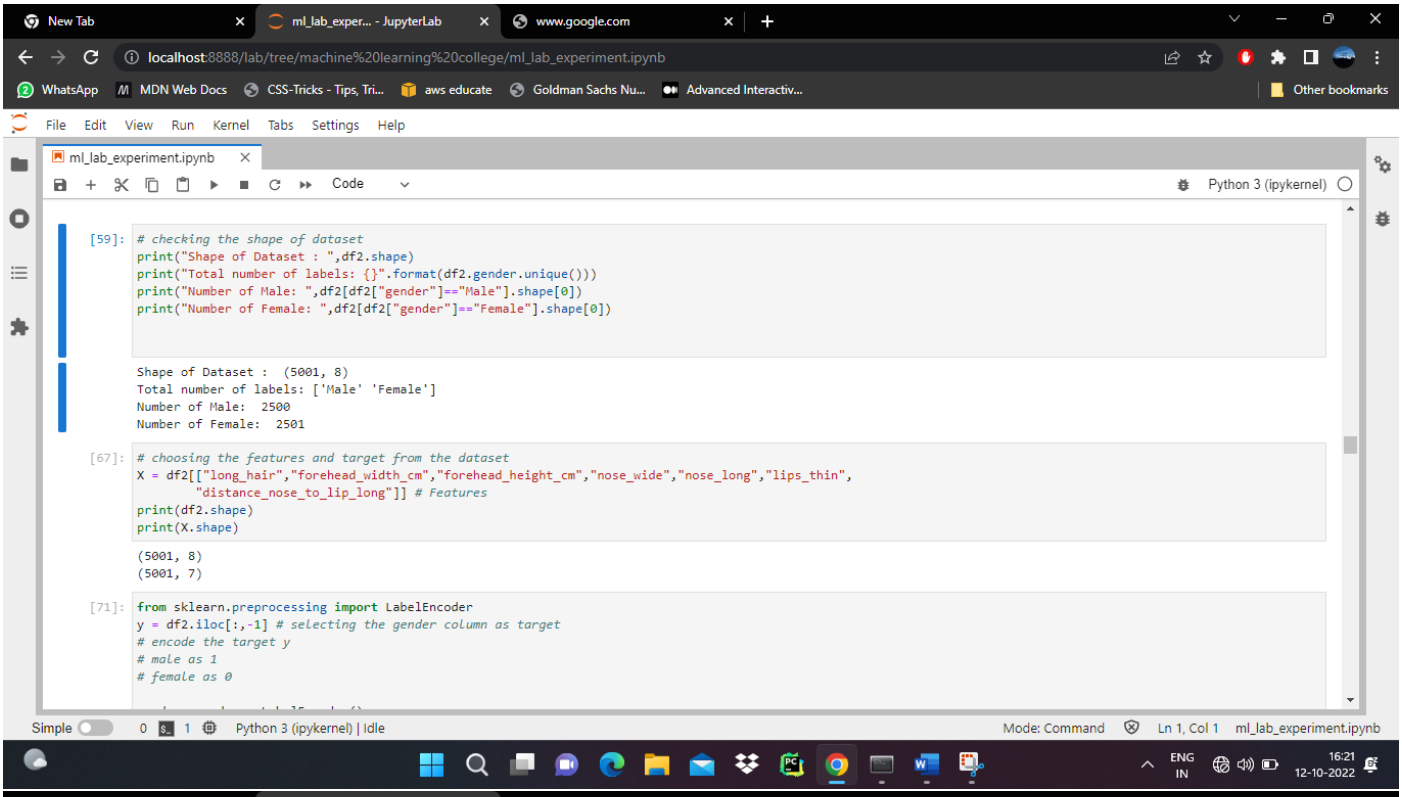
About the dataset:

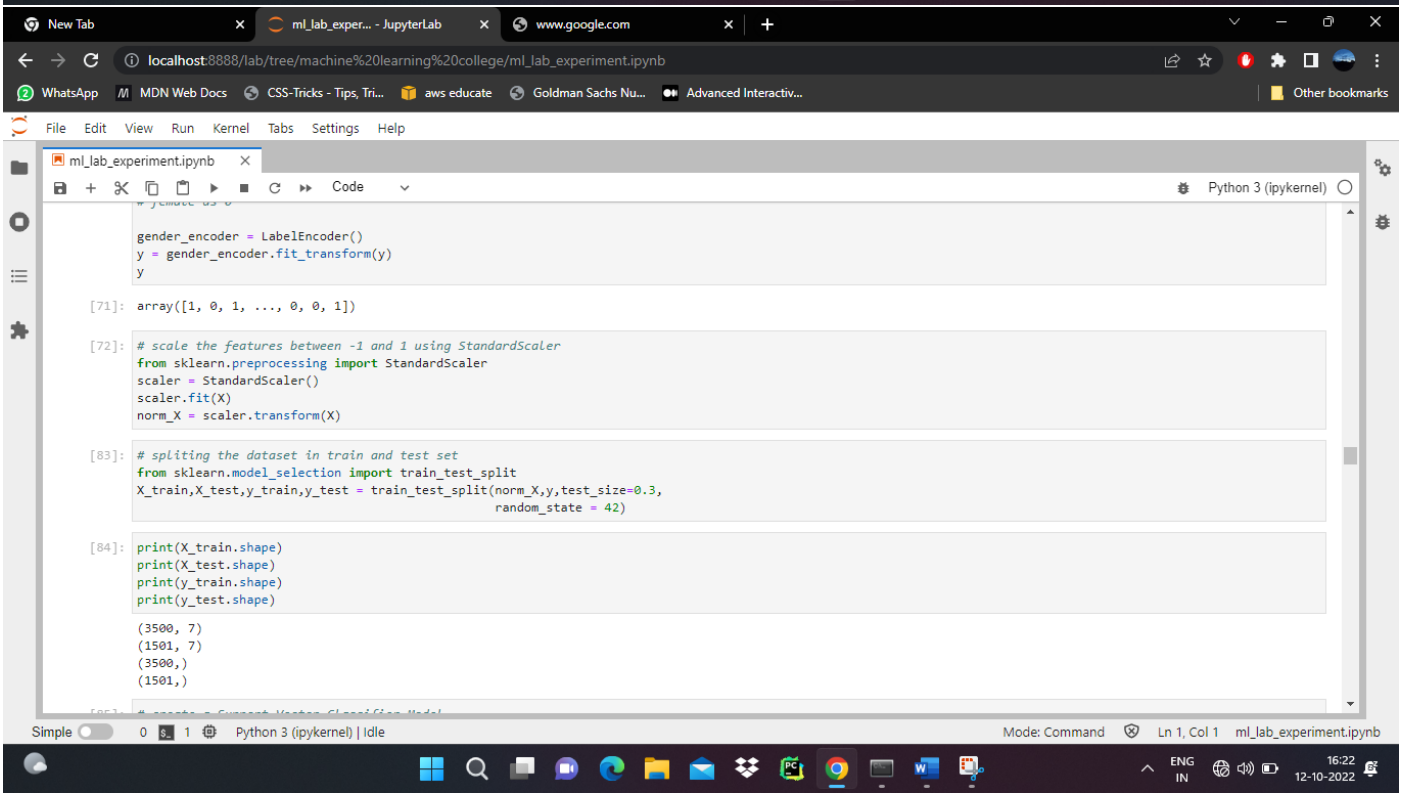
This dataset contains 7 features and a label column.

long*hair* - This column contains 0's and 1's where 1 is "long hair" and 0 is "not long hair".
foreheadwidth*cm* - This column is in CM's. This is the width of the forehead.
foreheadheight*cm* - This is the height of the forehead and it's in Cm's.
nosewide - This column contains 0's and 1's where 1 is "wide nose" and 0 is "not wide nose".
nose*long* - This column contains 0's and 1's where 1 is "Long nose" and 0 is "not long nose".
lipsthin - This column contains 0's and 1's where 1 represents the "thin lips" while 0 is "Not thin lips".
distance*nose*to*lip*long - This column contains 0's and 1's where 1 represents the "long distance between nose and lips" while 0 is "short distance between nose and lips".
gender - This is either "Male" or "Female".

Code:




New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Code
Python 3 (ipykernel)
[59]:
# checking the shape of dataset
print("Shape of Dataset : ",df2.shape)
print("Total number of labels: {}".format(df2.gender.unique()))
print("Number of Male: ",df2[df2["gender"]=="Male"].shape[0])
print("Number of Female: ",df2[df2["gender"]=="Female"].shape[0])
Shape of Dataset : (5001, 8)
Total number of labels: ['Male' 'Female']
Number of Male: 2500
Number of Female: 2501
[67]:
# choosing the features and target from the dataset
X = df2[["long_hair","forehead_width_cm","forehead_height_cm","nose_wide","nose_long","lips_thin",
"distance_nose_to_lip_long"]] # Features
print(df2.shape)
print(X.shape)
(5001, 8)
(5001, 7)
[71]:
from sklearn.preprocessing import LabelEncoder
y = df2.iloc[:,-1] # selecting the gender column as target
# encode the target y
# male as 1
# female as 0
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
ml_lab_experiment.ipynb
ENG IN
16:21
12-10-2022


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Code
Python 3 (ipykernel)
gender_encoder = LabelEncoder()
y = gender_encoder.fit_transform(y)
y
[71]:
array([1, 0, 1, ..., 0, 0, 1])
[72]:
# scale the features between -1 and 1 using StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
norm_X = scaler.transform(X)
[83]:
# spliting the dataset in train and test set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(norm_X,y,test_size=0.3,
random_state = 42)
[84]:
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
(3500, 7)
(1501, 7)
(3500,)
(1501,)
Simple
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
ml_lab_experiment.ipynb
ENG IN
16:22
12-10-2022

New Tab | ml_lab_exper... - JupyterLab | www.google.com

localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb

WhatsApp | MDN Web Docs | CSS-Tricks - Tips, Tri... | aws educate | Goldman Sachs Nu... | Advanced Interactiv... | Other bookmarks

File Edit View Run Kernel Tabs Settings Help

ml_lab_experiment.ipynb

Code | Python 3 (ipykernel)

```
[85]: # create a Support Vector Classifier Model
from sklearn.svm import SVC

# svc model on default parameter
svc = SVC()
# fit the training data to the model
svc.fit(X_train,y_train)
```

```
[85]: SVC()
```

```
[86]: # predict the result on test data
from sklearn.metrics import accuracy_score
y_pred = svc.predict(X_test)
print("Accuracy Score: ",accuracy_score(y_test,y_pred))
```

```
Accuracy Score:  0.966688874083944
```

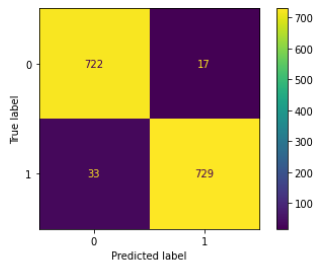
```
[88]: from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       739
           1       0.98      0.96      0.97       762

    accuracy                           0.97      1501
   macro avg       0.97      0.97      0.97      1501
weighted avg       0.97      0.97      0.97      1501
```

```
[96]: from sklearn.metrics import confusion_matrix,plot_confusion_matrix
print(confusion_matrix(y_test,y_pred))
plot_confusion_matrix(svc,X_test,y_test)
```

```
[[722  17]
 [ 33 729]]
```

```
[96]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x22382ce2c10>
```



Simple | 0 | 1 | Python 3 (ipykernel) | Idle | Mode: Command | Ln 1, Col 1 | ml_lab_experiment.ipynb

ENG IN 16:22 12-10-2022

ENG IN 16:23 12-10-2022

Result:

The given dataset predicts the gender of a person based on the input parameters.
The accuracy is:

Accuracy: 0.966688874083944

Conclusion:

- The nose_width, lips_thin, forehead_width, and distance_nose_to_lip_long are the important features for the prediction of stroke in the given database.

- The accuracy of the model is 96.66% for the given dataset.

Classification Algorithm	Accuracy (%)
• Logistic Regression	96.40
• Decision Tree	96.46
• Support Vector Machine (SVM)	96.66

Experiment: 5

Aim: To perform the K-Nearest Neighbours algorithm on the Social Network ADs classification dataset.

Theory:

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on the Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a good suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

About the dataset:

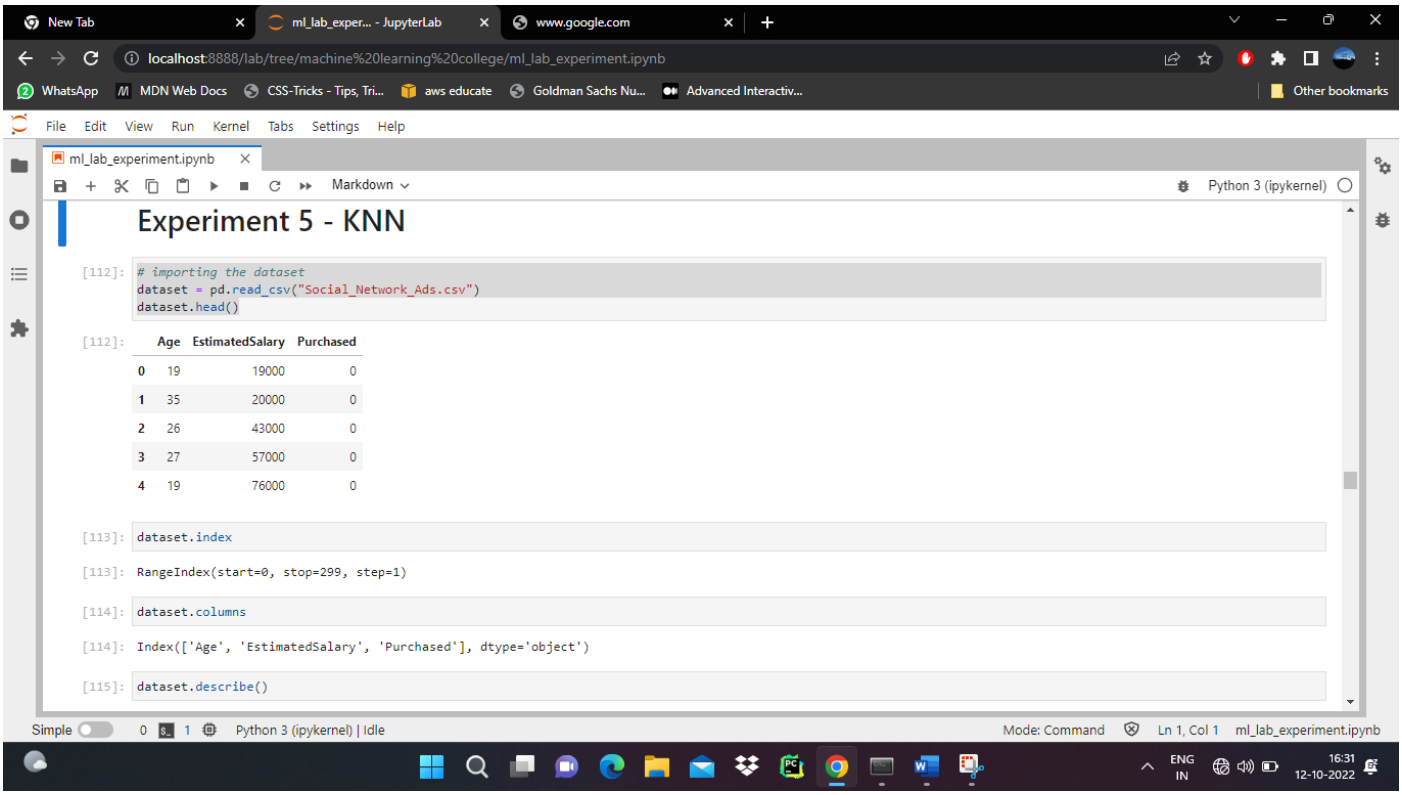
This dataset contains 2 features and a label column.

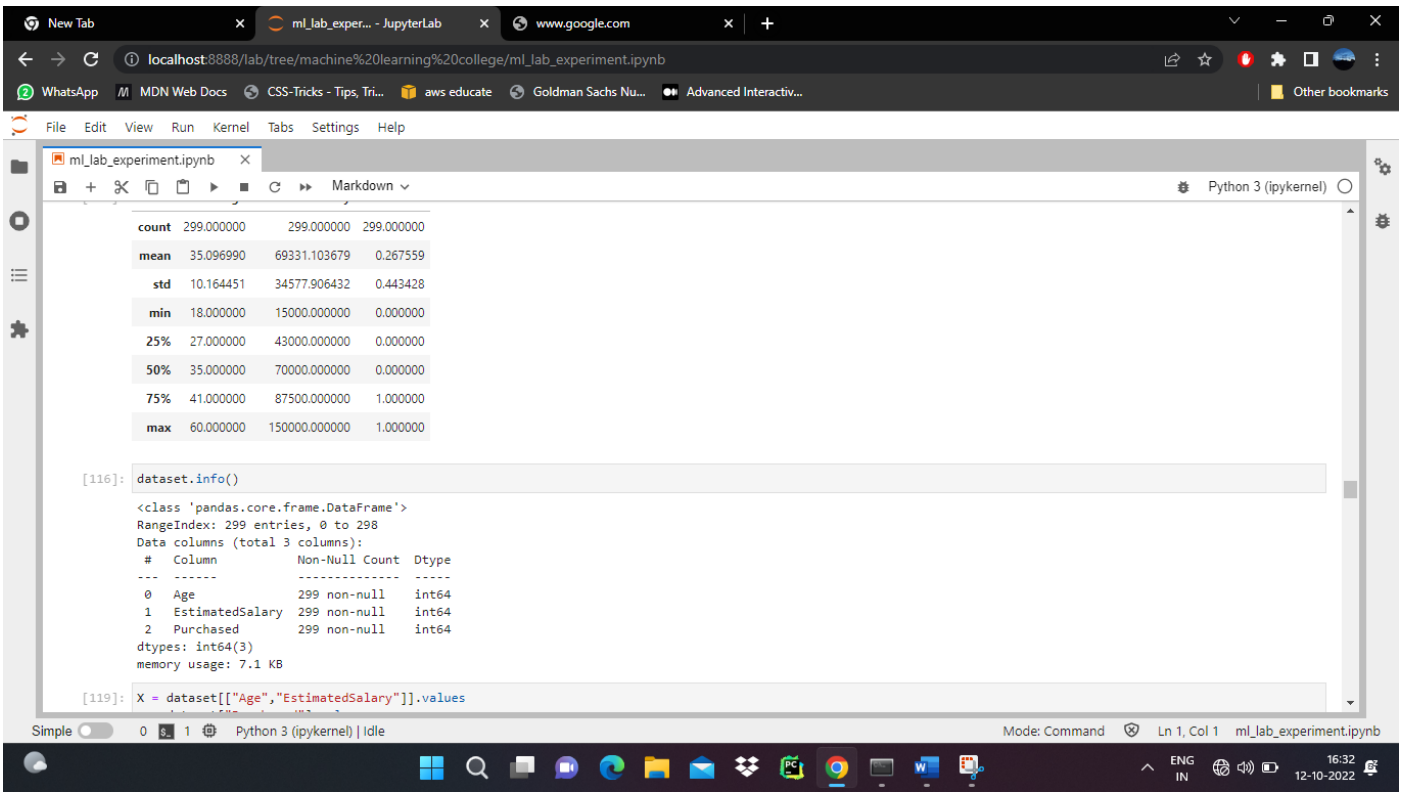
Age-This feature contains the age of the person.

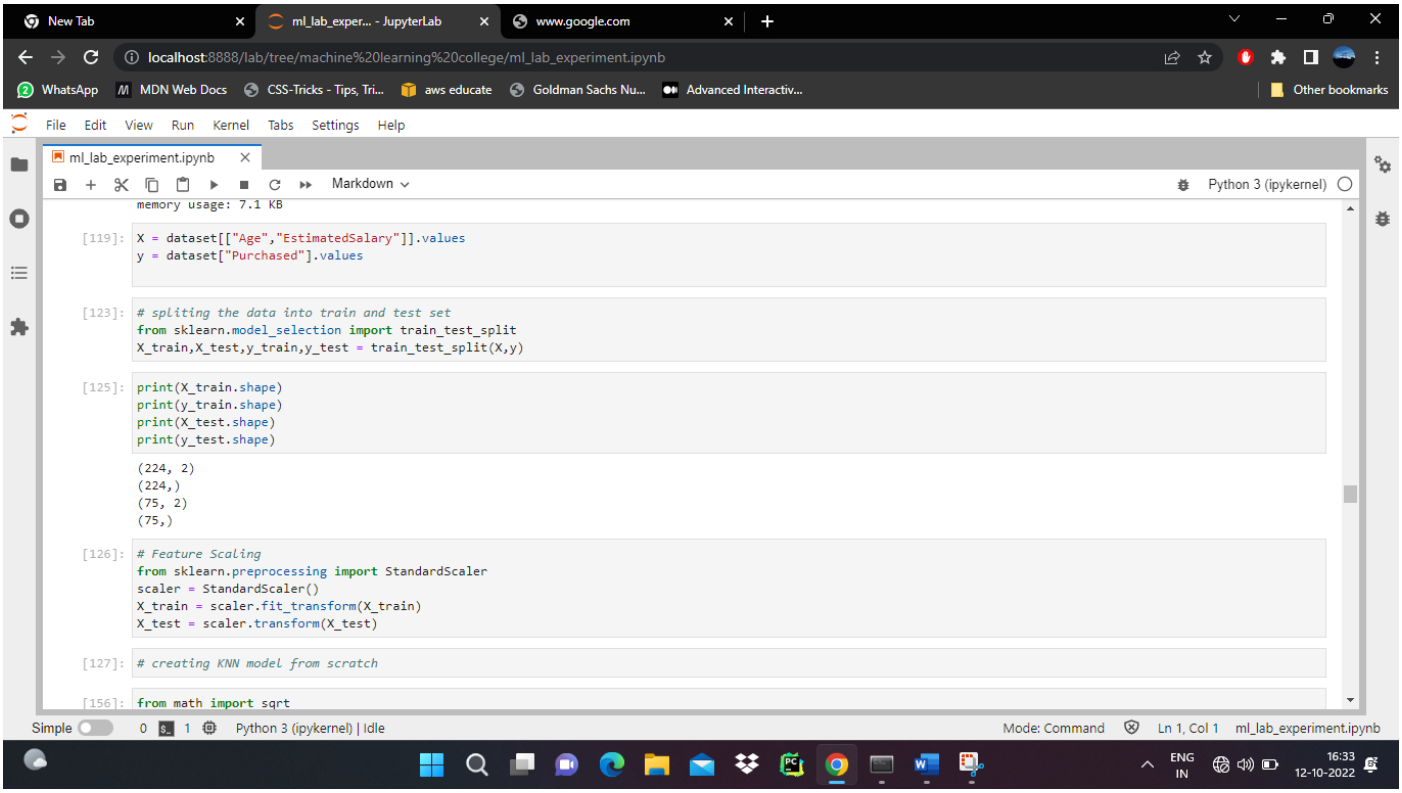
EstimatedSalary- This feature has the estimated salary of that person.

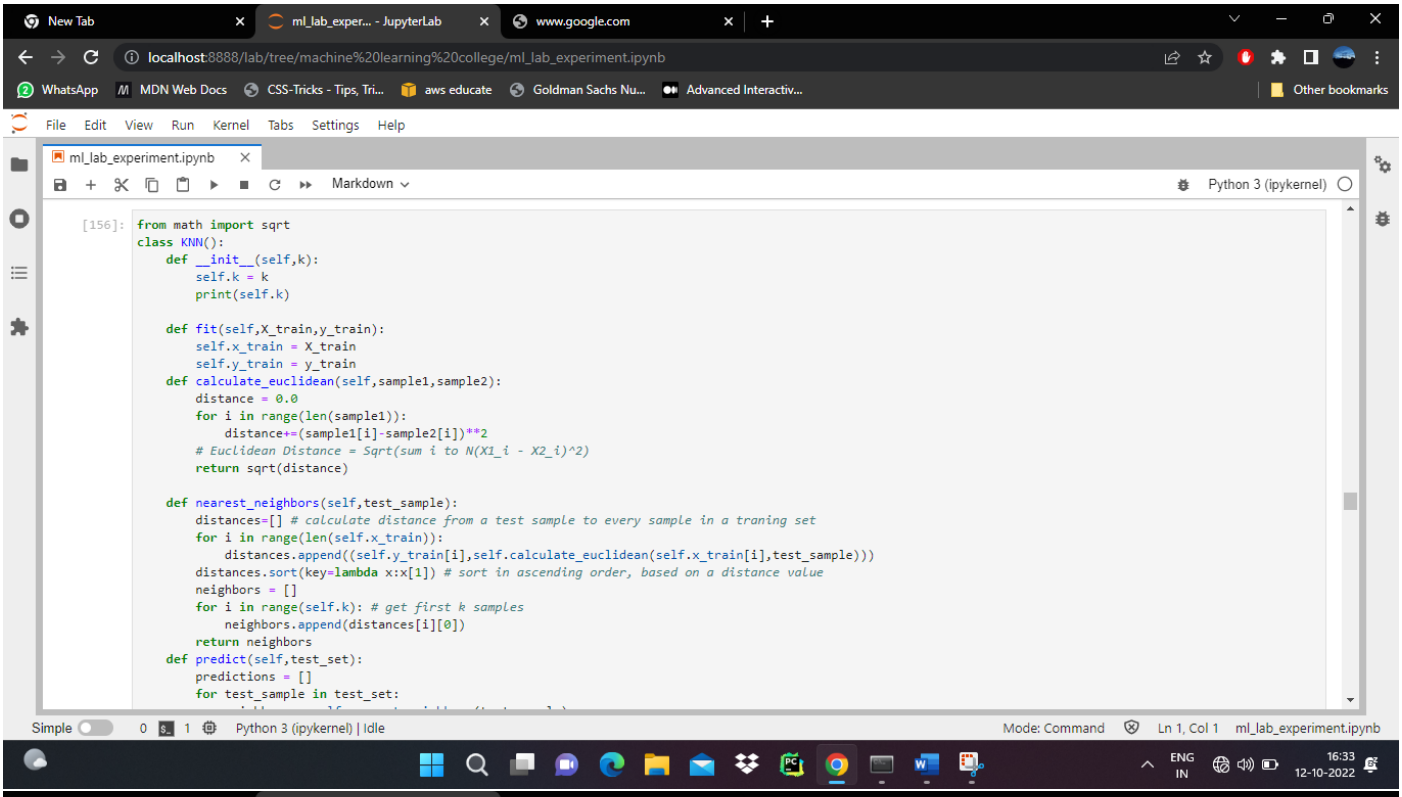
Purchased- Whether the person purchased the item after seeing the AD or not.

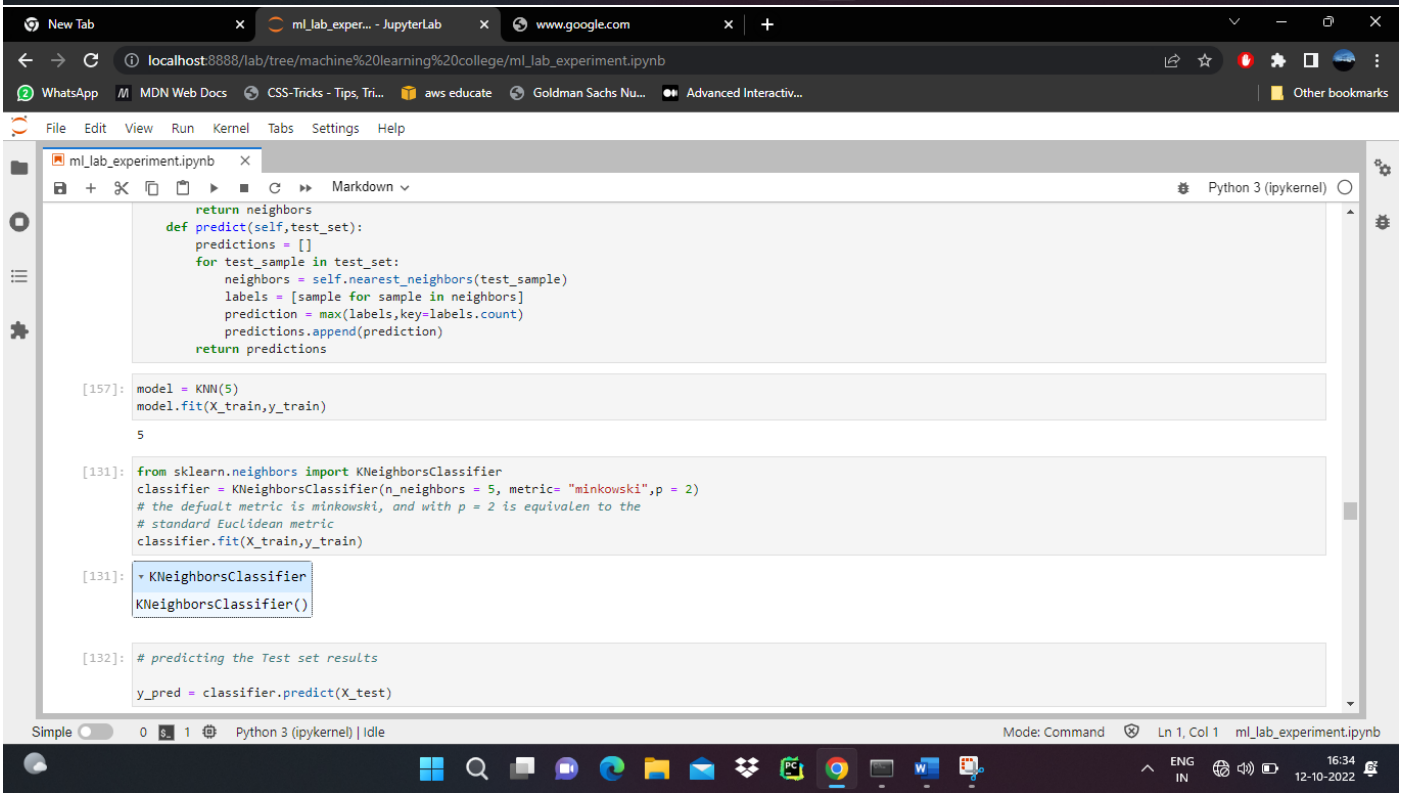
Code:

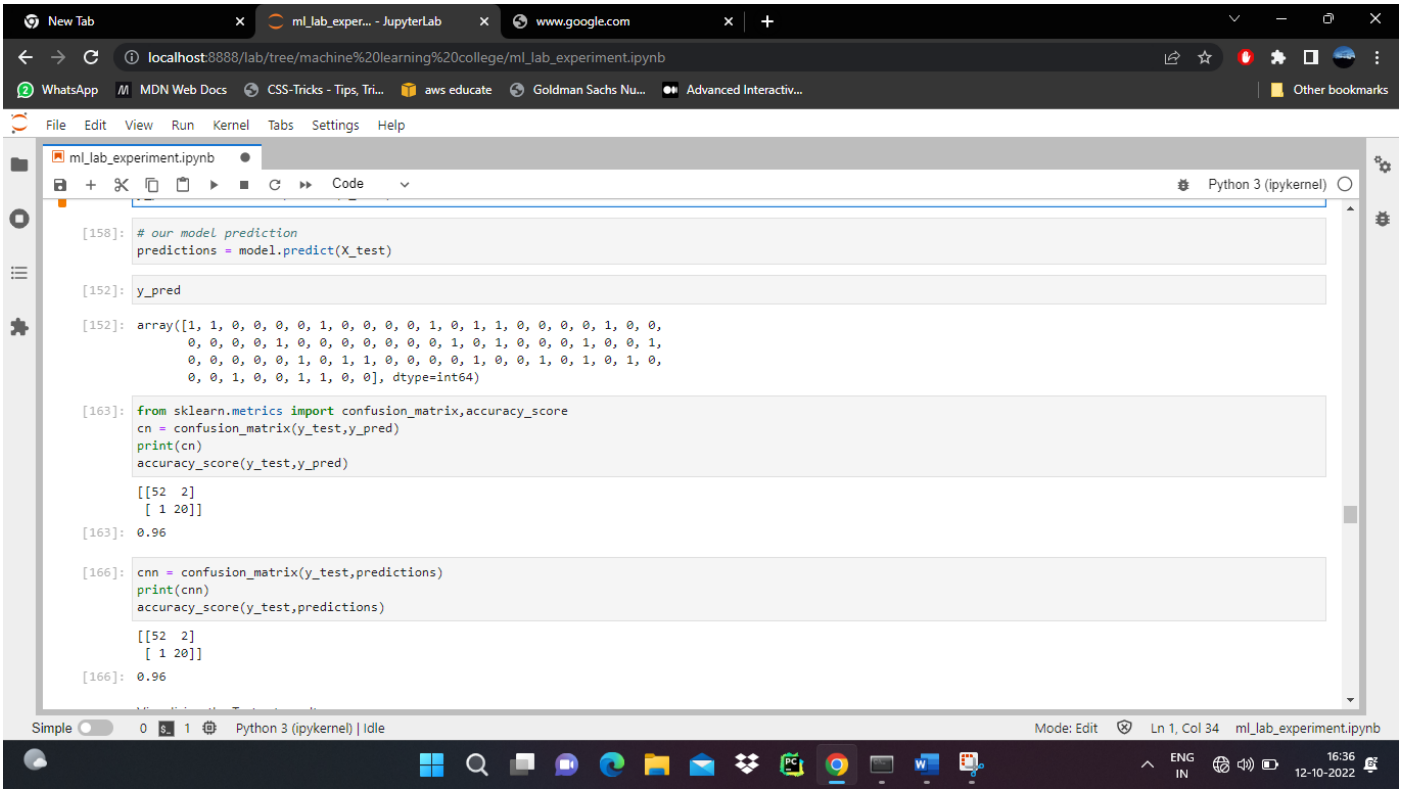

New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Markdown
Python 3 (ipykernel)
Experiment 5 - KNN
[112]: # importing the dataset
dataset = pd.read_csv("Social_Network_Ads.csv")
dataset.head()
[112]:
Age EstimatedSalary Purchased
0 19 19000 0
1 35 20000 0
2 26 43000 0
3 27 57000 0
4 19 76000 0
[113]: dataset.index
[113]: RangeIndex(start=0, stop=299, step=1)
[114]: dataset.columns
[114]: Index(['Age', 'EstimatedSalary', 'Purchased'], dtype='object')
[115]: dataset.describe()
Simple 0 1 Python 3 (ipykernel) | Idle
Mode: Command Ln 1, Col 1 ml_lab_experiment.ipynb
ENG IN 16:31 12-10-2022


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Markdown
Python 3 (ipykernel)
count 299.000000 299.000000 299.000000
mean 35.096990 69331.103679 0.267559
std 10.164451 34577.906432 0.443428
min 18.000000 15000.000000 0.000000
25% 27.000000 43000.000000 0.000000
50% 35.000000 70000.000000 0.000000
75% 41.000000 87500.000000 1.000000
max 60.000000 150000.000000 1.000000
[116]: dataset.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 3 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Age 299 non-null int64
1 EstimatedSalary 299 non-null int64
2 Purchased 299 non-null int64
dtypes: int64(3)
memory usage: 7.1 KB
[119]: X = dataset[["Age","EstimatedSalary"]].values
Simple 0 1 Python 3 (ipykernel) | Idle
Mode: Command Ln 1, Col 1 ml_lab_experiment.ipynb
ENG IN 16:32 12-10-2022


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Markdown
Python 3 (ipykernel)
memory usage: 7.1 KB
[119]:
X = dataset[["Age","EstimatedSalary"]].values
y = dataset["Purchased"].values
[123]:
# spliting the data into train and test set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y)
[125]:
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
(224, 2)
(224,)
(75, 2)
(75,)
[126]:
# Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
[127]:
creating KNN model from scratch
[156]:
from math import sqrt
Simple
0 1
Python 3 (ipykernel) | Idle
Mode: Command
Ln 1, Col 1
ml_lab_experiment.ipynb
ENG
IN
16:33
12-10-2022


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Markdown
Python 3 (ipykernel)
[156]:
from math import sqrt
class KNN():
def __init__(self,k):
self.k = k
print(self.k)
def fit(self,X_train,y_train):
self.x_train = X_train
self.y_train = y_train
def calculate_euclidean(self,sample1,sample2):
distance = 0.0
for i in range(len(sample1)):
distance+=(sample1[i]-sample2[i])**2
Euclidean Distance = Sqrt(sum i to N(X1_i - X2_i)^2)
return sqrt(distance)
def nearest_neighbors(self,test_sample):
distances=[] # calculate distance from a test sample to every sample in a traning set
for i in range(len(self.x_train)):
distances.append((self.y_train[i],self.calculate_euclidean(self.x_train[i],test_sample)))
distances.sort(key=lambda x:x[1]) # sort in ascending order, based on a distance value
neighbors = []
for i in range(self.k): # get first k samples
neighbors.append(distances[i][0])
return neighbors
def predict(self,test_set):
predictions = []
for test_sample in test_set:
Simple 0 1 Python 3 (ipykernel) | Idle
Mode: Command Ln 1, Col 1 ml_lab_experiment.ipynb
ENG IN
16:33
12-10-2022


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Markdown
Python 3 (ipykernel)
return neighbors
def predict(self,test_set):
predictions = []
for test_sample in test_set:
neighbors = self.nearest_neighbors(test_sample)
labels = [sample for sample in neighbors]
prediction = max(labels,key=labels.count)
predictions.append(prediction)
return predictions
[157]:
model = KNN(5)
model.fit(X_train,y_train)
5
[131]:
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric= "minkowski",p = 2)
# the defualt metric is minkowski, and with p = 2 is equivalen to the
# standard Euclidean metric
classifier.fit(X_train,y_train)
[131]:
KNeighborsClassifier
KNeighborsClassifier()
[132]:
# predicting the Test set results
y_pred = classifier.predict(X_test)
Simple 0 1 Python 3 (ipykernel) | Idle
Mode: Command Ln 1, Col 1 ml_lab_experiment.ipynb
ENG IN
16:34
12-10-2022



Result:

The given dataset predicts the purchase probability of a person buying a product based on the input parameters. The accuracy is:

Accuracy: 0.96

Conclusion:

The accuracy of the KNN algorithm is 96%.

Experiment: 6

Aim: To perform the K-Means Clustering algorithm on the Driver dataset.

Theory:

K-Means Clustering is an Unsupervised Learning Algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

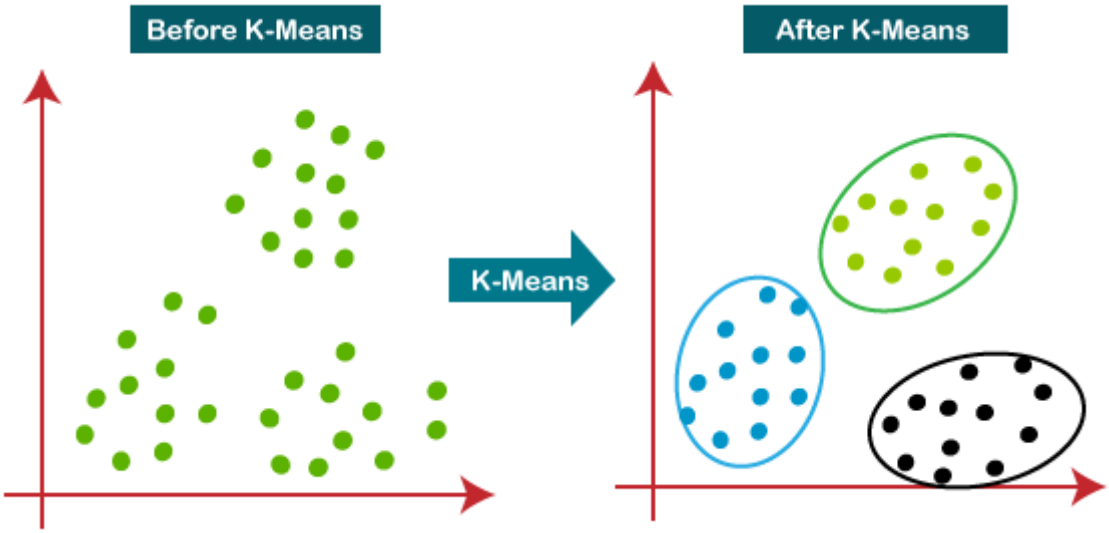
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

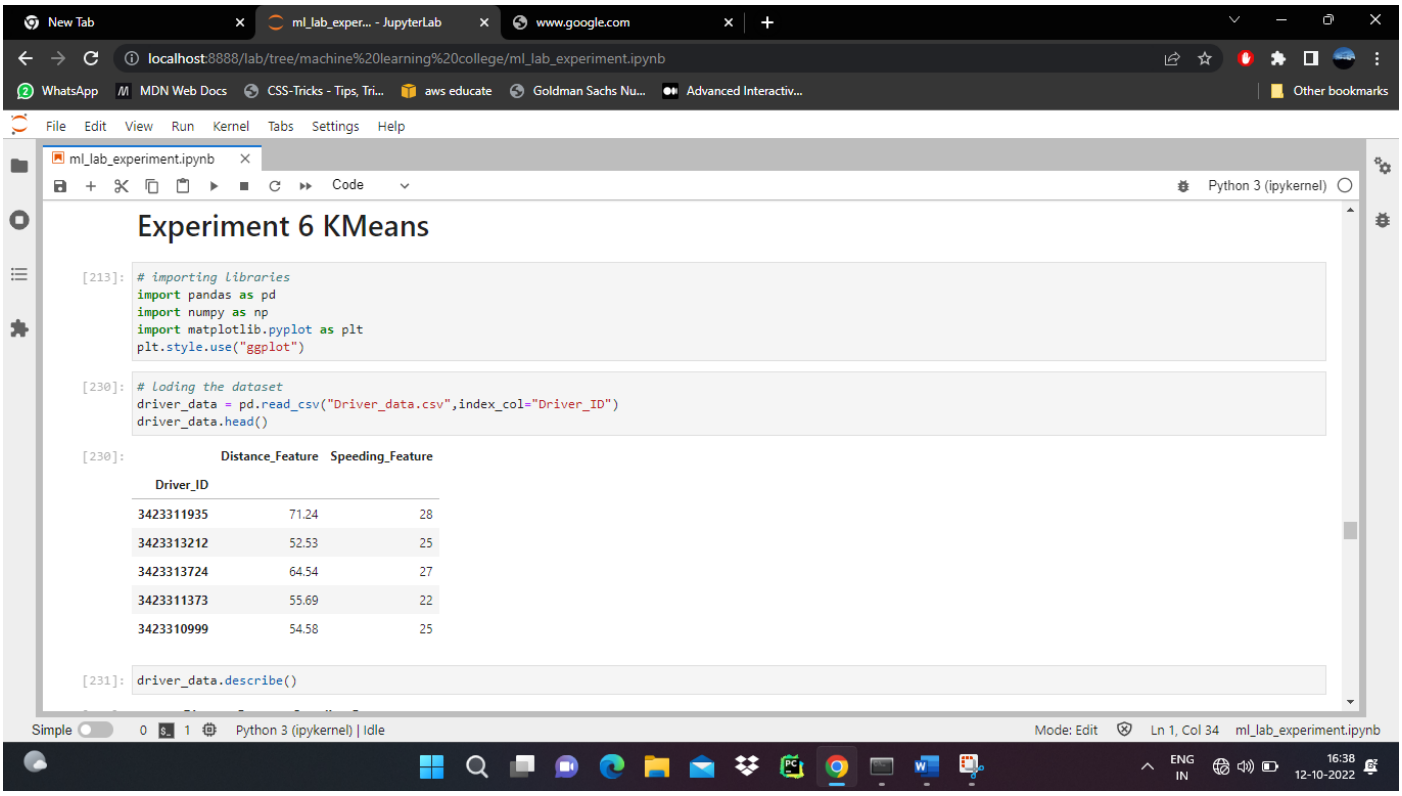
The k-means clustering algorithm mainly performs two tasks:

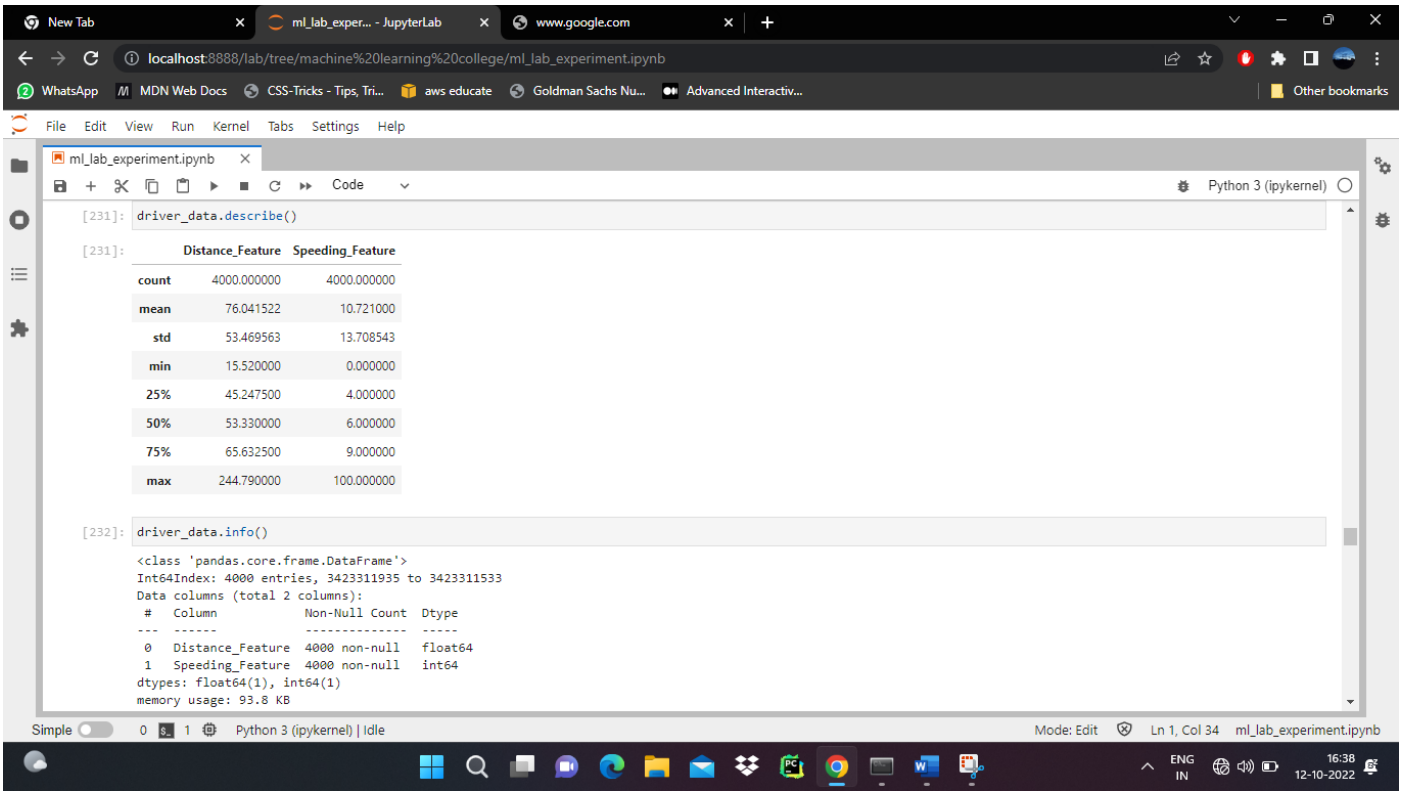
- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.



Code:


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Code
Python 3 (ipykernel)
Experiment 6 KMeans
[213]:
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("ggplot")
[230]:
# loding the dataset
driver_data = pd.read_csv("Driver_data.csv",index_col="Driver_ID")
driver_data.head()
[230]:
Distance_Feature Speeding_Feature
Driver_ID
3423311935 71.24 28
3423313212 52.53 25
3423313724 64.54 27
3423311373 55.69 22
3423310999 54.58 25
[231]:
driver_data.describe()
Simple
Python 3 (ipykernel) | Idle
Mode: Edit
Ln 1, Col 34
ml_lab_experiment.ipynb
ENG
IN
16:38
12-10-2022


New Tab
ml_lab_exper... - JupyterLab
www.google.com
localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb
WhatsApp
MDN Web Docs
CSS-Tricks - Tips, Tri...
aws educate
Goldman Sachs Nu...
Advanced Interactiv...
Other bookmarks
File Edit View Run Kernel Tabs Settings Help
ml_lab_experiment.ipynb
Code
Python 3 (ipykernel)
[231]:
driver_data.describe()
[231]:
Distance_Feature Speeding_Feature
count 4000.000000 4000.000000
mean 76.041522 10.721000
std 53.469563 13.708543
min 15.520000 0.000000
25% 45.247500 4.000000
50% 53.330000 6.000000
75% 65.632500 9.000000
max 244.790000 100.000000
[232]:
driver_data.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4000 entries, 3423311935 to 3423311533
Data columns (total 2 columns):
Column Non-Null Count Dtype
--- ------ -------------- -----
0 Distance_Feature 4000 non-null float64
1 Speeding_Feature 4000 non-null int64
dtypes: float64(1), int64(1)
memory usage: 93.8 KB
Simple
Python 3 (ipykernel) | Idle
Mode: Edit
Ln 1, Col 34
ml_lab_experiment.ipynb
ENG
IN
16:38
12-10-2022

New Tab | ml_lab_exper... - JupyterLab | www.google.com

localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb

WhatsApp | MDN Web Docs | CSS-Tricks - Tips, Tri... | aws educate | Goldman Sachs Nu... | Advanced Interactiv... | Other bookmarks

File Edit View Run Kernel Tabs Settings Help

ml_lab_experiment.ipynb

Code | Python 3 (ipykernel)

[233]:
```
from sklearn.cluster import KMeans
# Create a KMeans model with 4 clusters: model
model = KMeans(n_clusters=4)

# Use fit_predict to fit model and obtain cluster labels: labels
labels = model.fit_predict(driver_data)
```

[234]:
```
# centroids
kmeans.cluster_centers_
```

[234]:
```
array([[ 49.98800649,   5.20944484],
       [180.34311782,  10.52011494],
       [ 50.43591549,  32.39671362],
       [177.83509615,  70.28846154]])
```

[235]:
```
kmeans.labels_
```

[235]:
```
array([2, 2, 2, ..., 1, 1, 1])
```

[236]:
```
import numpy as np
unique,counts = np.unique(kmeans.labels_,return_counts=True)
```

[237]:
```
dict_data = dict(zip(unique,counts))
dict_data
```

[237]:
```
{0: 2774, 1: 696, 2: 426, 3: 104}
```

Simple | 0 | 1 | Python 3 (ipykernel) | Idle | Mode: Edit | Ln 1, Col 34 | ml_lab_experiment.ipynb

ENG IN 16:39 12-10-2022

New Tab | ml_lab_exper... - JupyterLab | www.google.com

localhost:8888/lab/tree/machine%20learning%20college/ml_lab_experiment.ipynb

WhatsApp | MDN Web Docs | CSS-Tricks - Tips, Tri... | aws educate | Goldman Sachs Nu... | Advanced Interactiv... | Other bookmarks

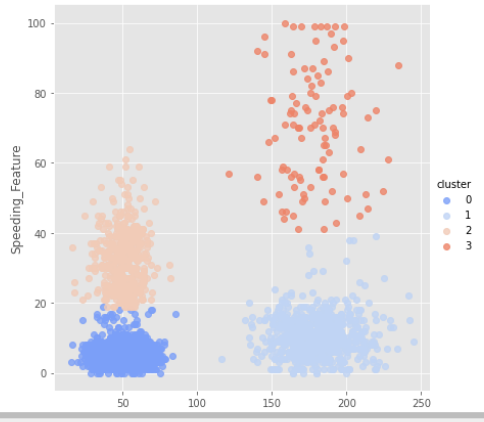
File Edit View Run Kernel Tabs Settings Help

ml_lab_experiment.ipynb

Code | Python 3 (ipykernel)

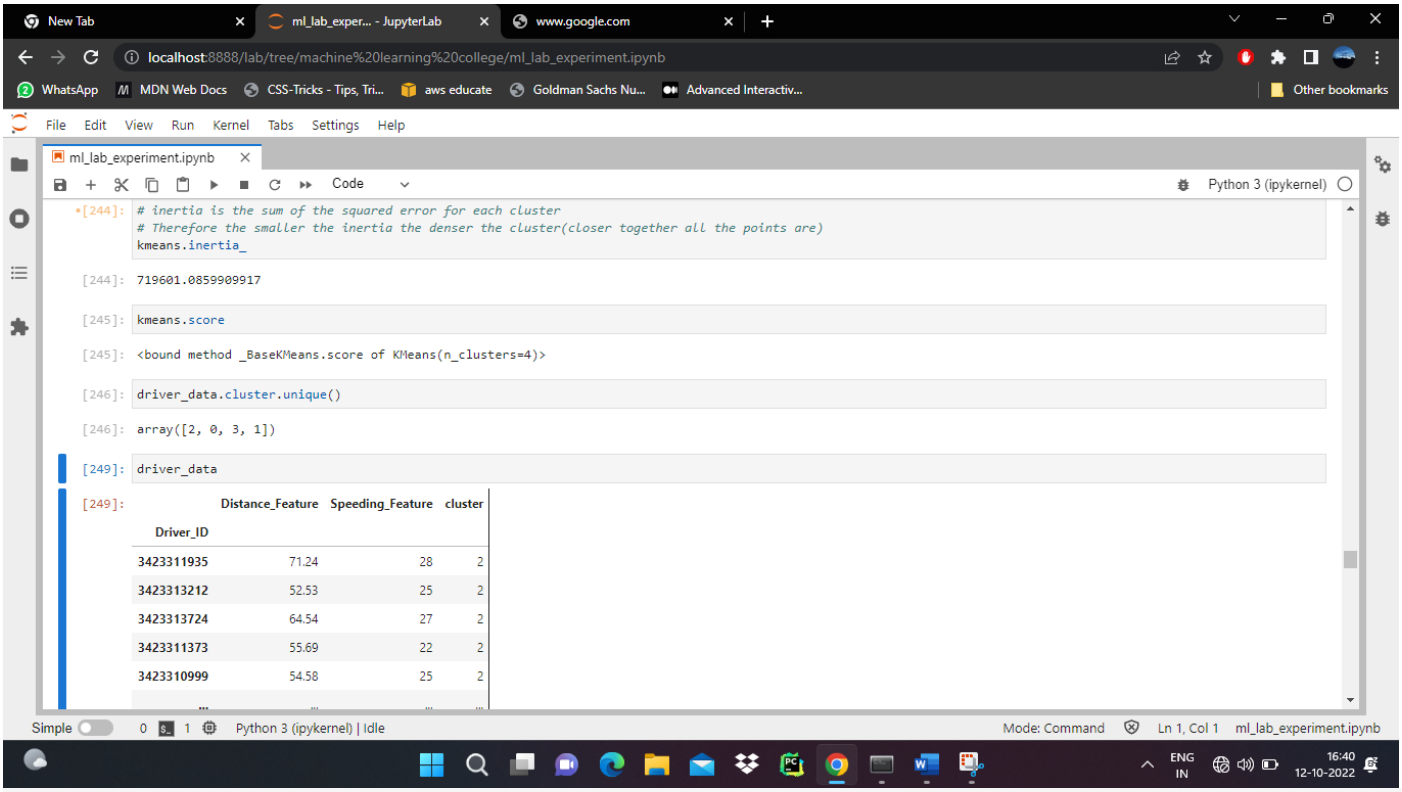
[243]:
```
import seaborn as sns
driver_data["cluster"] = kmeans.labels_
sns.lmplot("Distance_Feature","Speeding_Feature",data=driver_data,hue="cluster",palette="coolwarm",size=6,aspect=1,fit_reg=False)
```

[243]:
```
<seaborn.axisgrid.FacetGrid at 0x223d120cb80>
```

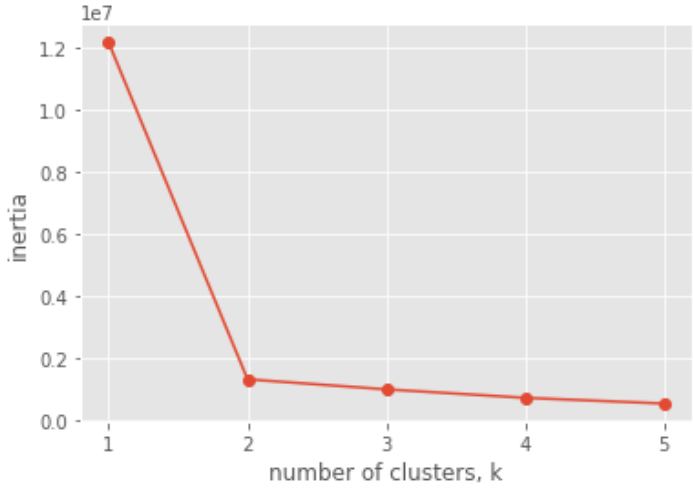


Simple | 0 | 1 | Python 3 (ipykernel) | Idle | Mode: Edit | Ln 1, Col 34 | ml_lab_experiment.ipynb

ENG IN 16:39 12-10-2022




```
# finding the best values of n_clusters
ks = range(1, 6)
inertias = []

for k in ks:
    # Create a KMeans instance with k clusters: model
    model=KMeans(n_clusters=k)
    # Fit model to samples
    model.fit(data)
    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

# Plot ks vs inertias
plt.plot(ks, inertias, '-o')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```




```
inertias
```

```
[12184626.129627977,
 1316420.850947719,
 992634.0606702475,
 719601.5383469039,
 534858.3428261087]
```

Result:

We applied K-Means Clustering Algorithm to the driver_data dataset and got k-means inertia, i.e., the sum of the square of errors for each cluster as 719601.1096991902

Conclusion:

The K-Means Algorithm gives 4 clusters for the drivers' dataset.

,

Experiment: 7

Aim: To perform a Naïve Bayes algorithm on the run or walk classification dataset.

Theory:

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

About the dataset:

This data is represented by following columns (each column contains sensor data for one of the sensor's axes):

- acceleration_x
- acceleration_y
- acceleration_z
- gyro_x
- gyro_y
- gyro_z

There is an activity type represented by "activity" column which acts as label and reflects following activities:

- "0": walking
- "1": running

Apart of that, the dataset contains "wrist" column which represents the wrist where the device was placed to collect a sample on:

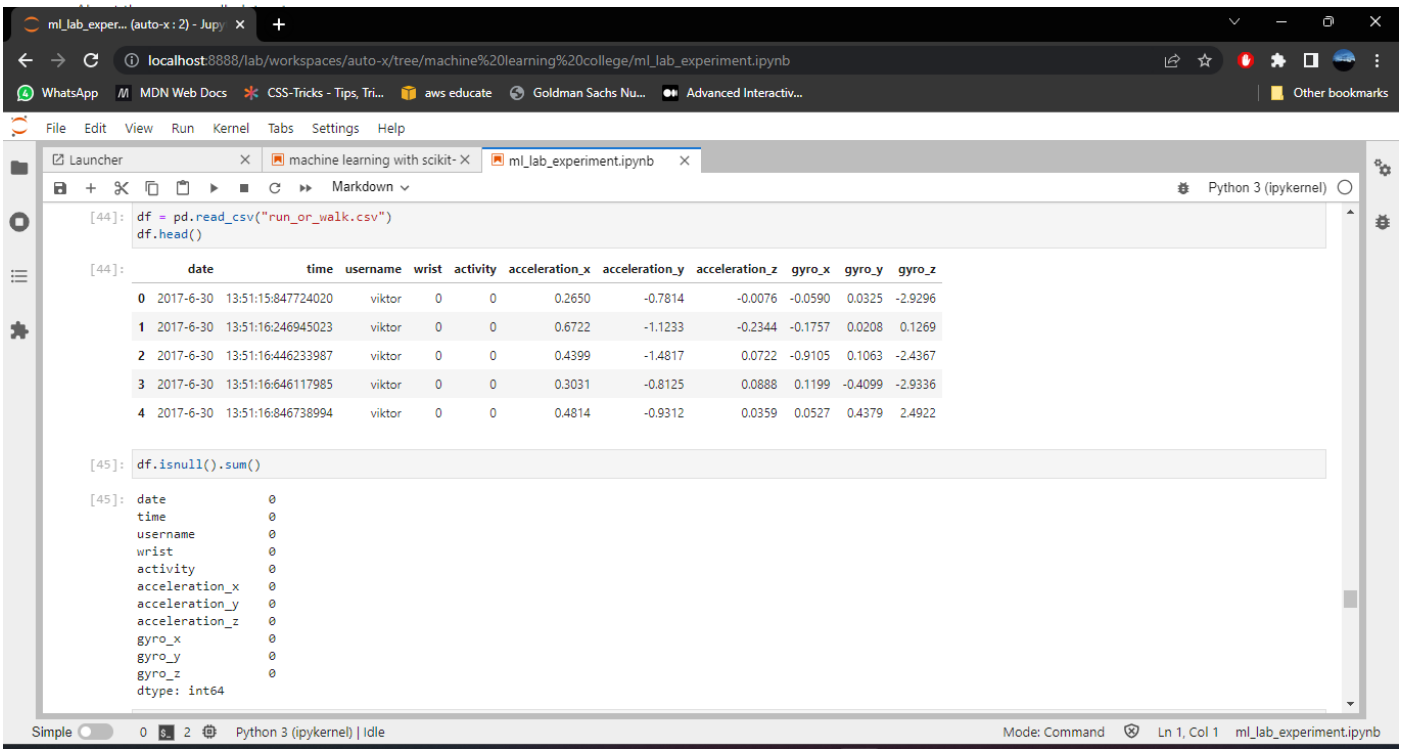
- "0": left wrist
- "1": right wrist

Additionally, the dataset contains "date", "time" and "username" columns which provide information about the exact date, time and user which collected these measurements.

Code:

Experiment 7 Naive Bayes Alogrithm

```
[43]: # import the necessary libraries
      import matplotlib.pyplot as plt
      import numpy as np
      import pandas as pd
      import seaborn as sns
      sns.set_theme(style="ticks")
```

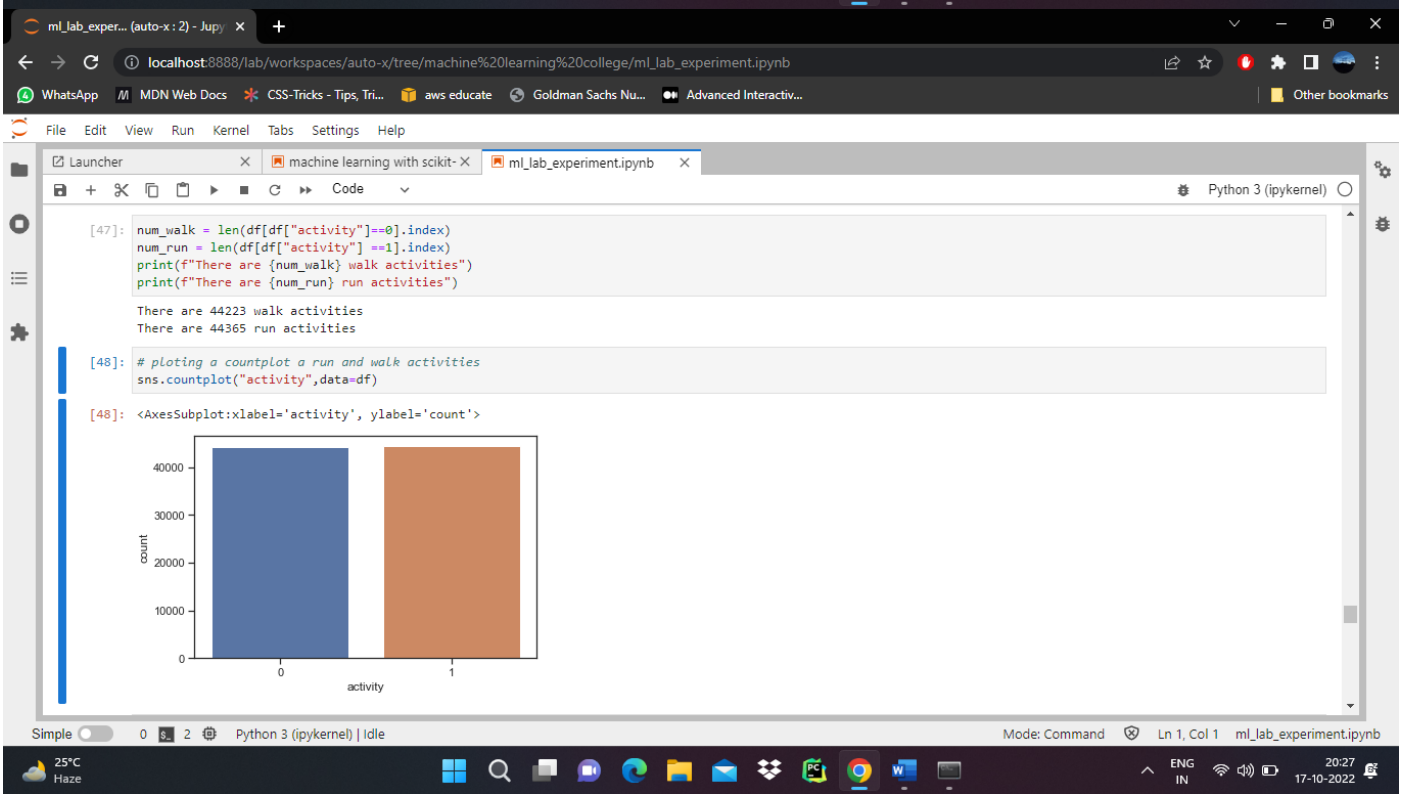


```
[44]: df = pd.read_csv("run_or_walk.csv")
      df.head()
```

	date	time	username	wrist	activity	acceleration_x	acceleration_y	acceleration_z	gyro_x	gyro_y	gyro_z
0	2017-6-30	13:51:15:847724020	viktor	0	0	0.2650	-0.7814	-0.0076	-0.0590	0.0325	-2.9296
1	2017-6-30	13:51:16:246945023	viktor	0	0	0.6722	-1.1233	-0.2344	-0.1757	0.0208	0.1269
2	2017-6-30	13:51:16:446233987	viktor	0	0	0.4399	-1.4817	0.0722	-0.9105	0.1063	-2.4367
3	2017-6-30	13:51:16:646117985	viktor	0	0	0.3031	-0.8125	0.0888	0.1199	-0.4099	-2.9336
4	2017-6-30	13:51:16:846738994	viktor	0	0	0.4814	-0.9312	0.0359	0.0527	0.4379	2.4922

```
[45]: df.isnull().sum()
```

```
[45]: date            0
      time            0
      username        0
      wrist           0
      activity        0
      acceleration_x  0
      acceleration_y  0
      acceleration_z  0
      gyro_x          0
      gyro_y          0
      gyro_z          0
      dtype: int64
```



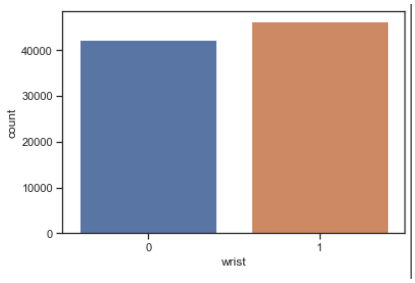
```
[47]: num_walk = len(df[df["activity"]==0].index)
      num_run = len(df[df["activity"] ==1].index)
      print(f"There are {num_walk} walk activities")
      print(f"There are {num_run} run activities")
```

```
There are 44223 walk activities
There are 44365 run activities
```

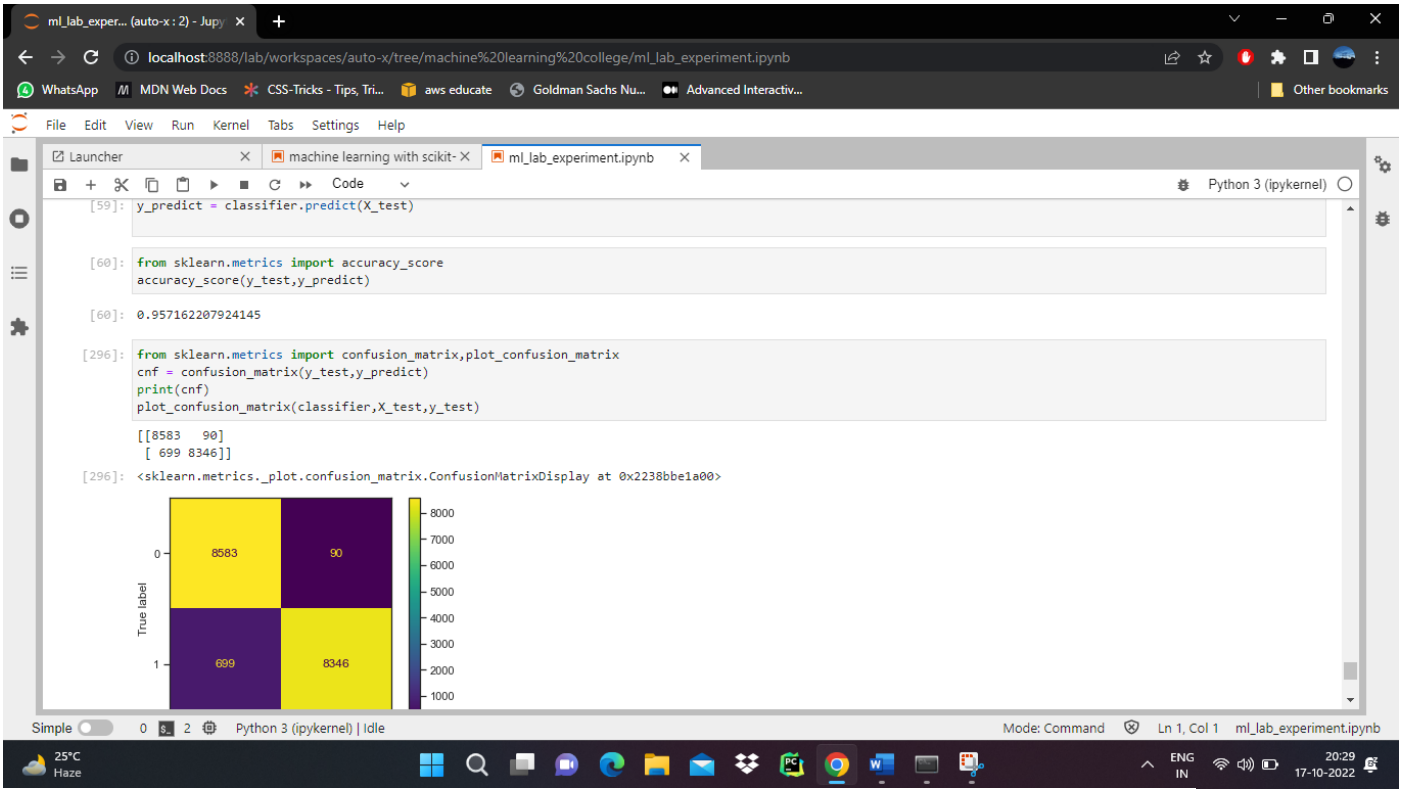
```
[48]: # ploting a countplot a run and walk activities
      sns.countplot("activity",data=df)
```

```
[48]: <AxesSubplot:xlabel='activity', ylabel='count'>
```

```
[49]: num_left = len(df[df["wrist"]==0].index)
      num_right = len(df[df["wrist"] ==1].index)
      print(f"There are {num_left} left-handed people")
      print(f"There are {num_right} right-handed people")

      There are 42330 left-handed people
      There are 46258 right-handed people

[50]: # ploting a countplot of left and right handed people
      sns.countplot("wrist",data=df)

[50]: <AxesSubplot:xlabel='wrist', ylabel='count'>
```




```
[56]: # splitting the dataset into training and testing data
      from sklearn.model_selection import train_test_split
      X,y = df.iloc[:,5:].values,df.iloc[:,4].values
      X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,
                                                       random_state=21)

[57]: print(X_train.shape)
      print(X_test.shape)
      print(y_train.shape)
      print(y_test.shape)
      print(y_test[:10])

      (70870, 6)
      (17718, 6)
      (70870,)
      (17718,)
      [0 0 0 0 1 0 1 0 0 0]

[58]: # creating a Gaussina Naive bayes model
      from sklearn.naive_bayes import GaussianNB
      classifier = GaussianNB()
      classifier.fit(X_train,y_train)

[58]: ▾ GaussianNB
      GaussianNB()
```




```
from sklearn.metrics import classification_report
target_names = ["walk","run"]
print(classification_report(y_test,y_predict,target_names=target_names))
```

```
              precision    recall  f1-score   support

        walk       0.92      0.99      0.96      8673
         run       0.99      0.92      0.95      9045

    accuracy                           0.96     17718
   macro avg       0.96      0.96      0.96     17718
weighted avg       0.96      0.96      0.96     17718
```

Result:

The given dataset predicts whether a person is running or walking based on the input parameters.
The accuracy, precision, and recall are:

Accuracy: 0957162207924145
Precision:0.955
Recall:0.96

Conclusion:

- The important features are acceleration_x, acceleration_y, acceleration_z, gyro_x, gyro_y and gyro_z for the prediction of the state of running or walking for a person.

- The accuracy of the model is 95.71% for the given dataset.

Experiment: 8

Aim: To perform a PCA on Reducing Traffic Mortality in the USA dataset.

Theory:

Principle Component Analysis (PCA) is by far the most popular dimensionality reduction algorithm. Frist it identifies the hyperplane that lies closest to the data, and then it projects the data onto it.

About the dataset:

While the rate of fatal road accidents has been decreasing steadily since the 80s, the past ten years have seen a stagnation in this reduction. Coupled with the increase in number of miles driven in the nation, the total number of traffic related-fatalities has now reached a ten year high and is rapidly increasing.

Per request of the US Department of Transportation, we are currently investigating how to derive a strategy to reduce the incidence of road accidents across the nation. By looking at the demographics of traffic accident victims for each US state, we find that there is a lot of variation between states. Now we want to understand if there are patterns in this variation in order to derive suggestions for a policy action plan. In particular, instead of implementing a costly nation-wide plan we want to focus on groups of states with similar profiles. How can we find such groups in a statistically sound way and communicate the result effectively?

To accomplish these tasks, we will make use of data wrangling, plotting, dimensionality reduction, and unsupervised clustering.

The data given to us was originally collected by the National Highway Traffic Safety Administration and the National Association of Insurance Commissioners. This particular dataset was compiled and released as a CSV-file by FiveThirtyEight under the CC-BY4.0 license.

Code:

[14]:
```
# Check the name of the current folder
current_dir = !pwd
print(current_dir)

# List all files in this folder
file_list = !ls
print(file_list)

# List all files in the datasets directory
dataset_list = !ls datasets
print(dataset_list)

# View the first 20 lines of datasets/road-accidents.csv
accidents_head = !head -n 20 datasets/road-accidents.csv
accidents_head
```

```
['/c/Users/ADITYA/datacamp machine learning projects/Reducing Traffic Mortality in the USA']
['datasets', 'notebook.ipynb']
['miles-driven.csv', 'road-accidents.csv']
```

[14]:
```
['##### LICENSE #####',
 '# This data set is modified from the original at fivethirtyeight (https://github.com/fivethirtyeight/data/tree/master/bad-drivers)',
 '# and it is released under CC BY 4.0 (https://creativecommons.org/licenses/by/4.0/)',
 '##### COLUMN ABBREVIATIONS #####',
 '# drvr_fatl_col_bmiles = Number of drivers involved in fatal collisions per billion miles (2011)',
 '# perc_fatl_speed = Percentage Of Drivers Involved In Fatal Collisions Who Were Speeding (2009)',
 '# perc_fatl_alcohol = Percentage Of Drivers Involved In Fatal Collisions Who Were Alcohol-Impaired (2011)',
 '# perc_fatl_1st_time = Percentage Of Drivers Involved In Fatal Collisions Who Had Not Been Involved In Any Previous Accidents (2011)',
 '##### DATA BEGIN #####',
 'state|drvr_fatl_col_bmiles|perc_fatl_speed|perc_fatl_alcohol|perc_fatl_1st_time',
 'Alabama|18.8|39|30|80',
 'Alaska|18.1|41|25|94',
 'Arizona|18.6|35|28|96',
 'Arkansas|22.4|18|26|95',
 'California|12|35|28|89',
 'Colorado|13.6|37|28|95',
 'Connecticut|10.8|46|36|82',
 'Delaware|16.2|38|30|99',
 'District of Columbia|5.9|34|27|100',
 'Florida|17.9|21|29|94']
```

2. Read in and get an overview of the data

Next, we will orient ourselves to get to know the data with which we are dealing.

[15]:
```
# Import the `pandas` module as "pd"
import pandas as pd
# import warnings
import warnings
warnings.filterwarnings("ignore")


# Read in `road-accidents.csv`
car_acc = pd.read_csv('datasets/road-accidents.csv', comment='#', sep='|')

# Save the number of rows columns as a tuple
rows_and_cols = car_acc.shape
print('There are {} rows and {} columns.\n'.format(
    rows_and_cols[0], rows_and_cols[1]))

# Generate an overview of the DataFrame
car_acc_information = car_acc.info()
print(car_acc_information)

# Display the last five rows of the DataFrame
car_acc.tail()
```

```
There are 51 rows and 5 columns.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 5 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   state                 51 non-null     object
 1   drvr_fatl_col_bmiles  51 non-null     float64
 2   perc_fatl_speed       51 non-null     int64
 3   perc_fatl_alcohol     51 non-null     int64
 4   perc_fatl_1st_time    51 non-null     int64
dtypes: float64(1), int64(3), object(1)
memory usage: 2.1+ KB
None
```

[15]:

	state	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time
46	Virginia	12.7	19	27	88
47	Washington	10.6	42	33	86
48	West Virginia	23.8	34	28	87
49	Wisconsin	13.8	36	33	84
50	Wyoming	17.4	42	32	90

3. Create a textual and a graphical summary of the data

We now have an idea of what the dataset looks like. To further familiarize ourselves with this data, we will calculate summary statistics and produce a graphical overview of the data. The graphical overview is good to get a sense for the distribution of variables within the data and could consist of one histogram per column. It is often a good idea to also explore the pairwise relationship between all columns in the data set by using a using pairwise scatter plots (sometimes referred to as a "scatterplot matrix").

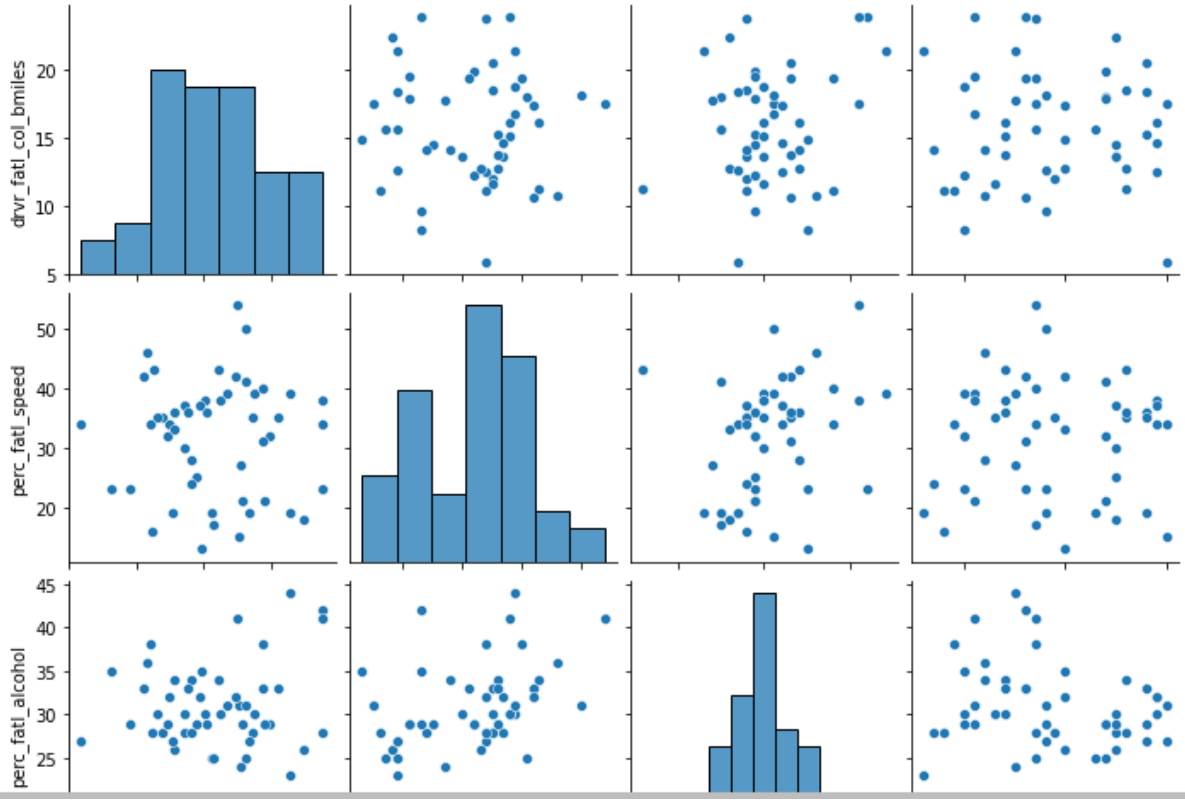
[16]:

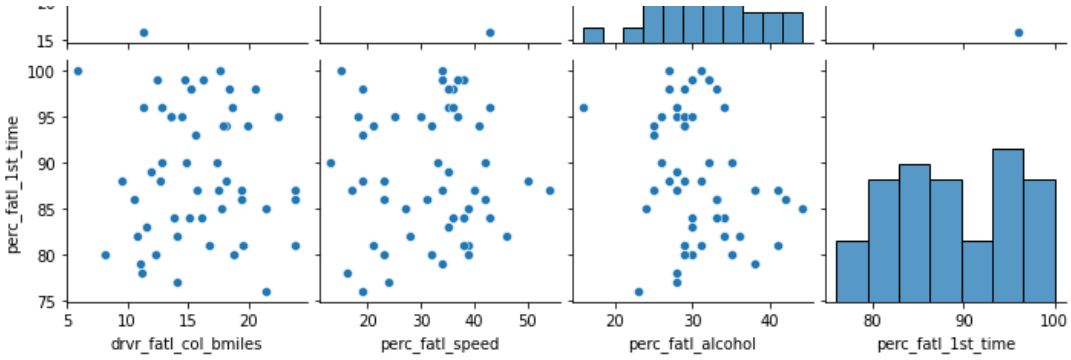
```
# import seaborn and make plots appear inline
import seaborn as sns
%matplotlib inline

# Compute the summary statistics of all columns in the `car_acc` DataFrame
sum_stat_car = car_acc.describe()
print(sum_stat_car)

# Create a pairwise scatter plot to explore the data
sns.pairplot(car_acc)
```

```
       drvr_fatl_col_bmiles  perc_fatl_speed  perc_fatl_alcohol  \
count             51.000000        51.000000          51.000000
mean              15.790196        31.725490          30.686275
std                4.122002         9.633438           5.132213
min                5.900000        13.000000          16.000000
25%               12.750000        23.000000          28.000000
50%               15.600000        34.000000          30.000000
75%               18.500000        38.000000          33.000000
max               23.900000        54.000000          44.000000

       perc_fatl_1st_time
count            51.00000
mean             88.72549
std               6.96011
min              76.00000
25%              83.50000
50%              88.00000
75%              95.00000
max             100.00000
```

[16]: `<seaborn.axisgrid.PairGrid at 0x13ebf4c3940>`

[16]: <seaborn.axisgrid.PairGrid at 0x15ed14c5940>





4. Quantify the association of features and accidents

We can already see some potentially interesting relationships between the target variable (the number of fatal accidents) and the feature variables (the remaining three columns).

To quantify the pairwise relationships that we observed in the scatter plots, we can compute the Pearson correlation coefficient matrix. The Pearson correlation coefficient is one of the most common methods to quantify correlation between variables, and by convention, the following thresholds are usually used:

- 0.2 = weak
- 0.5 = medium
- 0.8 = strong
- 0.9 = very strong

17]:
```
# Compute the correlation coefficent for all column pairs
corr_columns = car_acc.corr()
corr_columns
```

17]:

	drvr_fatl_col_bmiles	perc_fatl_speed	perc_fatl_alcohol	perc_fatl_1st_time
drvr_fatl_col_bmiles	1.000000	-0.029080	0.199426	-0.017942
perc_fatl_speed	-0.029080	1.000000	0.286244	0.014066
perc_fatl_alcohol	0.199426	0.286244	1.000000	-0.245455
perc_fatl_1st_time	-0.017942	0.014066	-0.245455	1.000000

5. Fit a multivariate linear regression

From the correlation table, we see that the amount of fatal accidents is most strongly correlated with alcohol consumption (first row). But in addition, we also see that some of the features are correlated with each other, for instance, speeding and alcohol consumption are positively correlated. We, therefore, want to compute the association of the target with each feature while adjusting for the effect of the remaining features. This can be done using multivariate linear regression.

Both the multivariate regression and the correlation measure how strongly the features are associated with the outcome (fatal accidents). When comparing the regression coefficients with the correlation coefficients, we will see that they are slightly different. The reason for this is that the multiple regression computes the association of a feature with an outcome, given the association with all other features, which is not accounted for when calculating the correlation coefficients.

A particularly interesting case is when the correlation coefficient and the regression coefficient of the same feature have opposite signs. How can this be? For example, when a feature A is positively correlated with the outcome Y but also positively correlated with a different feature B that has a negative effect on Y, then the indirect correlation (A->B->Y) can overwhelm the direct correlation (A->Y). In such a case, the regression coefficient of feature A could be positive, while the correlation coefficient is negative. This is sometimes called a *masking* relationship. Let's see if the multivariate regression can reveal such a phenomenon.

18]:
```
# Import the linear model function from sklearn
from sklearn.linear_model import LinearRegression

# Create the features and target DataFrames
features = car_acc[["perc_fatl_speed","perc_fatl_alcohol","perc_fatl_1st_time"]]
target = car_acc["drvr_fatl_col_bmiles"]

# Create a linear regression object
reg = LinearRegression()
```

```
# Fit a multivariate linear regression model
reg.fit(features,target)

# Retrieve the regression coefficients
fit_coef = reg.coef_
fit_coef
```

[18]:
```
array([-0.04180041,  0.19086404,  0.02473301])
```

6. Perform PCA on standardized data

We have learned that alcohol consumption is weakly associated with the number of fatal accidents across states. This could lead us to conclude that alcohol consumption should be a focus for further investigations and maybe strategies should divide states into high versus low alcohol consumption in accidents. But there are also associations between alcohol consumptions and the other two features, so it might be worth trying to split the states in a way that accounts for all three features.

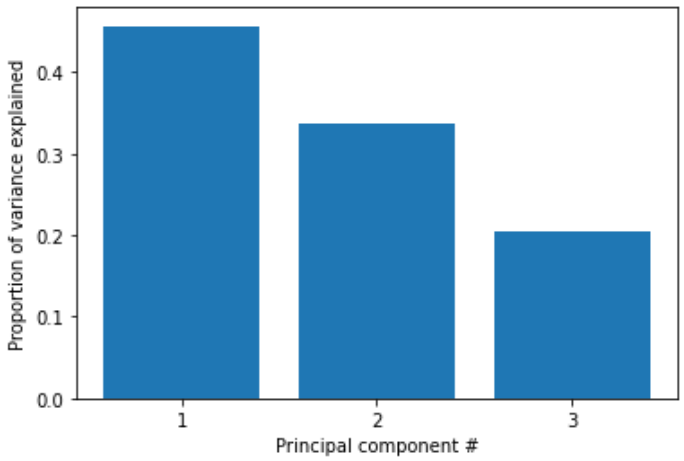
One way of clustering the data is to use PCA to visualize data in reduced dimensional space where we can try to pick up patterns by eye. PCA uses the absolute variance to calculate the overall variance explained for each principal component, so it is important that the features are on a similar scale (unless we would have a particular reason that one feature should be weighted more).

We'll use the appropriate scaling function to standardize the features to be centered with mean 0 and scaled with standard deviation 1.

[19]:
```
# Standardize and center the feature columns
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Import the PCA class function from sklearn
from sklearn.decomposition import PCA
pca = PCA()

# Fit the standardized data to the pca
pca.fit(features_scaled)

# Plot the proportion of variance explained on the y-axis of the bar plot
import matplotlib.pyplot as plt
plt.bar(range(1, pca.n_components_ + 1),  pca.explained_variance_ratio_)
plt.xlabel('Principal component #')
plt.ylabel('Proportion of variance explained')
plt.xticks([1, 2, 3])

# Compute the cumulative proportion of variance explained by the first two principal components
two_first_comp_var_exp = pca.explained_variance_ratio_.cumsum()[1]
print("The cumulative variance of the first two principal components is {}".format(
    round(two_first_comp_var_exp, 5)))
```

```
The cumulative variance of the first two principal components is 0.7947
```

```
The cumulative variance of the first two principal components is 0.7947
```



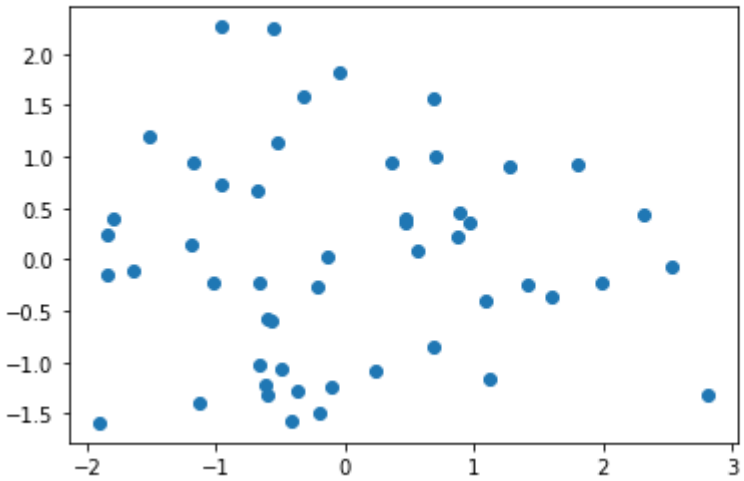
7. Visualize the first two principal components

The first two principal components enable visualization of the data in two dimensions while capturing a high proportion of the variation (79%) from all three features: speeding, alcohol influence, and first-time accidents. This enables us to use our eyes to try to discern patterns in the data with the goal to find groups of similar states. Although clustering algorithms are becoming increasingly efficient, human pattern recognition is an easily accessible and very efficient method of assessing patterns in data.

We will create a scatter plot of the first principle components and explore how the states cluster together in this visualization.

```
]: # Transform the scaled features using two principal components
pca = PCA(n_components=2)
p_comps = pca.fit_transform(features_scaled)

# Extract the first and second component to use for the scatter plot
p_comp1 = p_comps[:,0]
p_comp2 = p_comps[:,1]

# Plot the first two principal components in a scatter plot
plt.scatter(p_comp1,p_comp2)
```

```
[20]: <matplotlib.collections.PathCollection at 0x13eda3b4250>
```

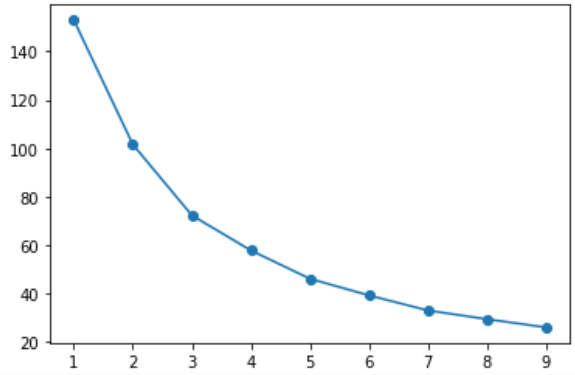


8. Find clusters of similar states in the data

It was not entirely clear from the PCA scatter plot how many groups in which the states cluster. To assist with identifying a reasonable number of clusters, we can use KMeans clustering by creating a scree plot and finding the "elbow", which is an indication of when the addition of more clusters does not add much explanatory power.

1]:
```
# Import KMeans from sklearn
from sklearn.cluster import KMeans


# A loop will be used to plot the explanatory power for up to 10 KMeans clusters
ks = range(1, 10)
inertias = []
for k in ks:
    # Initialize the KMeans object using the current number of clusters (k)
    km = KMeans(n_clusters=k, random_state=8)
    # Fit the scaled features to the KMeans object
    km.fit(features_scaled)
    # Append the inertia for `km` to the list of inertias
    inertias.append(km.inertia_)

# Plot the results in a line plot
plt.plot(ks,inertias, marker='o')
```

[21]: `[<matplotlib.lines.Line2D at 0x13eda4270a0>]`



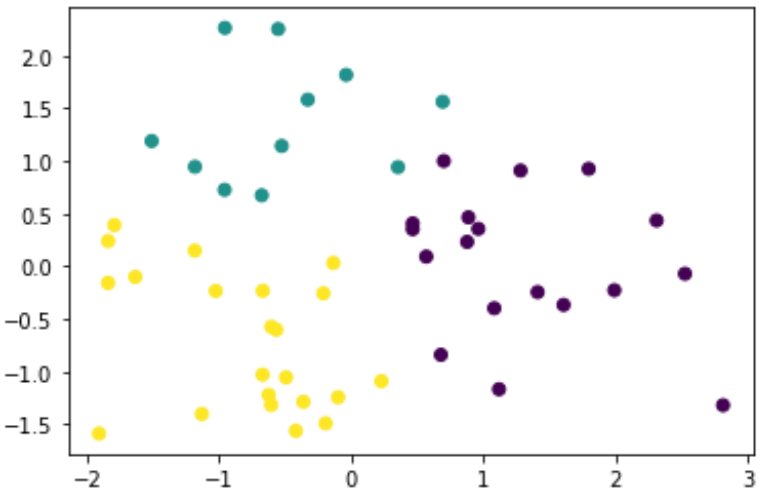
9. KMeans to visualize clusters in the PCA scatter plot

Since there wasn't a clear elbow in the scree plot, assigning the states to either two or three clusters is a reasonable choice, and we will resume our analysis using three clusters. Let's see how the PCA scatter plot looks if we color the states according to the cluster to which they are assigned.

2]:
```
# Create a KMeans object with 3 clusters, use random_state=8
km = KMeans(n_clusters=3,random_state=8)

# Fit the data to the `km` object
km.fit(features_scaled)

# Create a scatter plot of the first two principal components
# and color it according to the KMeans cluster assignment
plt.scatter(p_comp1,p_comp2,c=km.labels_)
```

2]: `<matplotlib.collections.PathCollection at 0x13eda48f4f0>`

22]: <matplotlib.collections.PathCollection at 0x13eda48f4f0>



10. Visualize the feature differences between the clusters

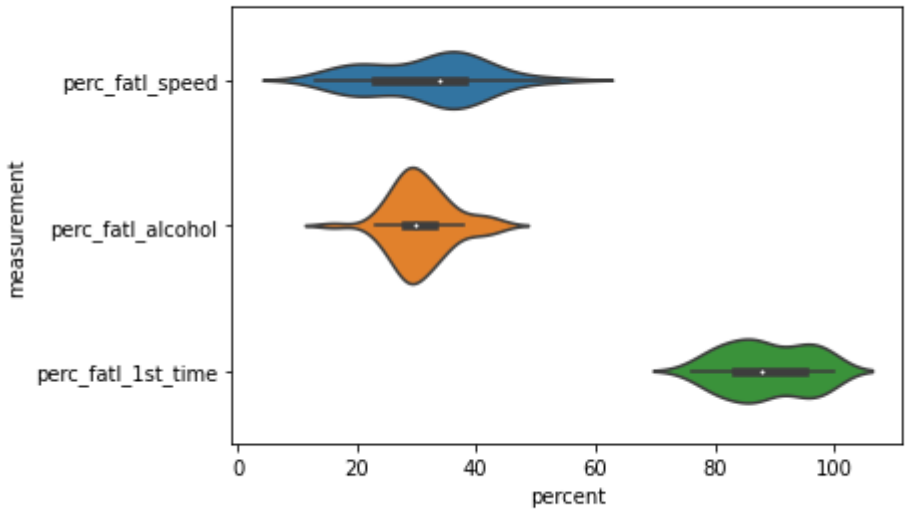
Thus far, we have used both our visual interpretation of the data and the KMeans clustering algorithm to reveal patterns in the data, but what do these patterns mean?

Remember that the information we have used to cluster the states into three distinct groups are the percentage of drivers speeding, under alcohol influence and that has not previously been involved in an accident. We used these clusters to visualize how the states group together when considering the first two principal components. This is good for us to understand structure in the data, but not always easy to understand, especially not if the findings are to be communicated to a non-specialist audience.

A reasonable next step in our analysis is to explore how the three clusters are different in terms of the three features that we used for clustering. Instead of using the scaled features, we return to using the unscaled features to help us interpret the differences.

```
# Create a new column with the labels from the KMeans clustering
car_acc['cluster'] = km.labels_

# Reshape the DataFrame to the long format
melt_car = pd.melt(car_acc, id_vars='cluster', var_name="measurement",value_name="percent",
                   value_vars=["perc_fatl_speed","perc_fatl_alcohol","perc_fatl_1st_time"])

# Create a violin plot splitting and coloring the results according to the km-clusters
sns.violinplot(y="measurement",x="percent",data=melt_car)
```

: <AxesSubplot:xlabel='percent', ylabel='measurement'>

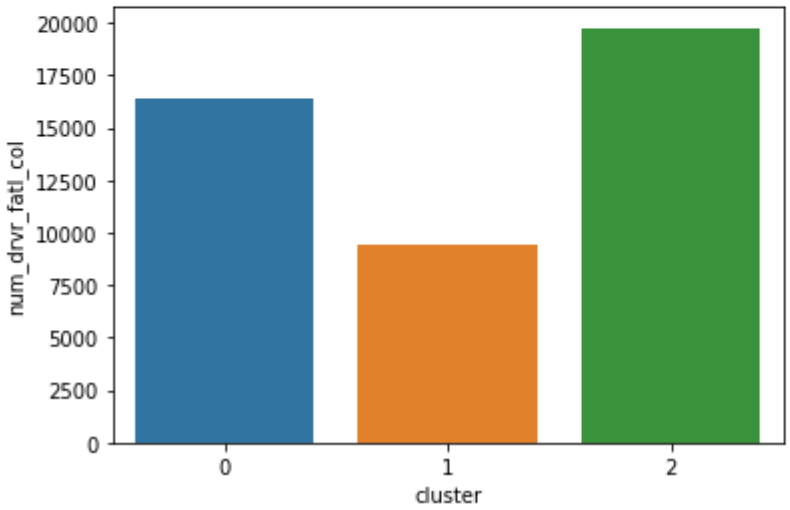


11. Compute the number of accidents within each cluster

Now it is clear that different groups of states may require different interventions. Since resources and time are limited, it is useful to start off with an intervention in one of the three groups first. Which group would this be? To determine this, we will include data on how many miles are driven in each state, because this will help us to compute the total number of fatal accidents in each state. Data on miles driven is available in another tab-delimited text file. We will assign this new information to a column in the DataFrame and create a violin plot for how many total fatal traffic accidents there are within each state cluster.

```
# Read in the new dataset
miles_driven = pd.read_csv('datasets/miles-driven.csv', sep='|')

# Merge the `car_acc` DataFrame with the `miles_driven` DataFrame
car_acc_miles = car_acc.merge(miles_driven,on="state")

# Create a new column for the number of drivers involved in fatal accidents
car_acc_miles['num_drvr_fatl_col'] = car_acc_miles["drvr_fatl_col_bmiles"] * car_acc_miles["million_miles_annually"]/1000

# Create a barplot of the total number of accidents per cluster
sns.barplot(x="cluster", y="num_drvr_fatl_col", data=car_acc_miles, estimator=sum, ci=None)

# Calculate the number of states in each cluster and their 'num_drvr_fatl_col' mean and sum.
count_mean_sum = car_acc_miles.groupby('cluster')['num_drvr_fatl_col'].agg(['count', 'mean', 'sum'])
count_mean_sum
```

cluster	count	mean	sum
0	18	911.406439	16405.3159
1	11	860.505945	9465.5654
2	22	898.378595	19764.3291



12. Make a decision when there is no clear right choice

As we can see, there is no obvious correct choice regarding which cluster is the most important to focus on. Yet, we can still argue for a certain cluster and motivate this using our findings above. Which cluster do you think should be a focus for policy intervention and further investigation?

```
# Which cluster would you choose?
cluster_num = 2
```

Experiment: 9

Aim: To Perform Gradient Boosting Ensemble Algorithm.

Theory:

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

There is a technique called the Gradient Boosted Trees whose base learner is CART (Classification and Regression Trees).

Code:

```
# creating a simple quadratic dataset:
np.random.seed(42)
X = np.random.rand(100,1) - 0.5
y = 3*X[:,0] **2 +0.05*np.random.randn(100)
```

now let's train a decision tree regressor on this dataset:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth = 2, random_state=42)
tree_reg1.fit(X,y)
```

```
DecisionTreeRegressor(max_depth=2, random_state=42)
```

```
y2 = y-tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X,y2)
```

```
DecisionTreeRegressor(max_depth=2, random_state=42)
```

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

```
DecisionTreeRegressor(max_depth=2, random_state=42)
```

```
X_new = np.array([[0.8]])
```

```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

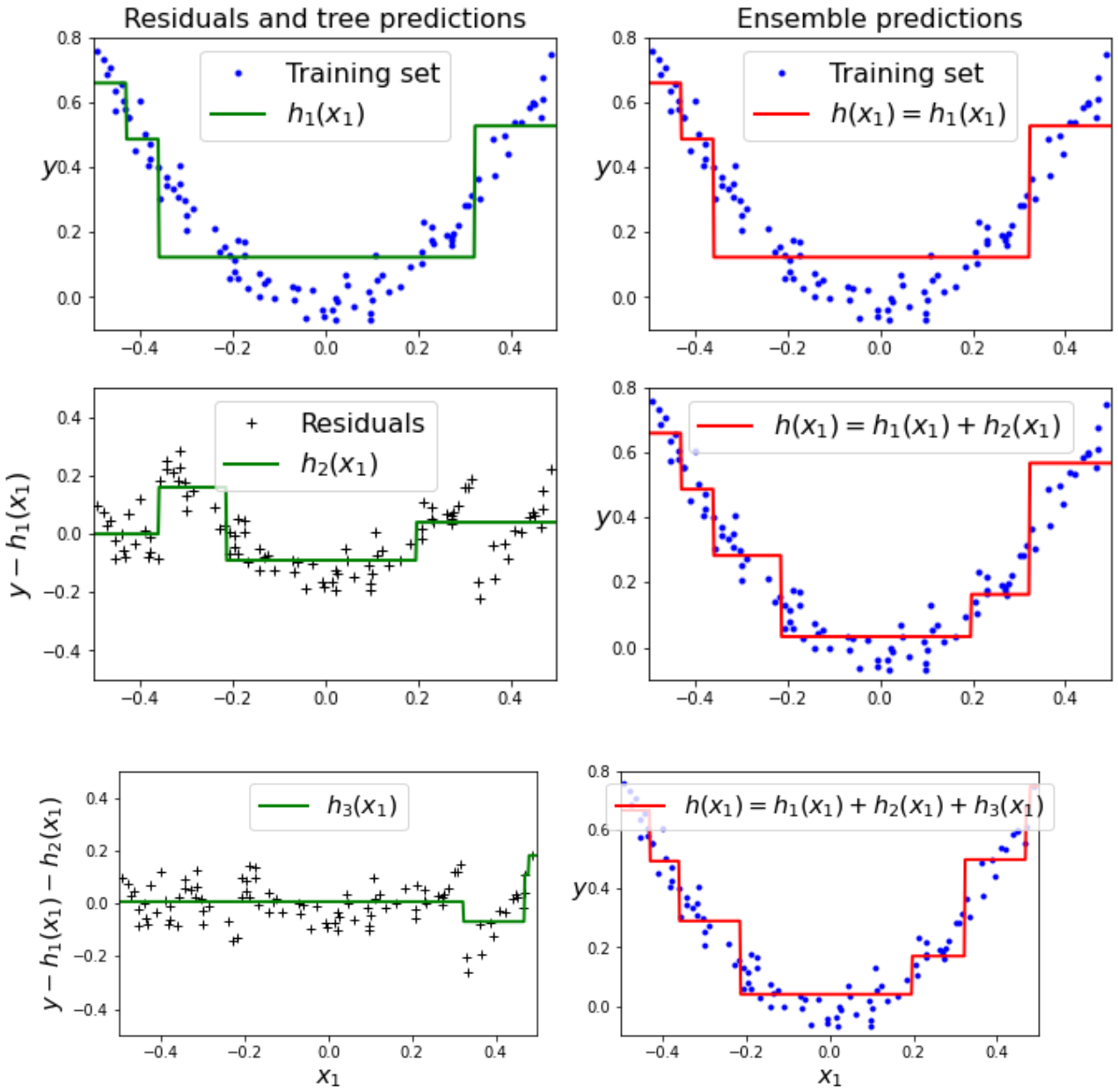
```
y_pred
```

```
array([0.75026781])
```

In this depiction of Gradient Boosting, the first predictor (top left) is trained normally, then each consecutive predictor (middle left and lower left) is trained on the previous predictor's residuals; the right column shows the resulting ensemble's predictions:

```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b.", data_label=None):
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(11,11))

plt.subplot(321)
plot_predictions([tree_reg1], X, y, axes=[-0.5, 0.5, -0.1, 0.8], label="$h_1(x_1)$", style="g-", data_label="Training set")
plt.ylabel("$y$", fontsize=16, rotation=0)
plt.title("Residuals and tree predictions", fontsize=16)

plt.subplot(322)
plot_predictions([tree_reg1], X, y, axes=[-0.5, 0.5, -0.1, 0.8], label="$h(x_1) = h_1(x_1)$", data_label="Training set")
plt.ylabel("$y$", fontsize=16, rotation=0)
plt.title("Ensemble predictions", fontsize=16)

plt.subplot(323)
plot_predictions([tree_reg2], X, y2, axes=[-0.5, 0.5, -0.5, 0.5], label="$h_2(x_1)$", style="g-", data_style="k+", data_label="Residuals")
plt.ylabel("$y - h_1(x_1)$", fontsize=16)

plt.subplot(324)
plot_predictions([tree_reg1, tree_reg2], X, y, axes=[-0.5, 0.5, -0.1, 0.8], label="$h(x_1) = h_1(x_1) + h_2(x_1)$")
plt.ylabel("$y$", fontsize=16, rotation=0)

plt.subplot(325)
plot_predictions([tree_reg3], X, y3, axes=[-0.5, 0.5, -0.5, 0.5], label="$h_3(x_1)$", style="g-", data_style="k+")
plt.ylabel("$y - h_1(x_1) - h_2(x_1)$", fontsize=16)
plt.xlabel("$x_1$", fontsize=16)

plt.subplot(326)
plot_predictions([tree_reg1, tree_reg2, tree_reg3], X, y, axes=[-0.5, 0.5, -0.1, 0.8], label="$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$")
plt.xlabel("$x_1$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)

plt.show()
```



Now let's try a gradient boosting regressor:

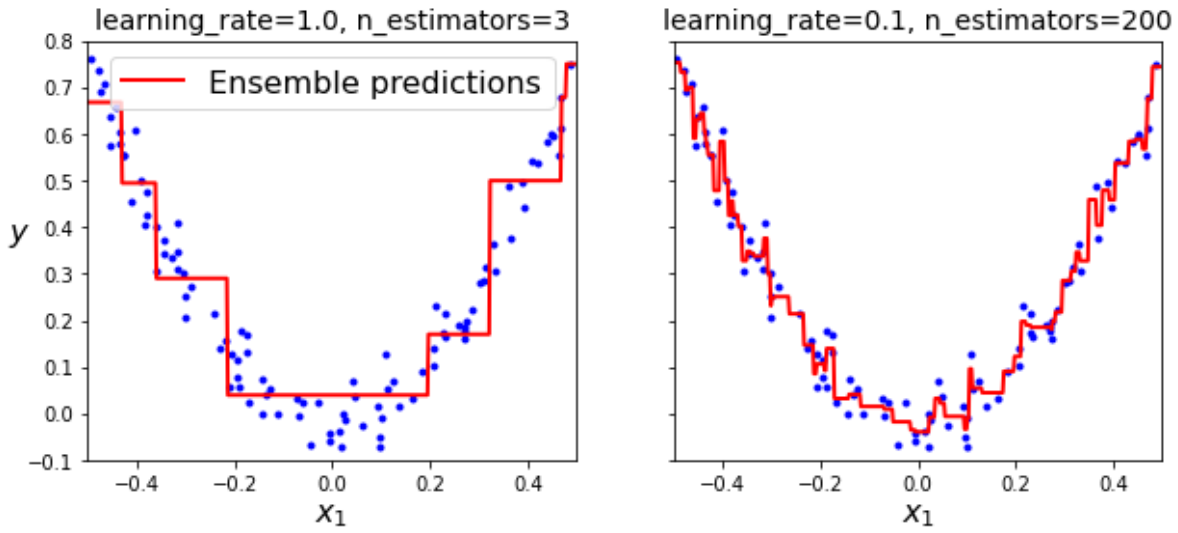
0]:
```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0, random_state=42)
gbrt.fit(X, y)
```

0]:
```
GradientBoostingRegressor(learning_rate=1.0, max_depth=2, n_estimators=3,
                          random_state=42)
```

```
gbrt_slow = GradientBoostingRegressor(max_depth=2, n_estimators=200, learning_rate=0.1, random_state=42)
gbrt_slow.fit(X, y)
```

```
GradientBoostingRegressor(max_depth=2, n_estimators=200, random_state=42)
```

```
fix, axes = plt.subplots(ncols=2, figsize=(10,4), sharey=True)

plt.sca(axes[0])
plot_predictions([gbrt], X, y, axes=[-0.5, 0.5, -0.1, 0.8], label="Ensemble predictions")
plt.title("learning_rate={}, n_estimators={}".format(gbrt.learning_rate, gbrt.n_estimators), fontsize=14)
plt.xlabel("$x_1$", fontsize=16)
plt.ylabel("$y$", fontsize=16, rotation=0)

plt.sca(axes[1])
plot_predictions([gbrt_slow], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("learning_rate={}, n_estimators={}".format(gbrt_slow.learning_rate, gbrt_slow.n_estimators), fontsize=14)
plt.xlabel("$x_1$", fontsize=16)

plt.show()
```



Gradient Boosting with Early stopping:

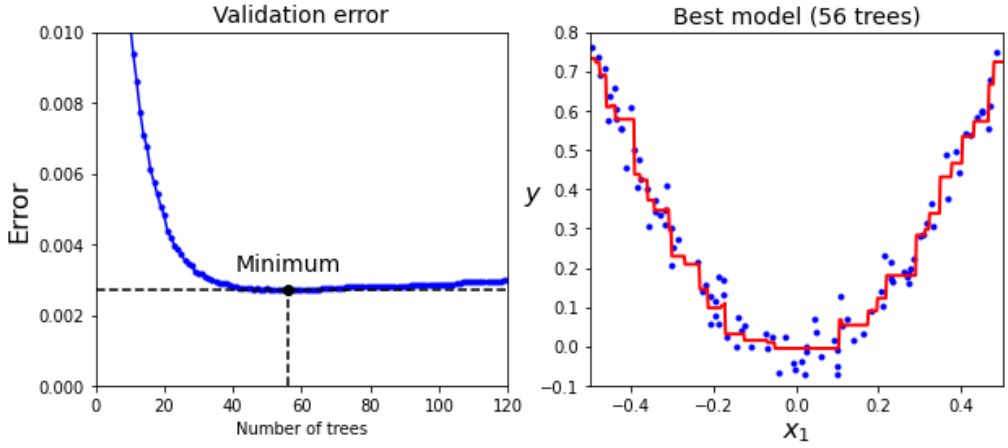
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_val, y_train, y_val = train_test_split(X, y, random_state=49)

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=120, random_state=42)
gbrt.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred)
          for y_pred in gbrt.staged_predict(X_val)]
bst_n_estimators = np.argmin(errors) + 1

gbrt_best = GradientBoostingRegressor(max_depth=2, n_estimators=bst_n_estimators, random_state=42)
gbrt_best.fit(X_train, y_train)
```

```
▾                          GradientBoostingRegressor
GradientBoostingRegressor(max_depth=2, n_estimators=56, random_state=42)
```

Tuning the number of trees using early stopping:

```
min_error = np.min(errors)
```

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

plt.subplot(121)
plt.plot(np.arange(1, len(errors) + 1), errors, "b.-")
plt.plot([bst_n_estimators, bst_n_estimators], [0, min_error], "k--")
plt.plot([0, 120], [min_error, min_error], "k--")
plt.plot(bst_n_estimators, min_error, "ko")
plt.text(bst_n_estimators, min_error*1.2, "Minimum", ha="center", fontsize=14)
plt.axis([0, 120, 0, 0.01])
plt.xlabel("Number of trees")
plt.ylabel("Error", fontsize=16)
plt.title("Validation error", fontsize=14)

plt.subplot(122)
plot_predictions([gbrt_best], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("Best model (%d trees)" % bst_n_estimators, fontsize=14)
plt.ylabel("$y$", fontsize=16, rotation=0)
plt.xlabel("$x_1$", fontsize=16)

plt.show()
```



Early stopping with some patience (interrupts training only after there's no improvement for 5 epochs):

```
brt = GradientBoostingRegressor(max_depth=2, warm_start=True, random_state=42)

min_val_error = float("inf")
error_going_up = 0
for n_estimators in range(1, 120):
    gbrt.n_estimators = n_estimators
    gbrt.fit(X_train, y_train)
    y_pred = gbrt.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    if val_error < min_val_error:
        min_val_error = val_error
        error_going_up = 0
    else:
        error_going_up += 1
        if error_going_up == 5:
            break  # early stopping
```

```
print(gbrt.n_estimators)
```

```
61
```

```
print("Minimum validation MSE:", min_val_error)
```

```
Minimum validation MSE: 0.002712853325235463
```

Experiment: 10

Aim: To Perform Simple ANN on Iris Dataset

Theory:

The perceptron is one of the simples ANN architectures, invented in 1957 by Frank Rosenblatt. It is based on a slightly different artificial neuron called a threshold logic unit (TLU), or sometimes a linear threshold unit (TLU) : the inputs and output are now numbers (instead of binary on/off values) and each input connection is associated with a weight. The TLU computes a weighted sum of its inputs (z = w1 x1 + w2 x2 + ⋯ + w n xn = x T w), then applies a step function to that sum and outputs the result: h w(x) = step(z), where z = x T w.

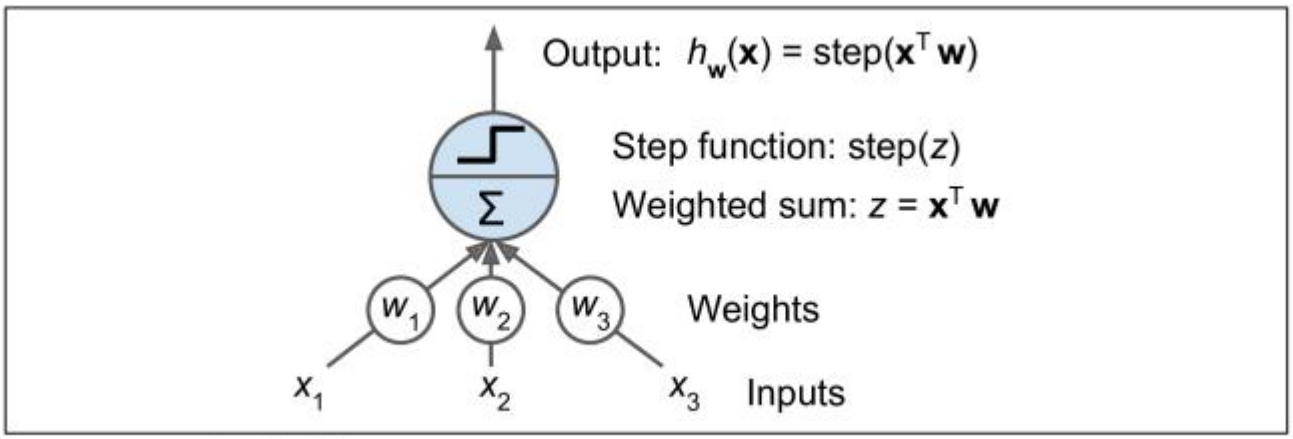


Figure 10-4. Threshold logic unit

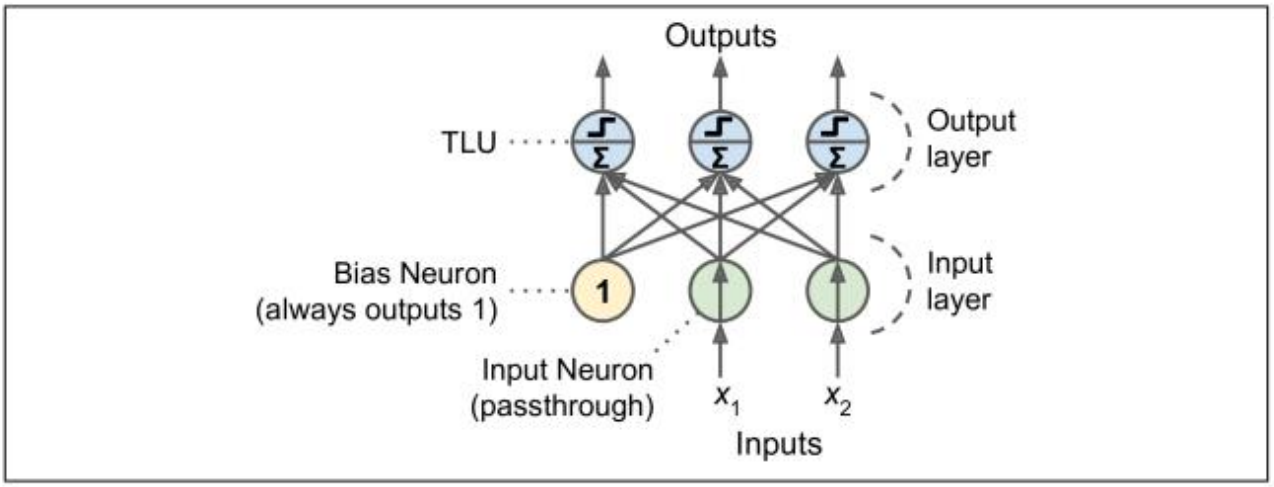


Figure 10-5. Perceptron diagram

Dataset Description:

.. _iris_dataset:

Iris plants dataset

Data Set Characteristics:

: Number of Instances: 150 (50 in each of three classes)
: Number of Attributes: 4 numeric, predictive attributes and the class
: Attribute Information:
- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
    - Iris-Setosa
    - Iris-Versicolour
    - Iris-Virginica

Code:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:,(2,3)] # petal lenght, petal width
y = (iris.target==0).astype('int')

per_clf = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
per_clf.fit(X,y)
```

```
Perceptron(random_state=42)
```

```
y_pred=per_clf.predict(X)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y,y_pred)
```

```
1.0
```

```
from sklearn.metrics import confusion_matrix,plot_confusion_matrix
cnf = confusion_matrix(y,y_pred)
print(cnf)
plot_confusion_matrix(per_clf,X,y)
```

```
[[100   0]
 [  0  50]]
```

: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e6bc0b7160>