Practical 1

Aim:- Python ecosystem for machine learning python,sci py, sci kit learn

Sol.

Python

Dating from 1991, the Python programming language was considered a gap-filler, a way to write scripts that “automate the boring stuff” (as one popular book on learning Python put it) or to rapidly prototype applications that will be implemented in other languages.
However, over the past few years, Python has emerged as a first-class citizen in modern software development, infrastructure management, and data analysis. It is no longer a back-room utility language, but a major force in web application creation and systems management, and a key driver of the explosion in big data analytics and machine intelligence.
Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often dePython is a highlevel, general-purpose programming language. Its design philosophy emphasizes scribed as a "batteries included" language due to its comprehensive standard library
Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020. Python consistently ranks as one of the most popular programming languages.

Python is an excellent language for data visualization for a few reasons:
1. It has a wide range of libraries that make it easy to work with data.
2. It provides powerful tools for data analysis.
3. Its syntax is concise and straightforward, making it easy to learn and use.
4. It is well-suited for rapid prototyping.

SciPy

What is SciPy?

SciPy is a scientific computation library that uses NumPy underneath.
SciPy stands for Scientific Python.
It provides more utility functions for optimization, stats and signal processing.
Like NumPy, SciPy is open source so we can use it freely.
SciPy was created by NumPy's creator Travis Olliphant.

Why Use SciPy?

If SciPy uses NumPy underneath, why can we not just use NumPy? SciPy has optimized and added functions that are frequently used in NumPy and Data Science.

Imagine you have an e-commerce website and that you are designing the algorithm to rank your products in your search page. What will be the first item that you display? The one with the best reviews? The one with the lowest price? Or a combination of both? The problem gets complicated pretty quickly.
A simple solution is to use your intuition, collect the feedback from your customers or get the metrics from your website and handcraft the perfect formula that works for you. Not very scientific isn't it? A more complex approach involves building many ranking formulas and use A/B testing to select the one with the best performance.
Here we will instead use the data from our customers to automatically learn their *preference function* such that the ranking of our search page is the one that maximise the likelihood of scoring a *conversion* (i.e. the customer buys your item). Specifically we will learn how to rank movies from the movielens open dataset based on artificially generated user data. The full steps are available on Github in a Jupyter notebook format.
SciPy is another open-source library for data processing and modeling that builds on NumPy for scientific computation applications. It contains more fully-featured versions of the linear algebra modules found in NumPy and many other numerical algorithms.
SciPy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics, and other classes of problems.
It also adds a collection of algorithms and high-level commands for manipulating and visualizing data. For instance, by combining SciPy and

NumPy, you can do things like image processing.

SciPy is an ecosystem of Python libraries for mathematics, science and engineering. It is an add-on to Python that you will need for machine learning.
The SciPy ecosystem is comprised of the following core modules relevant to machine learning:

- NumPy: A foundation for SciPy that allows you to efficiently work with data in arrays.
- Matplotlib: Allows you to create 2D charts and plots from data.
- pandas: Tools and data structures to organize and analyze your data.

To be effective at machine learning in Python you must install and become familiar with SciPy. Specifically:

- You will use Pandas to load explore and better understand your data.
- You will use Matplotlib (and wrappers of Matplotlib in other frameworks) to create plots and charts of your data.
- You will prepare your data as NumPy arrays for modeling in machine learning algorithms.

Scikit-learn

What is scikit learn?

Sci-kit (pronounced sai kit) learn is a free software machine learning library for Python. Library in computer language simply means a collection of languages, behaviors, routines, scripts, files, programs and functions which can be used or referenced in a programming code.

Machine learning itself is a method of study of algorithms that can be used by computer systems to perform a specific task or a group of tasks without getting an explicit instruction from a controller, which would be a human in this case, but rather relying on past patterns and experiences to draw inferences that could then be used to perform the required task.

Scikit-learn, also called sklearn, is a library for learning, improving, and executing machine learning models. It builds on NumPy and SciPy by adding a set of algorithms for common machine-learning and data-mining tasks.
Sklearn is the most popular Python library for performing classification, regression, and clustering algorithms. It's considered a very curated library because developers don't have to choose between different versions of the same algorithm

Sci-kit learn is therefore a collection of languages, functions and routines for python that helps provide many supervised and unsupervised learning algorithms.
Sci-kit learn is built from already existing and familiar technology like NumPy, pandas, iPhython, Sympy and Matplotlib.
The two kinds of machine learning models that exist in reality are:

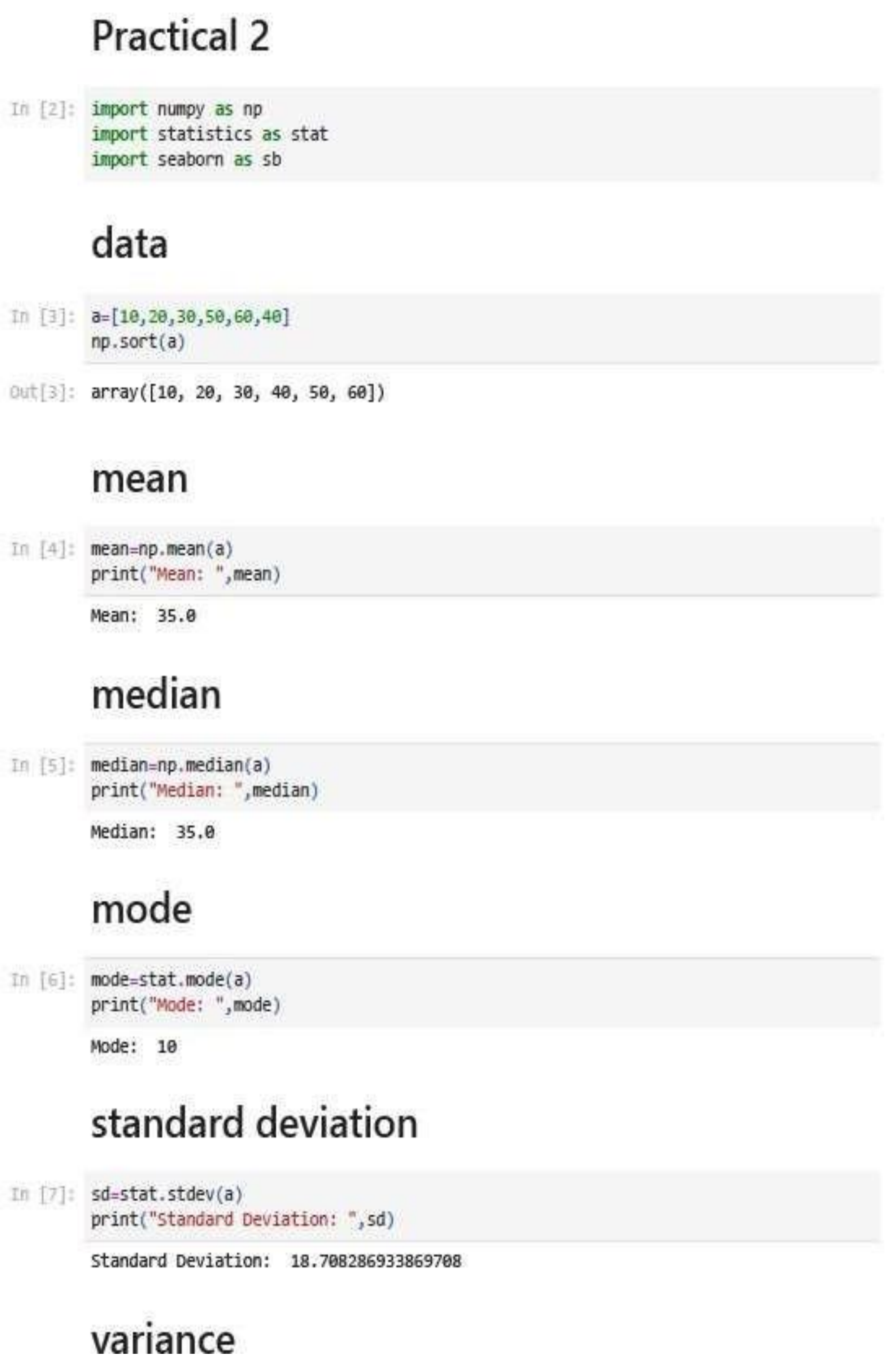
- *The traditional machine learning model*
- *Artificial neural network*

Sci-kit learn is a library in python that helps in building traditional machine learning models. There are other libraries in python that help in building artificial neural networks, like the python library called Keras.

Practical 2

Aim:- Study of preprocessing methods. Write a program to find following statistics from a given dataset. Mean, mode, median, variance, standard deviation, quartiles, interquartile range.

Sol.



Practical 2

```
In [2]: import numpy as np
        import statistics as stat
        import seaborn as sb
```

data

```
In [3]: a=[10,20,30,50,60,40]
        np.sort(a)

Out[3]: array([10, 20, 30, 40, 50, 60])
```

mean

```
In [4]: mean=np.mean(a)
        print("Mean: ",mean)

        Mean:  35.0
```

median

```
In [5]: median=np.median(a)
        print("Median: ",median)

        Median:  35.0
```

mode

```
In [6]: mode=stat.mode(a)
        print("Mode: ",mode)

        Mode:  10
```

standard deviation

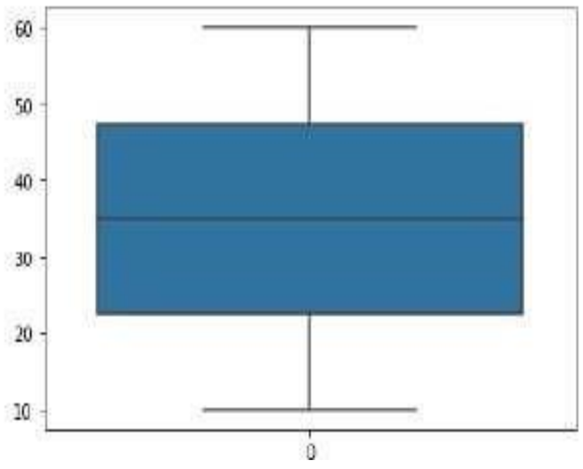
```
In [7]: sd=stat.stdev(a)
        print("Standard Deviation: ",sd)

        Standard Deviation:  18.708286933869708
```

variance

```
In [8]: var=stat.variance(a)
        print("Variance",var)

        Variance 350
```

quartile

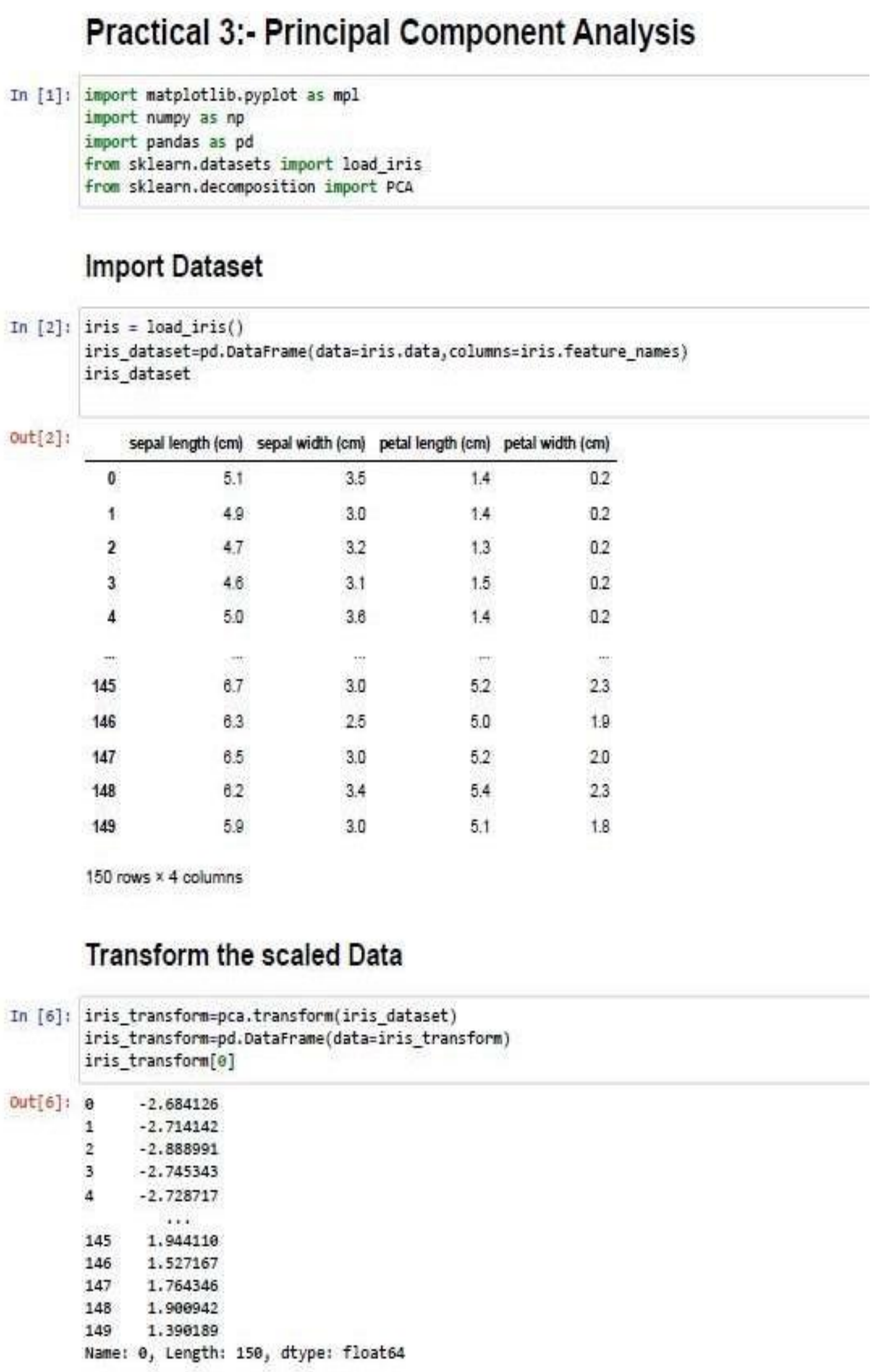
```
In [9]: q1,q3=np.percentile(a,[25,25])
        sb.boxplot(a)
        q1,q3

Out[9]: (22.5, 22.5)
```



Practical 3

Aim:- Study and implement PCA in python.

Sol.



Practical 3:- Principal Component Analysis

In [1]:
```
import matplotlib.pyplot as mpl
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
```

Import Dataset

In [2]:
```
iris = load_iris()
iris_dataset=pd.DataFrame(data=iris.data,columns=iris.feature_names)
iris_dataset
```

Out[2]:

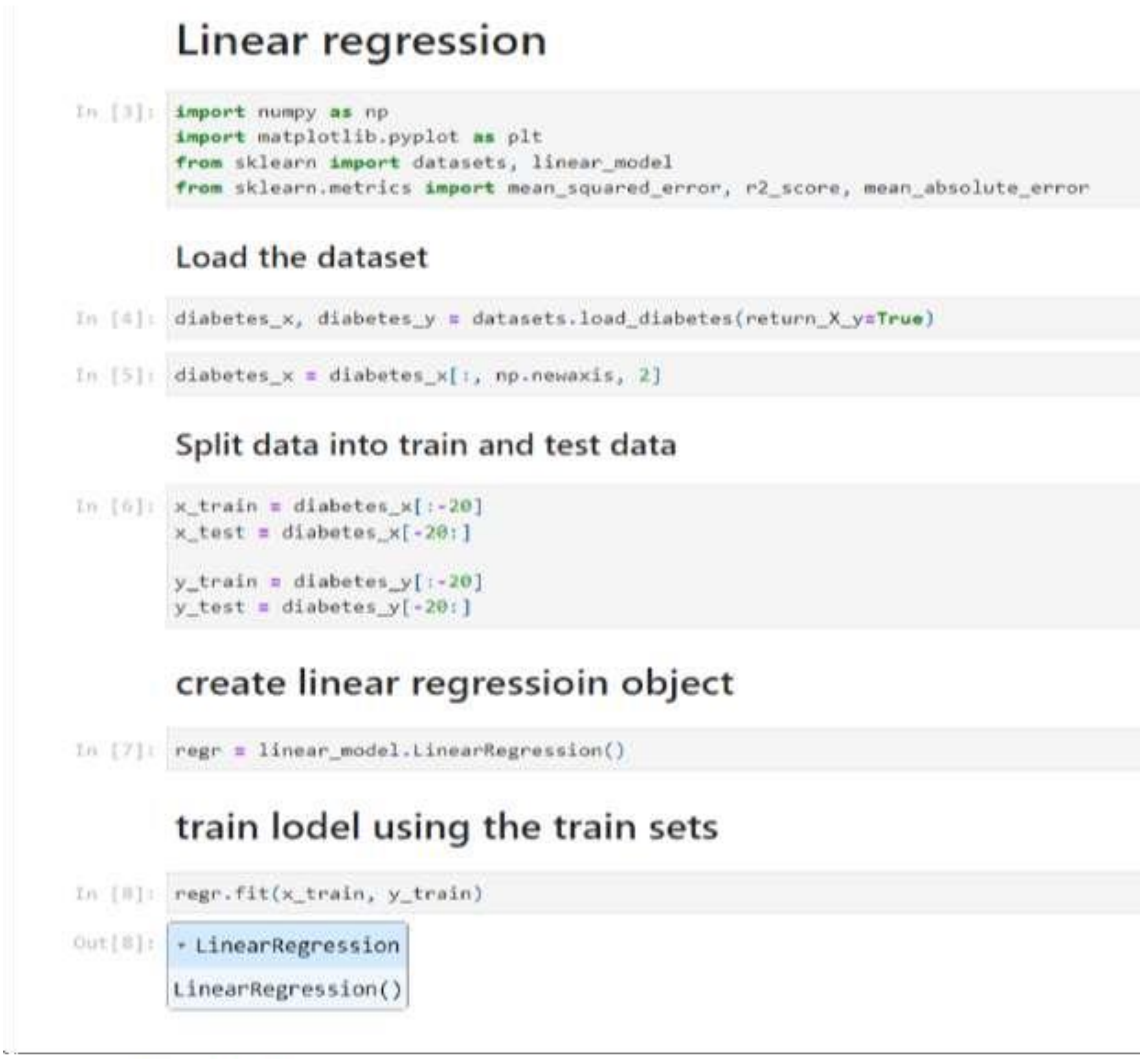
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Transform the scaled Data

In [6]:
```
iris_transform=pca.transform(iris_dataset)
iris_transform=pd.DataFrame(data=iris_transform)
iris_transform[0]
```

Out[6]:
```
0     -2.684126
1     -2.714142
2     -2.888991
3     -2.745343
4     -2.728717
         ...
145    1.944110
146    1.527167
147    1.764346
148    1.900942
149    1.390189
Name: 0, Length: 150, dtype: float64
```



Linear regression

```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import datasets, linear_model
        from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

Load the dataset

```
In [4]: diabetes_x, diabetes_y = datasets.load_diabetes(return_X_y=True)

In [5]: diabetes_x = diabetes_x[:, np.newaxis, 2]
```

Split data into train and test data

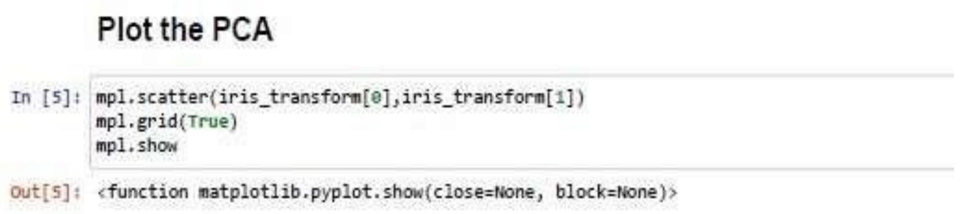
```
In [6]: x_train = diabetes_x[:-20]
        x_test = diabetes_x[-20:]

        y_train = diabetes_y[:-20]
        y_test = diabetes_y[-20:]
```

create linear regressioin object

```
In [7]: regr = linear_model.LinearRegression()
```

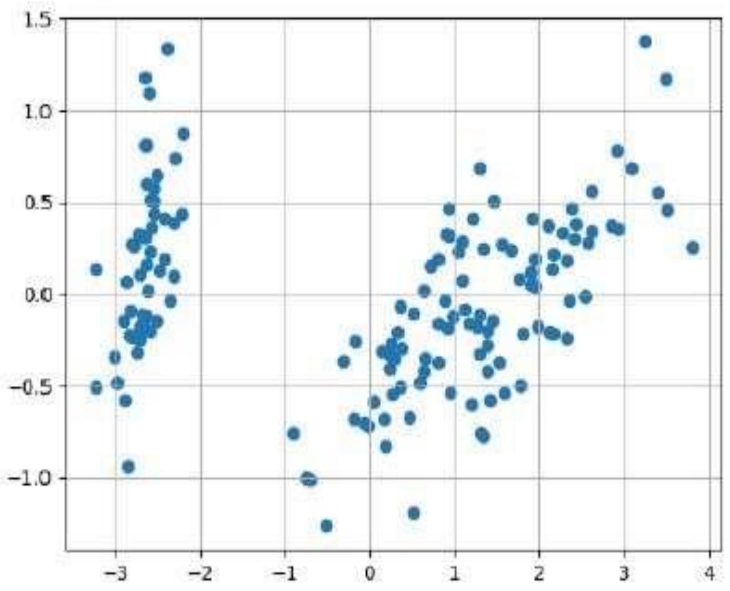
train lodel using the train sets

```
In [8]: regr.fit(x_train, y_train)

Out[8]: ▾ LinearRegression
        LinearRegression()
```



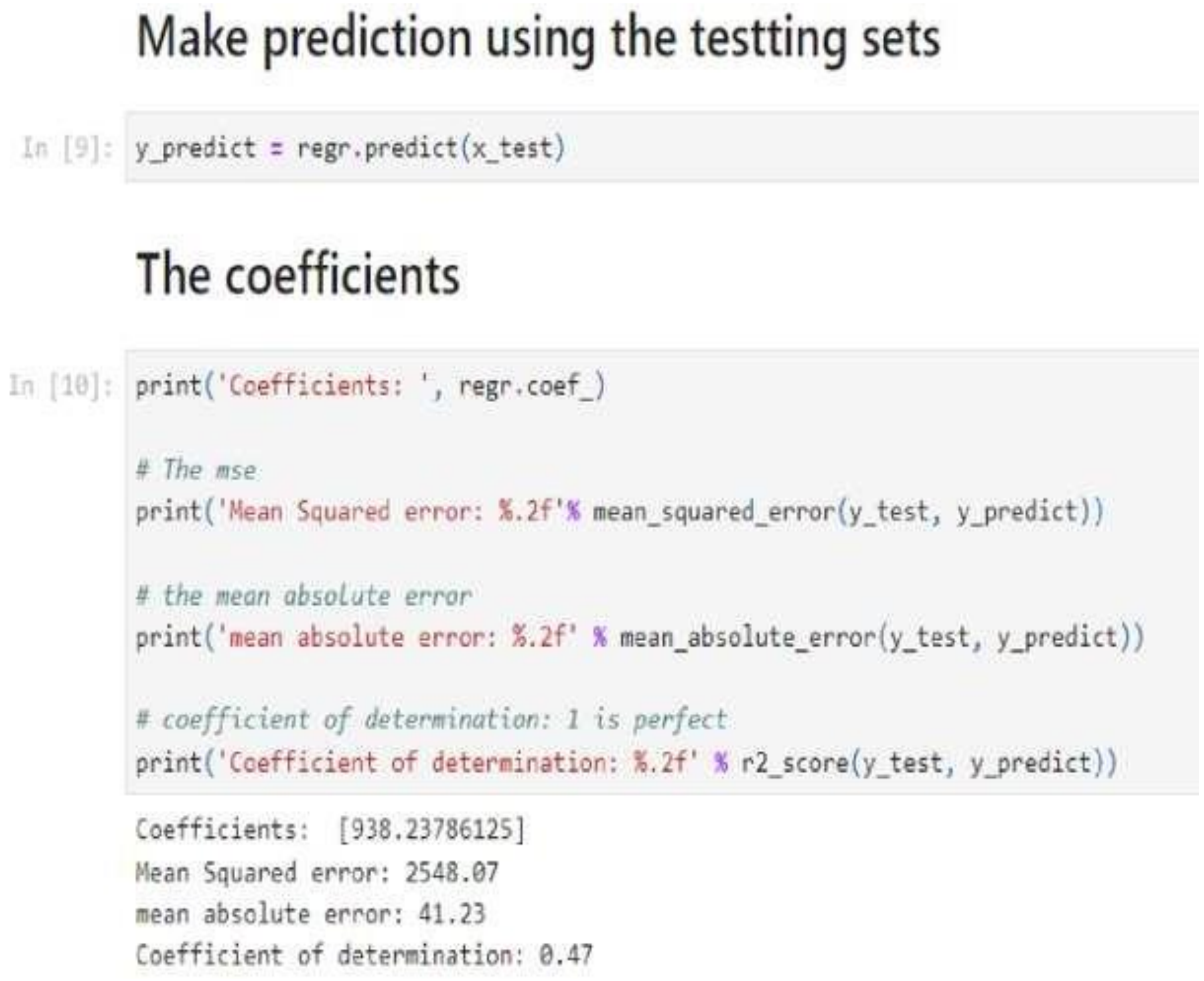
Plot the PCA

```
In [5]: mpl.scatter(iris_transform[0],iris_transform[1])
        mpl.grid(True)
        mpl.show

Out[5]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Practical 4

Aim:- Study and implement simple linear regression.
Sol.



Make prediction using the testting sets

```
In [9]: y_predict = regr.predict(x_test)
```

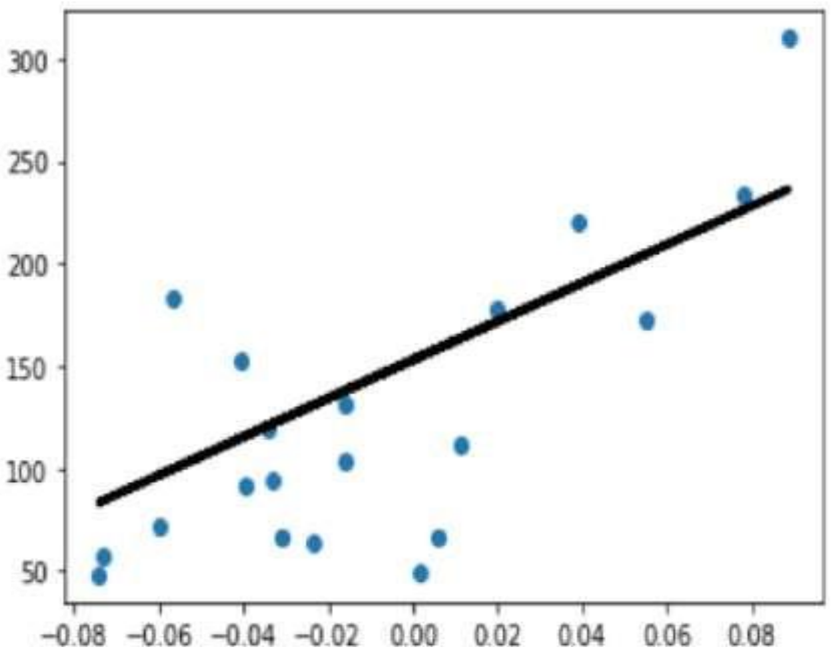
The coefficients

```
In [10]: print('Coefficients: ', regr.coef_)

         # The mse
         print('Mean Squared error: %.2f'% mean_squared_error(y_test, y_predict))

         # the mean absolute error
         print('mean absolute error: %.2f' % mean_absolute_error(y_test, y_predict))

         # coefficient of determination: 1 is perfect
         print('Coefficient of determination: %.2f' % r2_score(y_test, y_predict))
```

```
Coefficients:  [938.23786125]
Mean Squared error: 2548.07
mean absolute error: 41.23
Coefficient of determination: 0.47
```

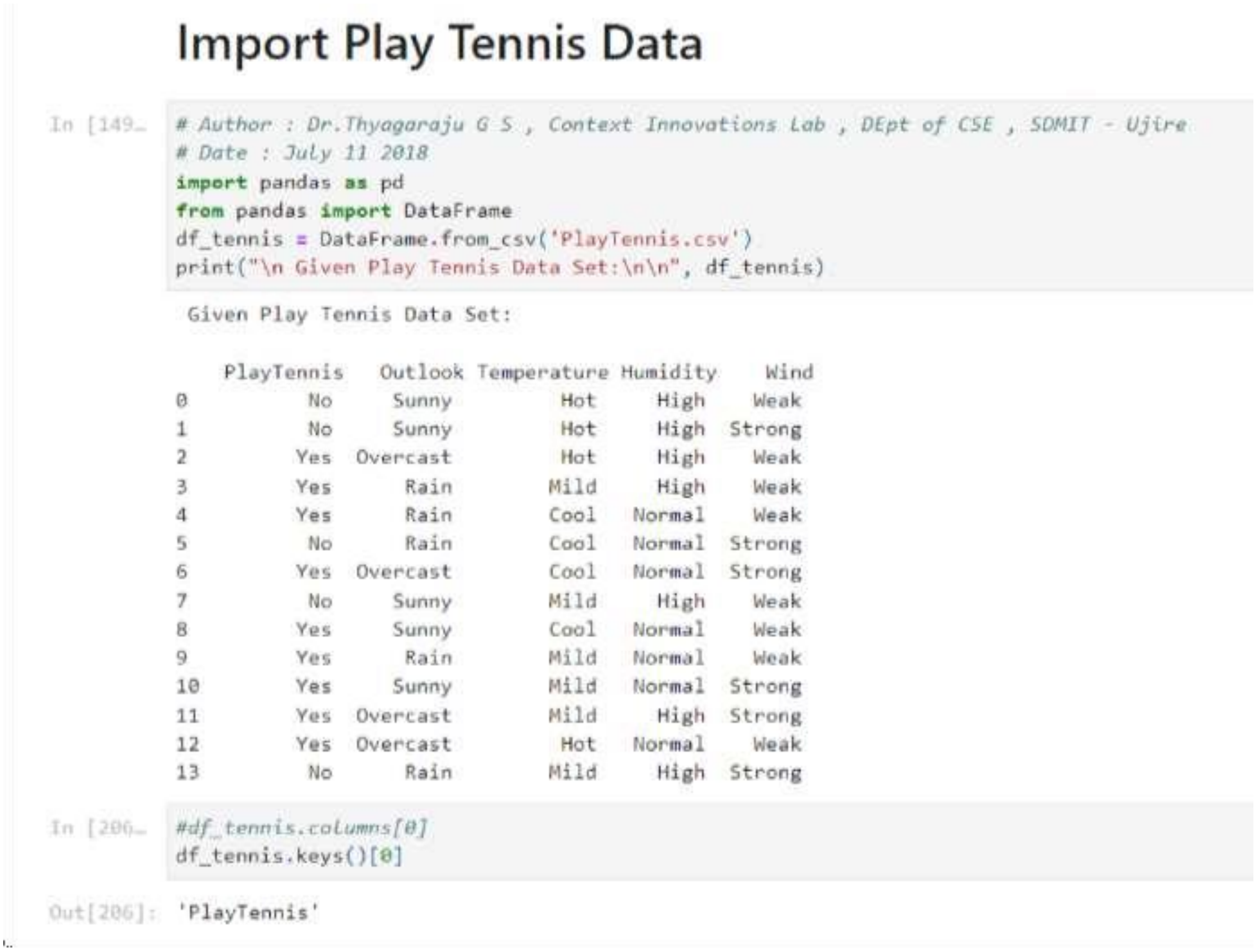
plot outputs

In [12]:
```
plt.plot(x_test, y_predict, color='black', linewidth=3)
plt.scatter(x_test, y_test)

plt.show()
```



Practical 5

Aim :- Write a program to demonstrate the working of the decision tree-based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
Sol.



Import Play Tennis Data

In [149…

```
# Author : Dr.Thyagaraju G S , Context Innovations Lab , DEpt of CSE , SDMIT - Ujire
# Date : July 11 2018
import pandas as pd
from pandas import DataFrame
df_tennis = DataFrame.from_csv('PlayTennis.csv')
print("\n Given Play Tennis Data Set:\n\n", df_tennis)
```

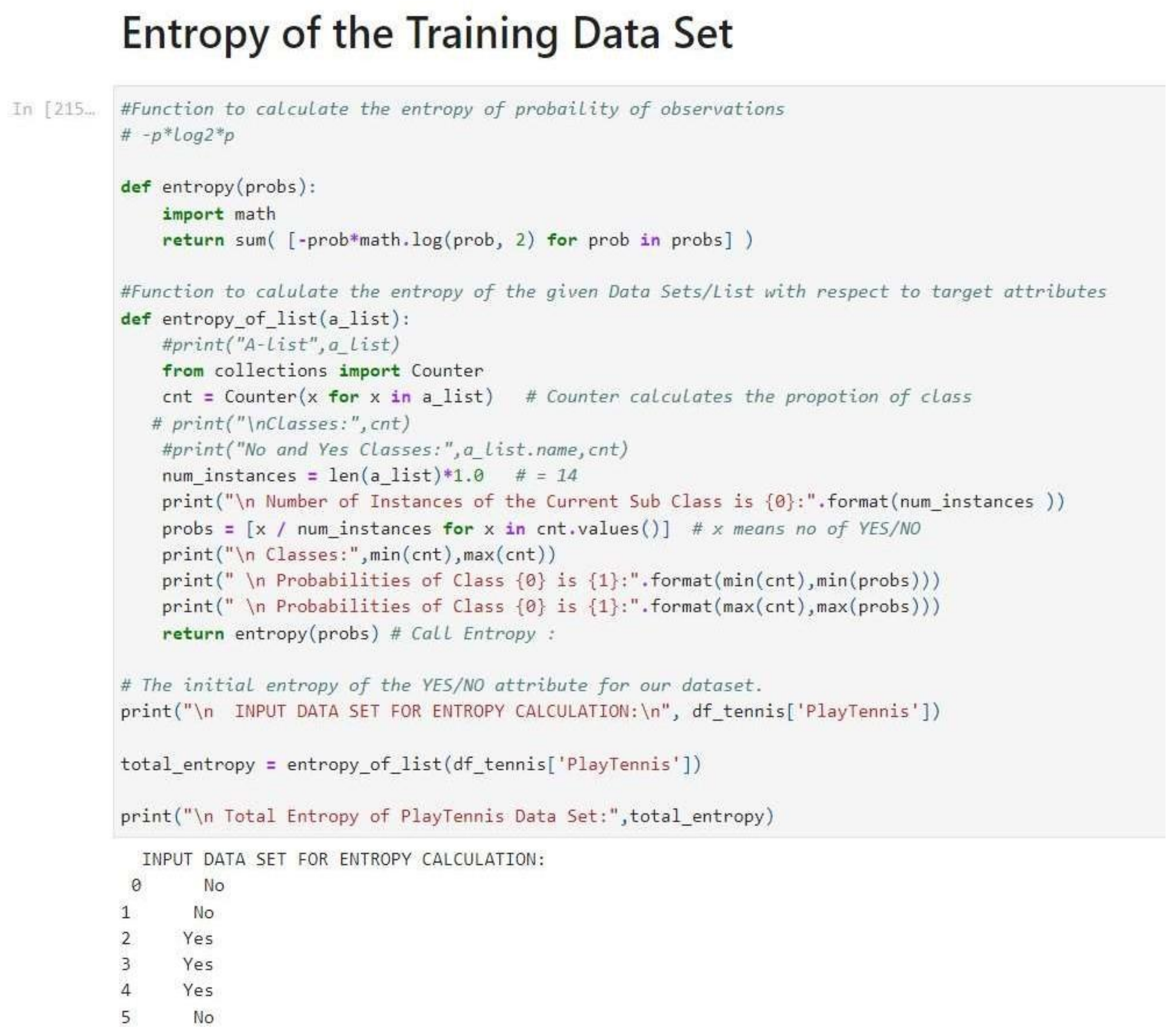
```
 Given Play Tennis Data Set:

    PlayTennis   Outlook Temperature Humidity    Wind
0          No     Sunny         Hot     High    Weak
1          No     Sunny         Hot     High  Strong
2         Yes  Overcast         Hot     High    Weak
3         Yes      Rain        Mild     High    Weak
4         Yes      Rain        Cool   Normal    Weak
5          No      Rain        Cool   Normal  Strong
6         Yes  Overcast        Cool   Normal  Strong
7          No     Sunny        Mild     High    Weak
8         Yes     Sunny        Cool   Normal    Weak
9         Yes      Rain        Mild   Normal    Weak
10        Yes     Sunny        Mild   Normal  Strong
11        Yes  Overcast        Mild     High  Strong
12        Yes  Overcast         Hot   Normal    Weak
13         No      Rain        Mild     High  Strong
```

In [206…

```
#df_tennis.columns[0]
df_tennis.keys()[0]
```

Out[206]:

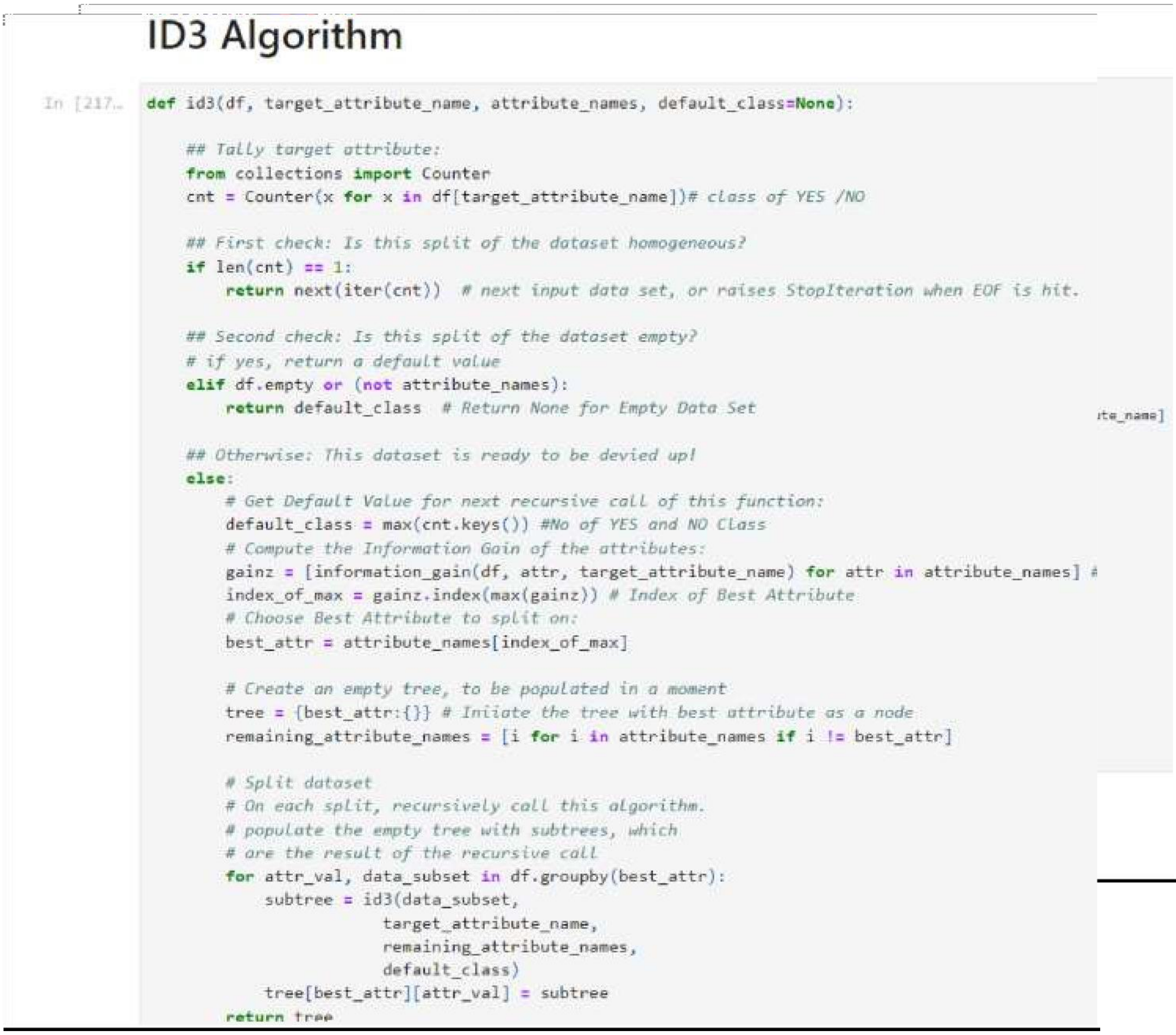
```
'PlayTennis'
```



Entropy of the Training Data Set

```
#Function to calculate the entropy of probaility of observations
# -p*log2*p

def entropy(probs):
    import math
    return sum( [-prob*math.log(prob, 2) for prob in probs] )

#Function to calulate the entropy of the given Data Sets/List with respect to target attributes
def entropy_of_list(a_list):
    #print("A-list",a_list)
    from collections import Counter
    cnt = Counter(x for x in a_list)   # Counter calculates the propotion of class
   # print("\nClasses:",cnt)
    #print("No and Yes Classes:",a_list.name,cnt)
    num_instances = len(a_list)*1.0   # = 14
    print("\n Number of Instances of the Current Sub Class is {0}:".format(num_instances ))
    probs = [x / num_instances for x in cnt.values()]  # x means no of YES/NO
    print("\n Classes:",min(cnt),max(cnt))
    print(" \n Probabilities of Class {0} is {1}:".format(min(cnt),min(probs)))
    print(" \n Probabilities of Class {0} is {1}:".format(max(cnt),max(probs)))
    return entropy(probs) # Call Entropy :

# The initial entropy of the YES/NO attribute for our dataset.
print("\n  INPUT DATA SET FOR ENTROPY CALCULATION:\n", df_tennis['PlayTennis'])

total_entropy = entropy_of_list(df_tennis['PlayTennis'])

print("\n Total Entropy of PlayTennis Data Set:",total_entropy)
```

```
  INPUT DATA SET FOR ENTROPY CALCULATION:
 0      No
1       No
2      Yes
3      Yes
4      Yes
5       No
```

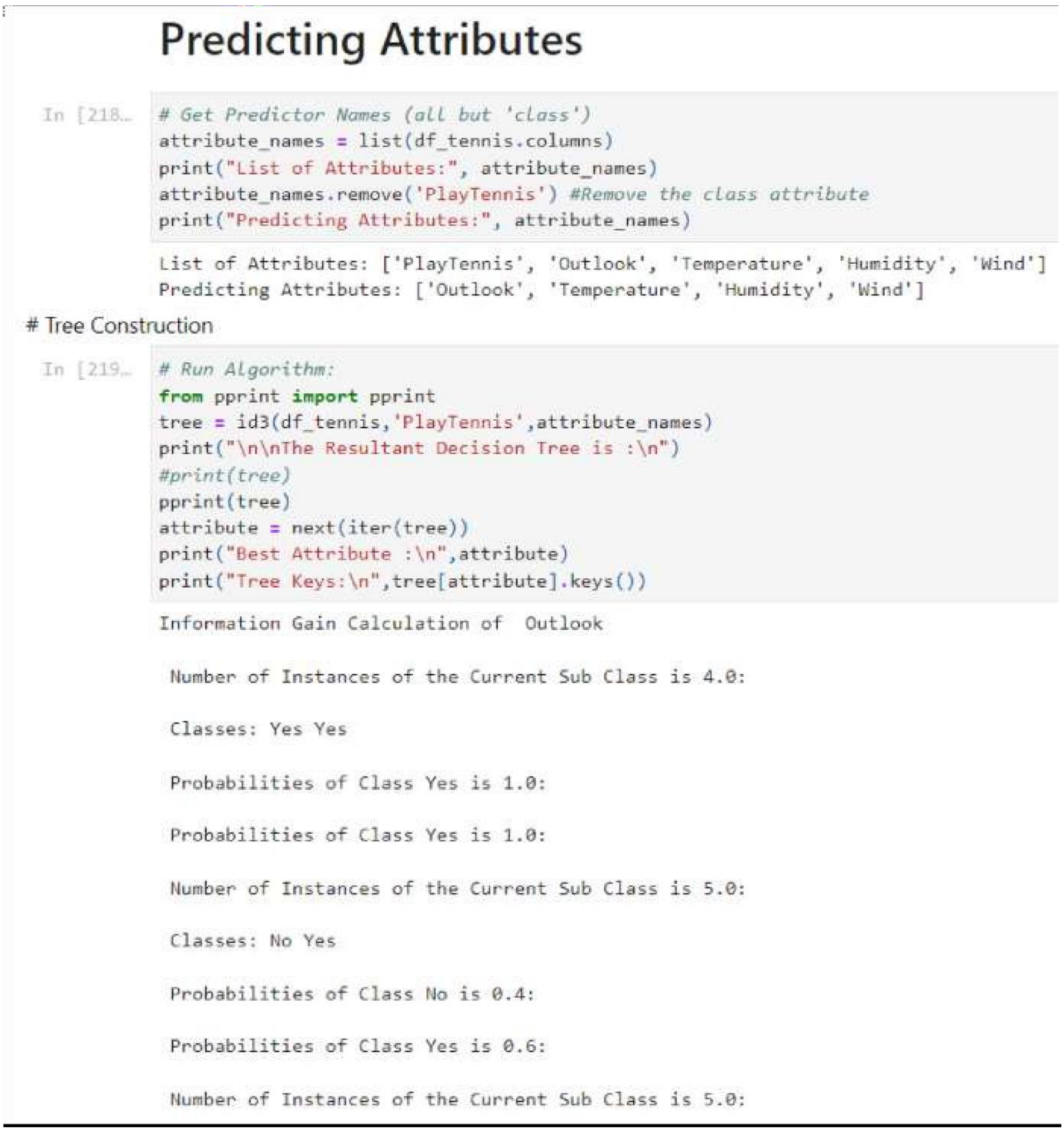


ID3 Algorithm

In [217…

```
def id3(df, target_attribute_name, attribute_names, default_class=None):

    ## Tally target attribute:
    from collections import Counter
    cnt = Counter(x for x in df[target_attribute_name])# class of YES /NO

    ## First check: Is this split of the dataset homogeneous?
    if len(cnt) == 1:
        return next(iter(cnt))  # next input data set, or raises StopIteration when EOF is hit.

    ## Second check: Is this split of the dataset empty?
    # if yes, return a default value
    elif df.empty or (not attribute_names):
        return default_class  # Return None for Empty Data Set

    ## Otherwise: This dataset is ready to be devied up!
    else:
        # Get Default Value for next recursive call of this function:
        default_class = max(cnt.keys()) #No of YES and NO Class
        # Compute the Information Gain of the attributes:
        gainz = [information_gain(df, attr, target_attribute_name) for attr in attribute_names] #
        index_of_max = gainz.index(max(gainz)) # Index of Best Attribute
        # Choose Best Attribute to split on:
        best_attr = attribute_names[index_of_max]

        # Create an empty tree, to be populated in a moment
        tree = {best_attr:{}} # Iniiate the tree with best attribute as a node
        remaining_attribute_names = [i for i in attribute_names if i != best_attr]

        # Split dataset
        # On each split, recursively call this algorithm.
        # populate the empty tree with subtrees, which
        # are the result of the recursive call
        for attr_val, data_subset in df.groupby(best_attr):
            subtree = id3(data_subset,
                        target_attribute_name,
                        remaining_attribute_names,
                        default_class)
            tree[best_attr][attr_val] = subtree
        return tree
```

ite_name]



Predicting Attributes

In [218…

```
# Get Predictor Names (all but 'class')
attribute_names = list(df_tennis.columns)
print("List of Attributes:", attribute_names)
attribute_names.remove('PlayTennis') #Remove the class attribute
print("Predicting Attributes:", attribute_names)
```

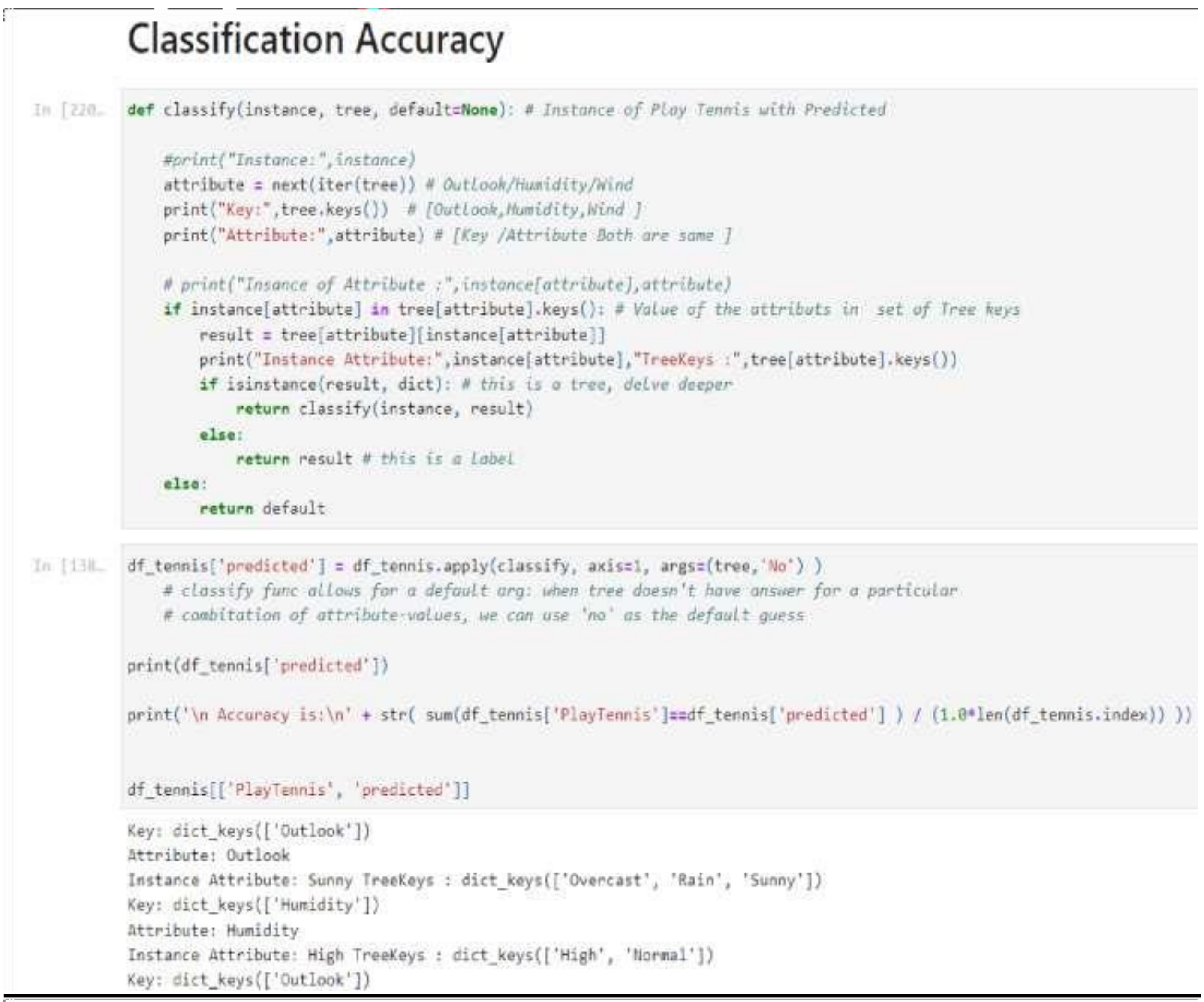
```
List of Attributes: ['PlayTennis', 'Outlook', 'Temperature', 'Humidity', 'Wind']
Predicting Attributes: ['Outlook', 'Temperature', 'Humidity', 'Wind']
```

Tree Construction

In [219…

```
# Run Algorithm:
from pprint import pprint
tree = id3(df_tennis,'PlayTennis',attribute_names)
print("\n\nThe Resultant Decision Tree is :\n")
#print(tree)
pprint(tree)
attribute = next(iter(tree))
print("Best Attribute :\n",attribute)
print("Tree Keys:\n",tree[attribute].keys())
```

```
Information Gain Calculation of  Outlook

 Number of Instances of the Current Sub Class is 4.0:

 Classes: Yes Yes

 Probabilities of Class Yes is 1.0:

 Probabilities of Class Yes is 1.0:

 Number of Instances of the Current Sub Class is 5.0:

 Classes: No Yes

 Probabilities of Class No is 0.4:

 Probabilities of Class Yes is 0.6:

 Number of Instances of the Current Sub Class is 5.0:
```

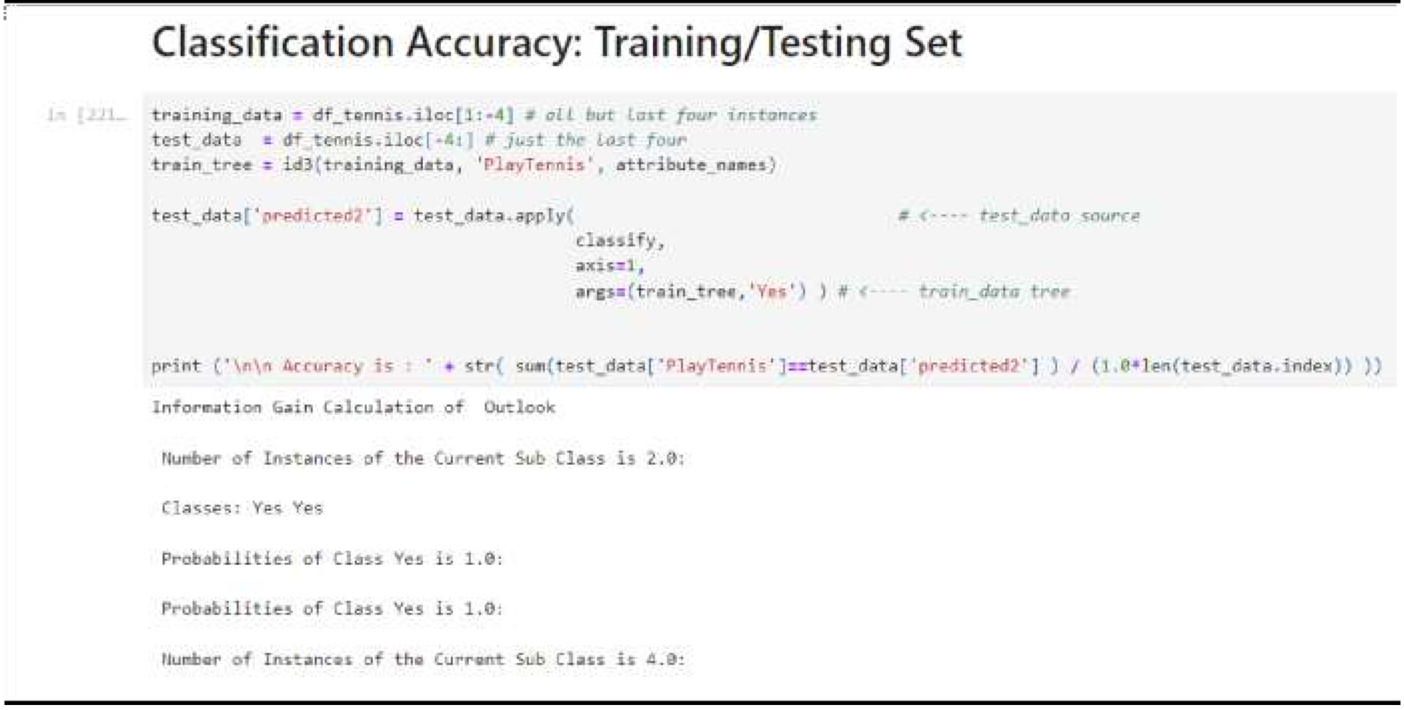


Classification Accuracy

```
In [220…
def classify(instance, tree, default=None): # Instance of Play Tennis with Predicted

    #print("Instance:",instance)
    attribute = next(iter(tree)) # Outlook/Humidity/Wind
    print("Key:",tree.keys())  # [Outlook,Humidity,Wind ]
    print("Attribute:",attribute) # [Key /Attribute Both are same ]

    # print("Insance of Attribute :",instance[attribute],attribute)
    if instance[attribute] in tree[attribute].keys(): # Value of the attributs in  set of Tree keys
        result = tree[attribute][instance[attribute]]
        print("Instance Attribute:",instance[attribute],"TreeKeys :",tree[attribute].keys())
        if isinstance(result, dict): # this is a tree, delve deeper
            return classify(instance, result)
        else:
            return result # this is a label
    else:
        return default
```

```
In [138…
df_tennis['predicted'] = df_tennis.apply(classify, axis=1, args=(tree,'No') )
    # classify func allows for a default arg: when tree doesn't have answer for a particular
    # combitation of attribute-values, we can use 'no' as the default guess

print(df_tennis['predicted'])

print('\n Accuracy is:\n' + str( sum(df_tennis['PlayTennis']==df_tennis['predicted'] ) / (1.0*len(df_tennis.index)) ))


df_tennis[['PlayTennis', 'predicted']]
```

```
Key: dict_keys(['Outlook'])
Attribute: Outlook
Instance Attribute: Sunny TreeKeys : dict_keys(['Overcast', 'Rain', 'Sunny'])
Key: dict_keys(['Humidity'])
Attribute: Humidity
Instance Attribute: High TreeKeys : dict_keys(['High', 'Normal'])
Key: dict_keys(['Outlook'])
```



Classification Accuracy: Training/Testing Set

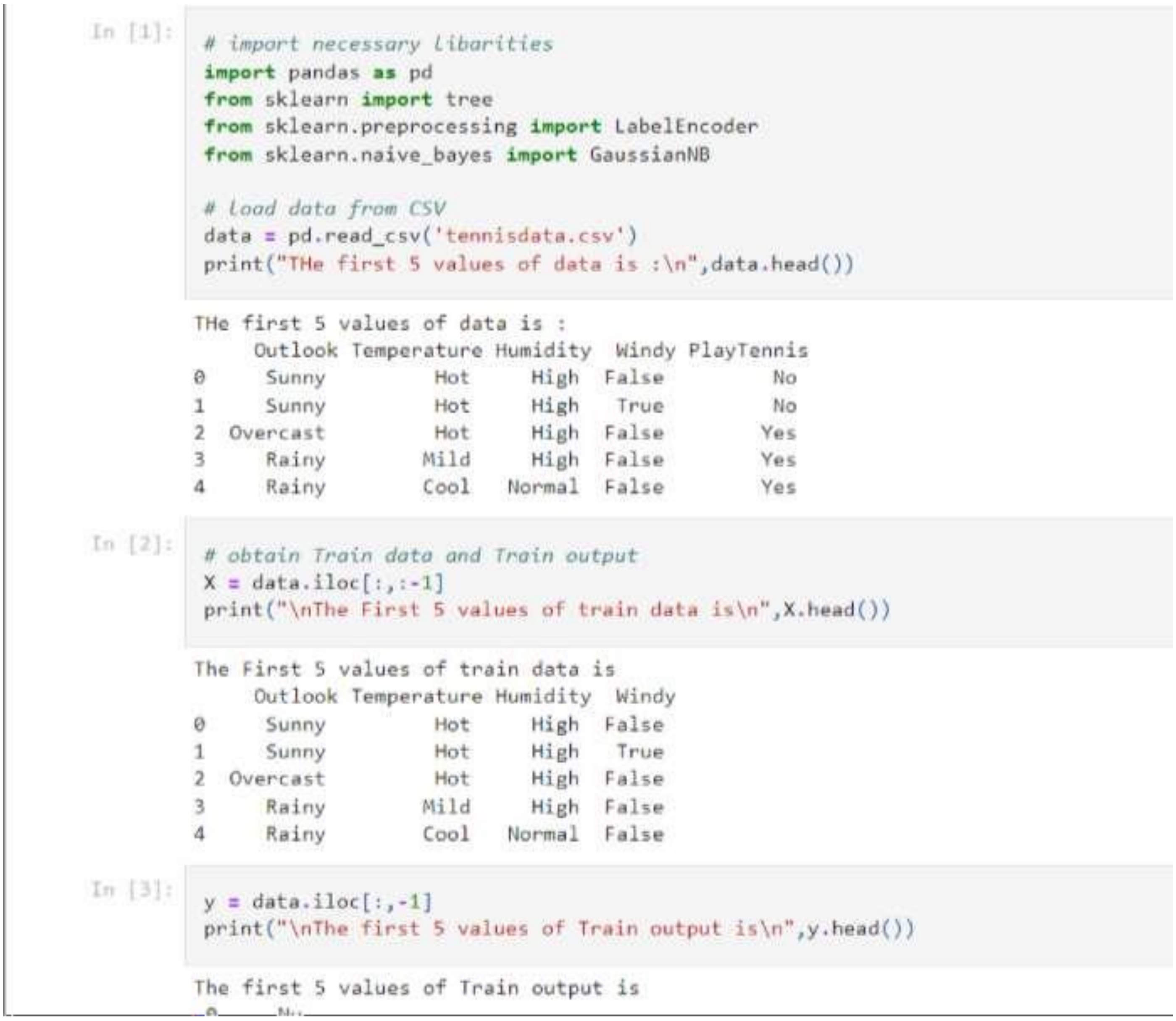
```
In [221…
training_data = df_tennis.iloc[1:-4] # all but last four instances
test_data  = df_tennis.iloc[-4:] # just the last four
train_tree = id3(training_data, 'PlayTennis', attribute_names)

test_data['predicted2'] = test_data.apply(                          # <---- test_data source
                                          classify,
                                          axis=1,
                                          args=(train_tree,'Yes') ) # <---- train_data tree


print ('\n\n Accuracy is : ' + str( sum(test_data['PlayTennis']==test_data['predicted2'] ) / (1.0*len(test_data.index)) ))
```

```
Information Gain Calculation of  Outlook

 Number of Instances of the Current Sub Class is 2.0:

 Classes: Yes Yes

 Probabilities of Class Yes is 1.0:

 Probabilities of Class Yes is 1.0:

 Number of Instances of the Current Sub Class is 4.0:
```

Practical 6

Aim :- Write a program to implement the Naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

Sol.



In [1]:
```python
# import necessary libarities
import pandas as pd
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB

# load data from CSV
data = pd.read_csv('tennisdata.csv')
print("THe first 5 values of data is :\n",data.head())
```

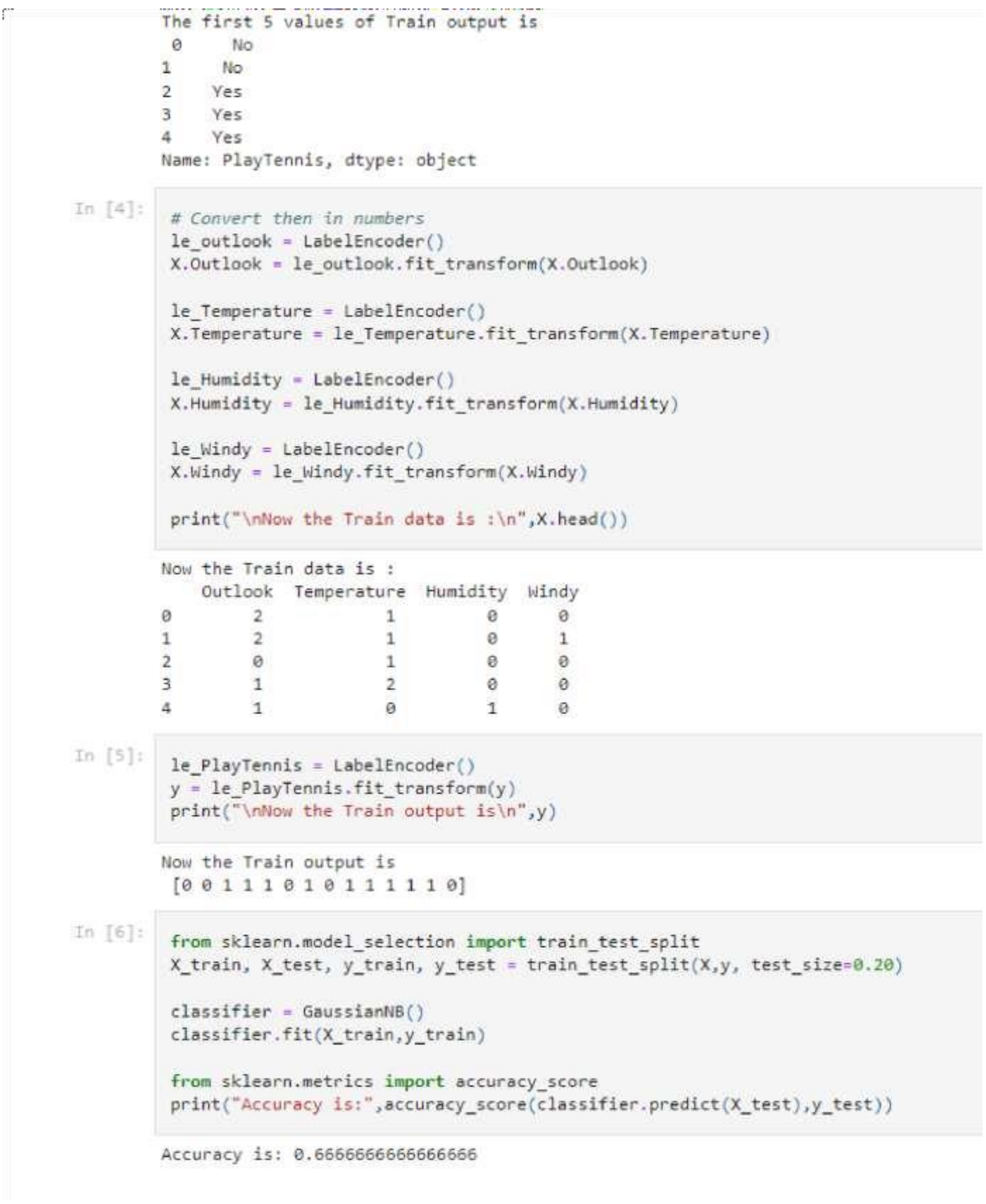
```
THe first 5 values of data is :
      Outlook Temperature Humidity  Windy PlayTennis
0      Sunny         Hot     High  False         No
1      Sunny         Hot     High   True         No
2   Overcast         Hot     High  False        Yes
3      Rainy        Mild     High  False        Yes
4      Rainy        Cool   Normal  False        Yes
```

In [2]:
```python
# obtain Train data and Train output
X = data.iloc[:,:-1]
print("\nThe First 5 values of train data is\n",X.head())
```

```
The First 5 values of train data is
      Outlook Temperature Humidity  Windy
0      Sunny         Hot     High  False
1      Sunny         Hot     High   True
2   Overcast         Hot     High  False
3      Rainy        Mild     High  False
4      Rainy        Cool   Normal  False
```

In [3]:
```python
y = data.iloc[:,-1]
print("\nThe first 5 values of Train output is\n",y.head())
```

```
The first 5 values of Train output is
```



```
The first 5 values of Train output is
 0     No
1     No
2    Yes
3    Yes
4    Yes
Name: PlayTennis, dtype: object
```

In [4]:

```
# Convert then in numbers
le_outlook = LabelEncoder()
X.Outlook = le_outlook.fit_transform(X.Outlook)

le_Temperature = LabelEncoder()
X.Temperature = le_Temperature.fit_transform(X.Temperature)

le_Humidity = LabelEncoder()
X.Humidity = le_Humidity.fit_transform(X.Humidity)

le_Windy = LabelEncoder()
X.Windy = le_Windy.fit_transform(X.Windy)

print("\nNow the Train data is :\n",X.head())
```

```
Now the Train data is :
    Outlook  Temperature  Humidity  Windy
0        2            1         0      0
1        2            1         0      1
2        0            1         0      0
3        1            2         0      0
4        1            0         1      0
```

In [5]:

```
le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)
print("\nNow the Train output is\n",y)
```

```
Now the Train output is
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

In [6]:

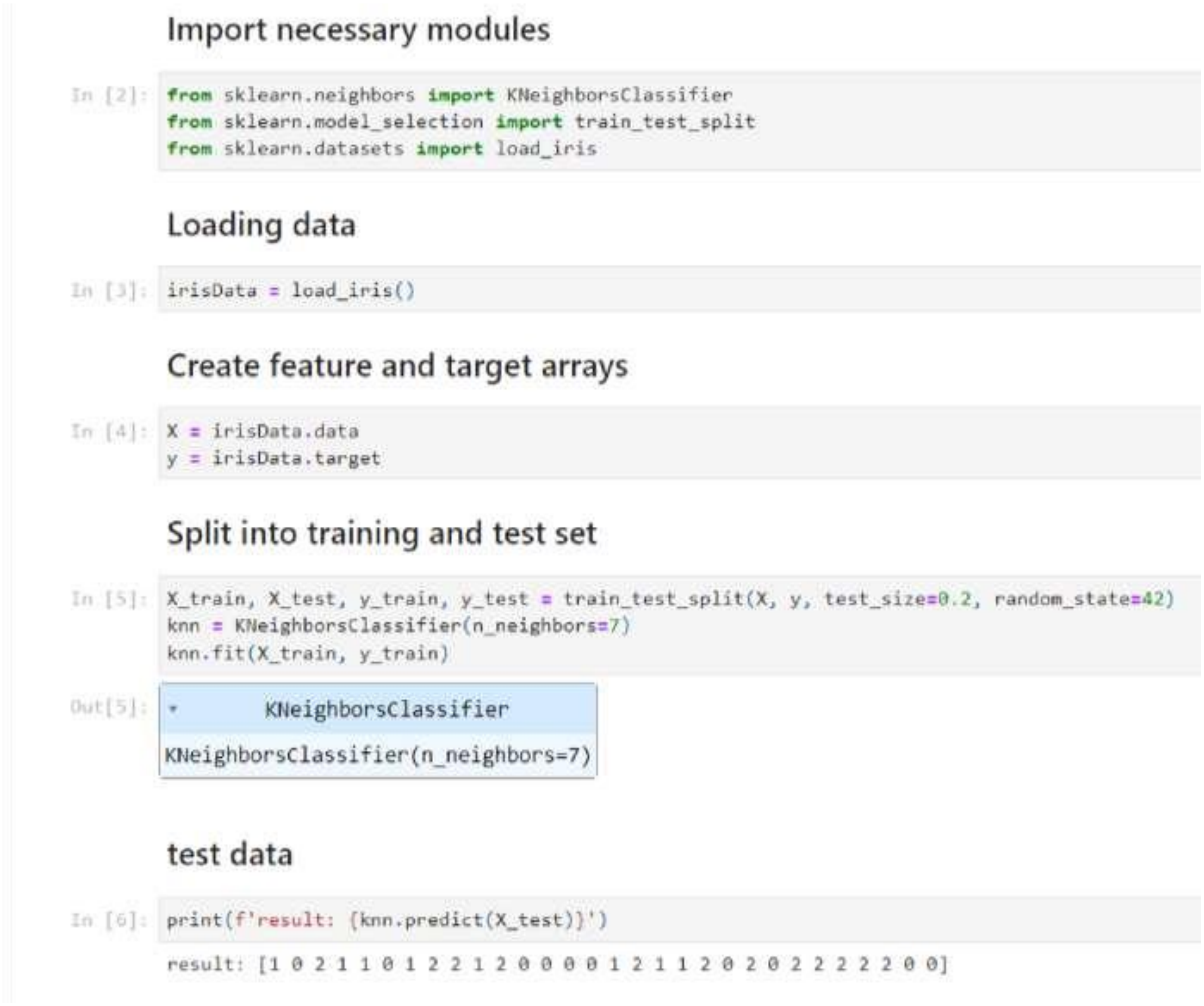
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.20)

classifier = GaussianNB()
classifier.fit(X_train,y_train)

from sklearn.metrics import accuracy_score
print("Accuracy is:",accuracy_score(classifier.predict(X_test),y_test))
```

```
Accuracy is: 0.6666666666666666
```

Practical 7

Aim :- Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Compute the accuracy of the classifier, considering few test data sets.

Sol.



Import necessary modules

```
In [2]: from sklearn.neighbors import KNeighborsClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.datasets import load_iris
```

Loading data

```
In [3]: irisData = load_iris()
```

Create feature and target arrays

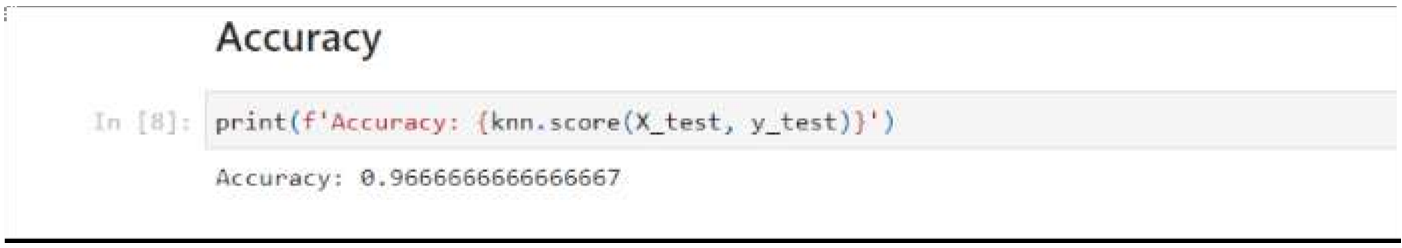
```
In [4]: X = irisData.data
        y = irisData.target
```

Split into training and test set

```
In [5]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        knn = KNeighborsClassifier(n_neighbors=7)
        knn.fit(X_train, y_train)

Out[5]:        KNeighborsClassifier
        KNeighborsClassifier(n_neighbors=7)
```

test data

```
In [6]: print(f'result: {knn.predict(X_test)}')

        result: [1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```



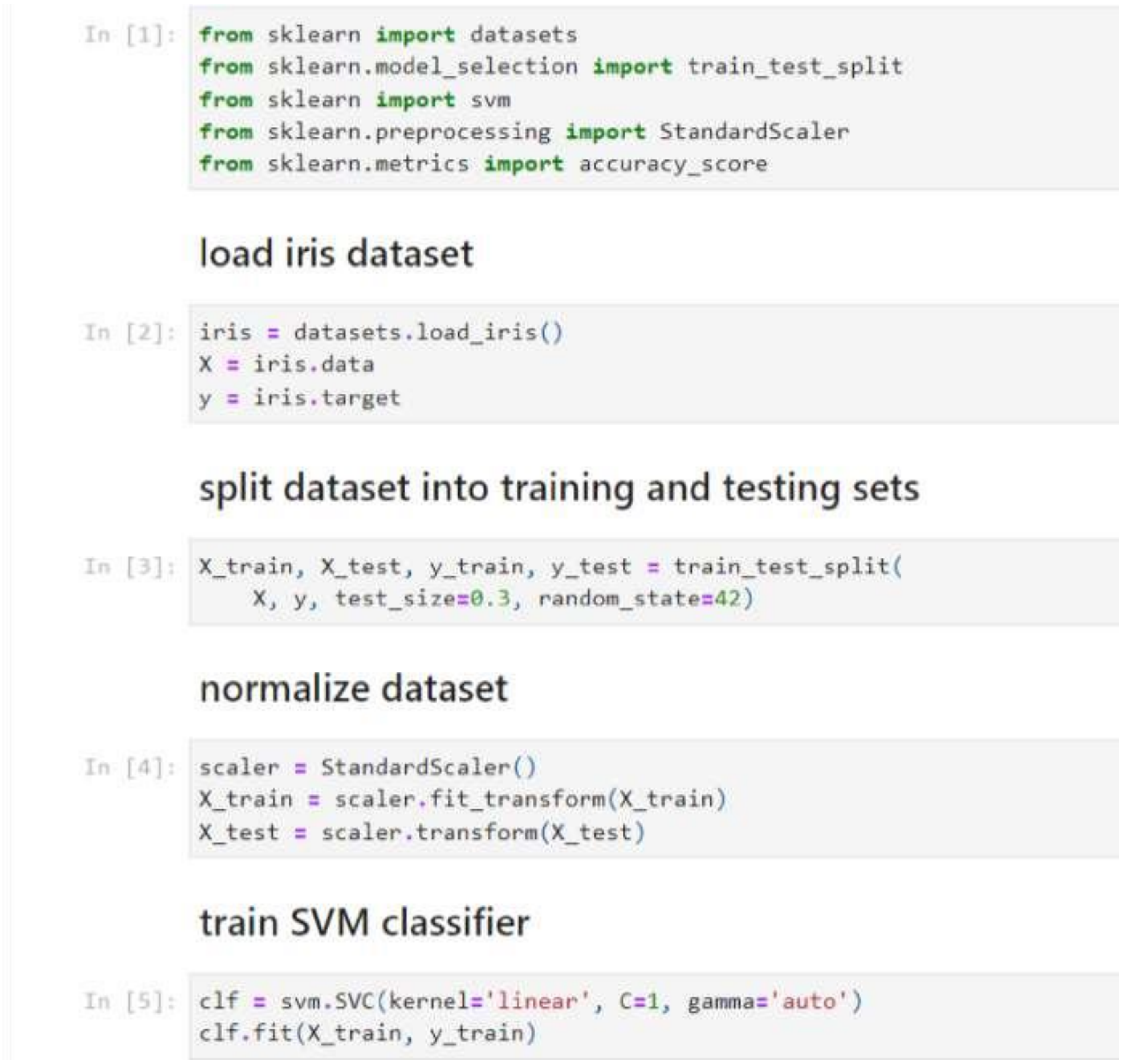
Accuracy

```
In [8]: print(f'Accuracy: {knn.score(X_test, y_test)}')

        Accuracy: 0.9666666666666667
```

Practical 8

Aim :- Write a program to implement SVM algorithm to classify the iris data set. Compute the accuracy of the classifier, considering few test data sets.

Sol.



```
In [1]: from sklearn import datasets
        from sklearn.model_selection import train_test_split
        from sklearn import svm
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import accuracy_score
```

load iris dataset

```
In [2]: iris = datasets.load_iris()
        X = iris.data
        y = iris.target
```

split dataset into training and testing sets

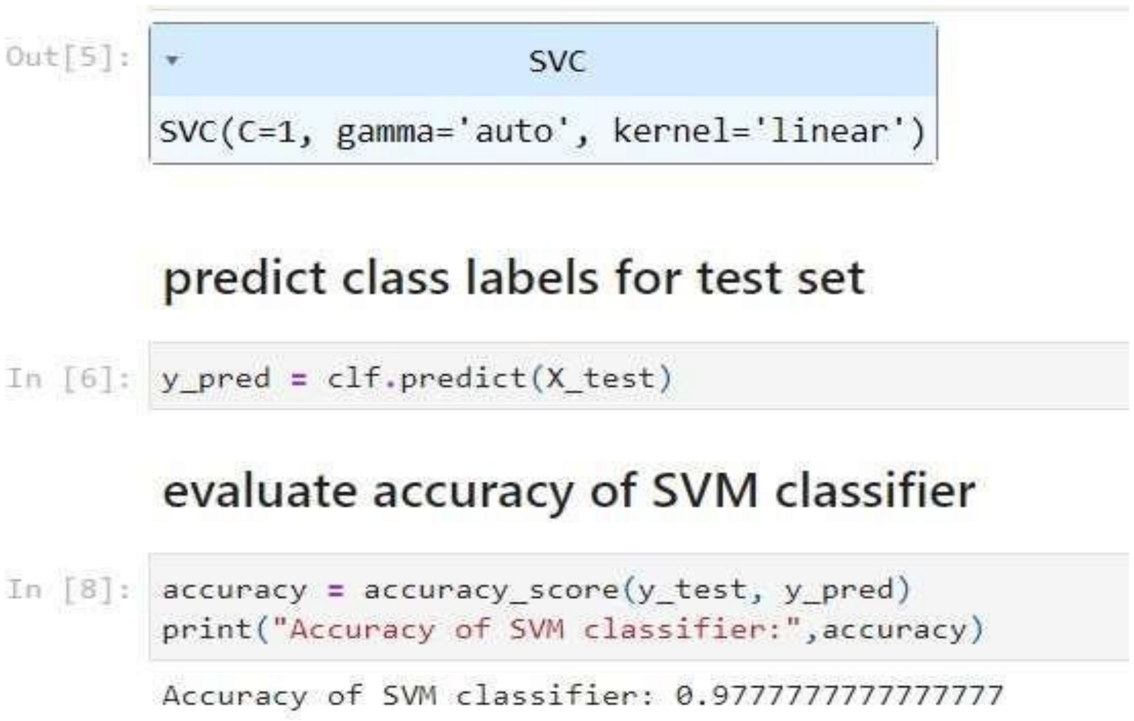
```
In [3]: X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.3, random_state=42)
```

normalize dataset

```
In [4]: scaler = StandardScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)
```

train SVM classifier

```
In [5]: clf = svm.SVC(kernel='linear', C=1, gamma='auto')
        clf.fit(X_train, y_train)
```



```
Out[5]:  SVC
         SVC(C=1, gamma='auto', kernel='linear')
```

predict class labels for test set

```
In [6]: y_pred = clf.predict(X_test)
```

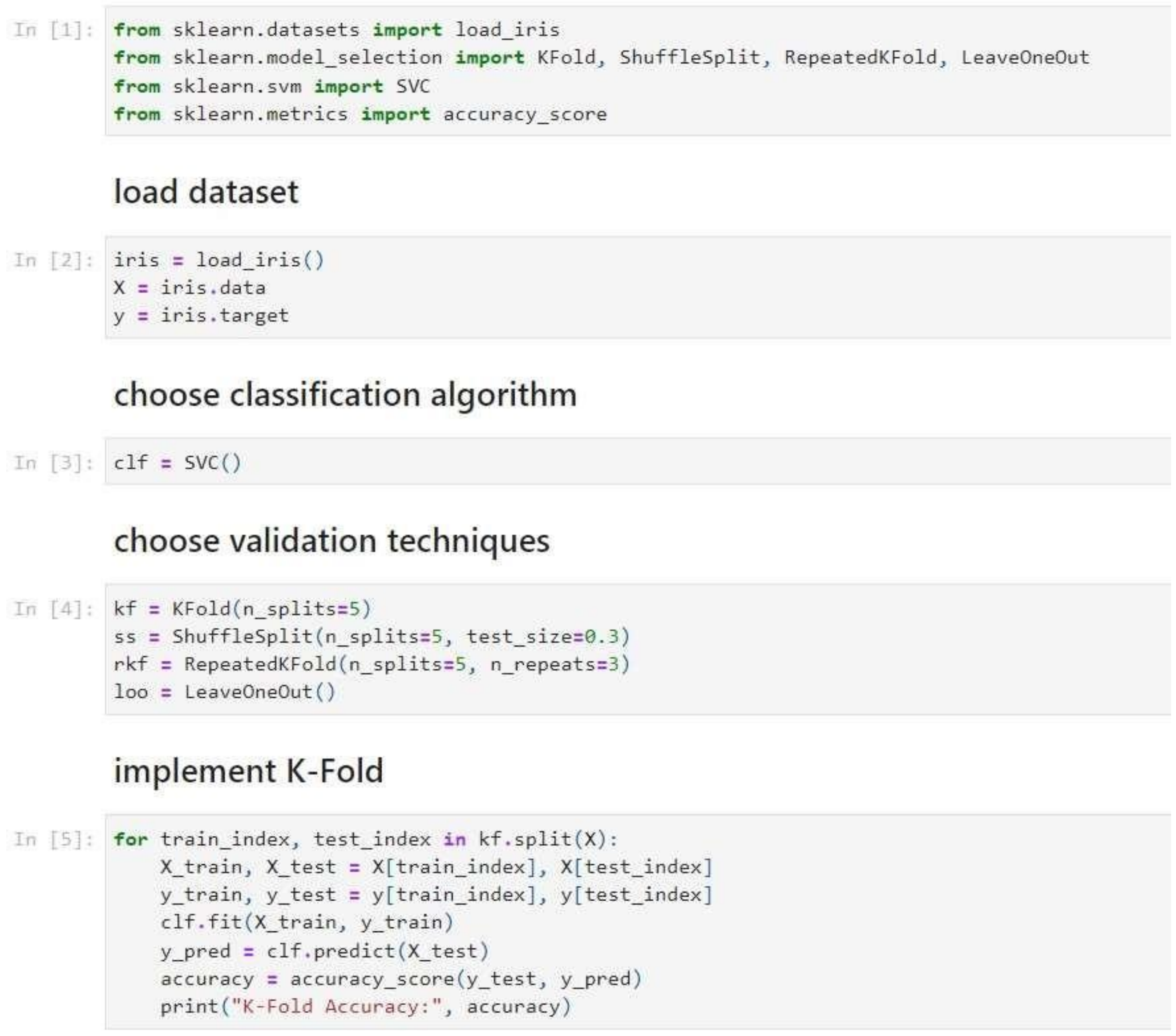
evaluate accuracy of SVM classifier

```
In [8]: accuracy = accuracy_score(y_test, y_pred)
        print("Accuracy of SVM classifier:",accuracy)

        Accuracy of SVM classifier: 0.9777777777777777
```

Practical 9

Aim :- Write a program to implement and check Sklearn's K-Fold, Shuffled K-fold, Repeated K-Fold and Leave-One-Out validation technique for appropriate classification algorithm and dataset.

Sol.



In [1]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold, ShuffleSplit, RepeatedKFold, LeaveOneOut
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

load dataset

In [2]:
```
iris = load_iris()
X = iris.data
y = iris.target
```

choose classification algorithm

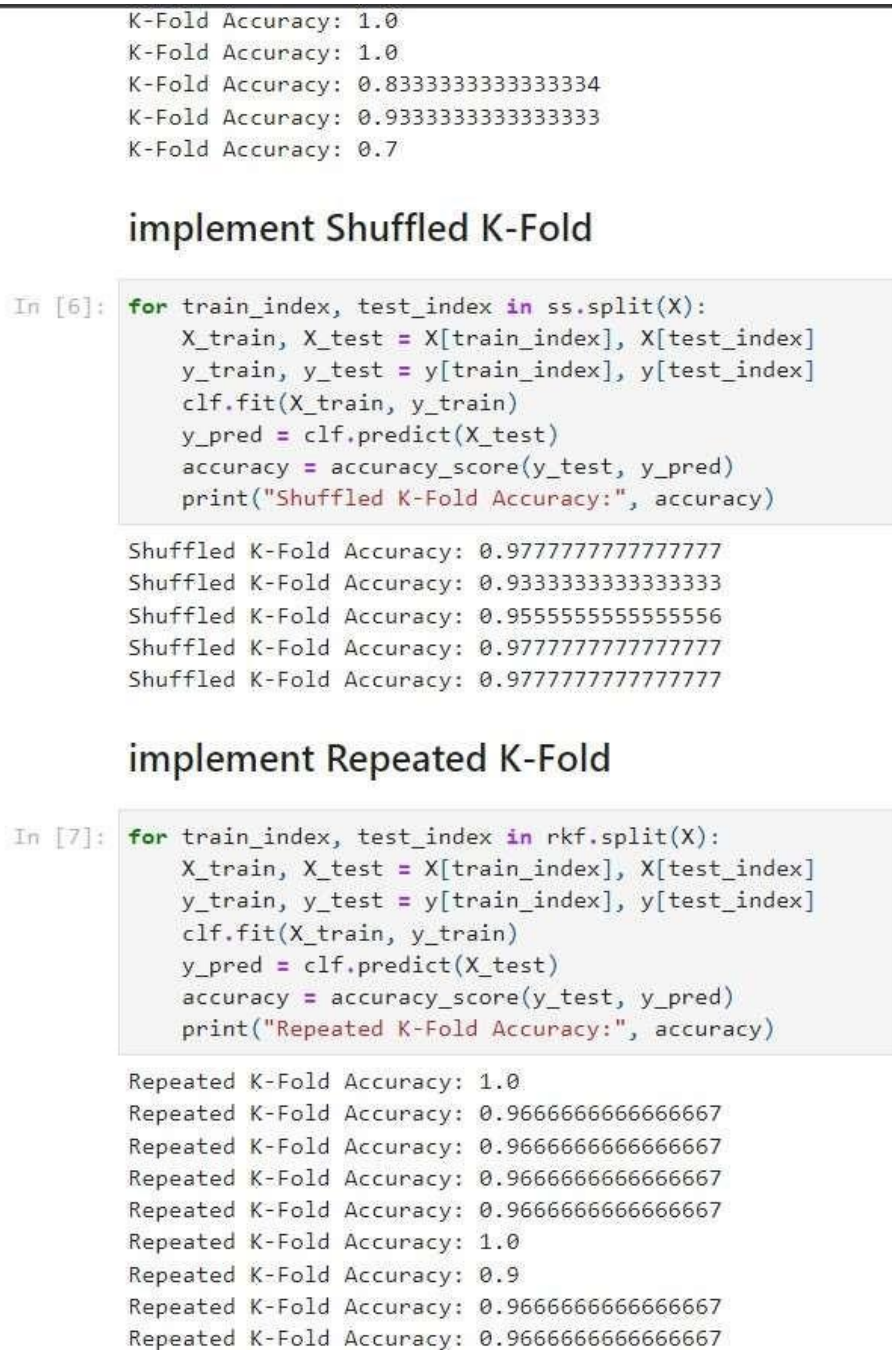
In [3]:
```
clf = SVC()
```

choose validation techniques

In [4]:
```
kf = KFold(n_splits=5)
ss = ShuffleSplit(n_splits=5, test_size=0.3)
rkf = RepeatedKFold(n_splits=5, n_repeats=3)
loo = LeaveOneOut()
```

implement K-Fold

In [5]:
```
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("K-Fold Accuracy:", accuracy)
```



```
K-Fold Accuracy: 1.0
K-Fold Accuracy: 1.0
K-Fold Accuracy: 0.8333333333333334
K-Fold Accuracy: 0.9333333333333333
K-Fold Accuracy: 0.7
```

implement Shuffled K-Fold

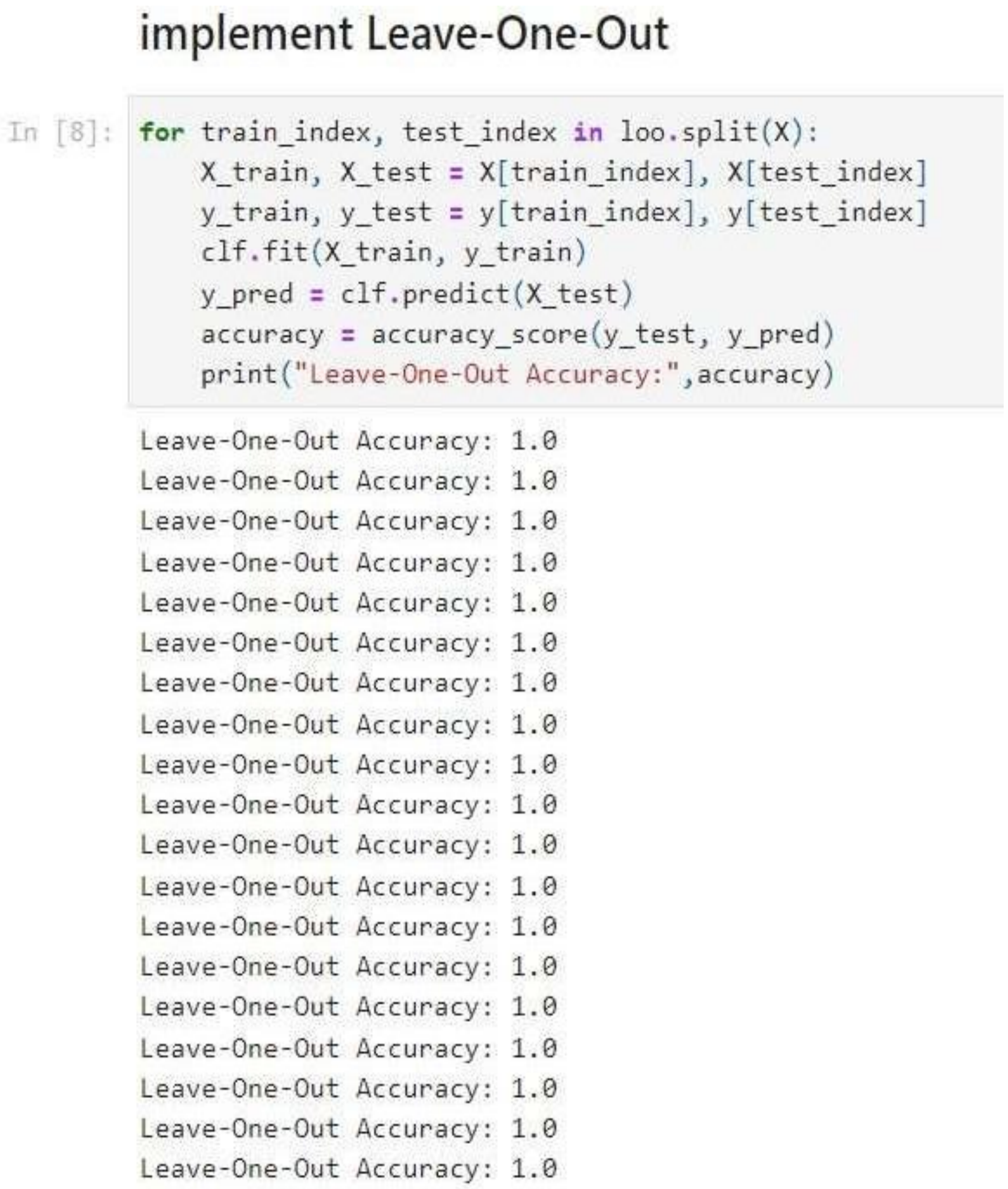
```
In [6]: for train_index, test_index in ss.split(X):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = y[train_index], y[test_index]
            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)
            print("Shuffled K-Fold Accuracy:", accuracy)
```

```
Shuffled K-Fold Accuracy: 0.9777777777777777
Shuffled K-Fold Accuracy: 0.9333333333333333
Shuffled K-Fold Accuracy: 0.9555555555555556
Shuffled K-Fold Accuracy: 0.9777777777777777
Shuffled K-Fold Accuracy: 0.9777777777777777
```

implement Repeated K-Fold

```
In [7]: for train_index, test_index in rkf.split(X):
            X_train, X_test = X[train_index], X[test_index]
            y_train, y_test = y[train_index], y[test_index]
            clf.fit(X_train, y_train)
            y_pred = clf.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)
            print("Repeated K-Fold Accuracy:", accuracy)
```

```
Repeated K-Fold Accuracy: 1.0
Repeated K-Fold Accuracy: 0.9666666666666667
Repeated K-Fold Accuracy: 0.9666666666666667
Repeated K-Fold Accuracy: 0.9666666666666667
Repeated K-Fold Accuracy: 0.9666666666666667
Repeated K-Fold Accuracy: 1.0
Repeated K-Fold Accuracy: 0.9
Repeated K-Fold Accuracy: 0.9666666666666667
Repeated K-Fold Accuracy: 0.9666666666666667
```



implement Leave-One-Out

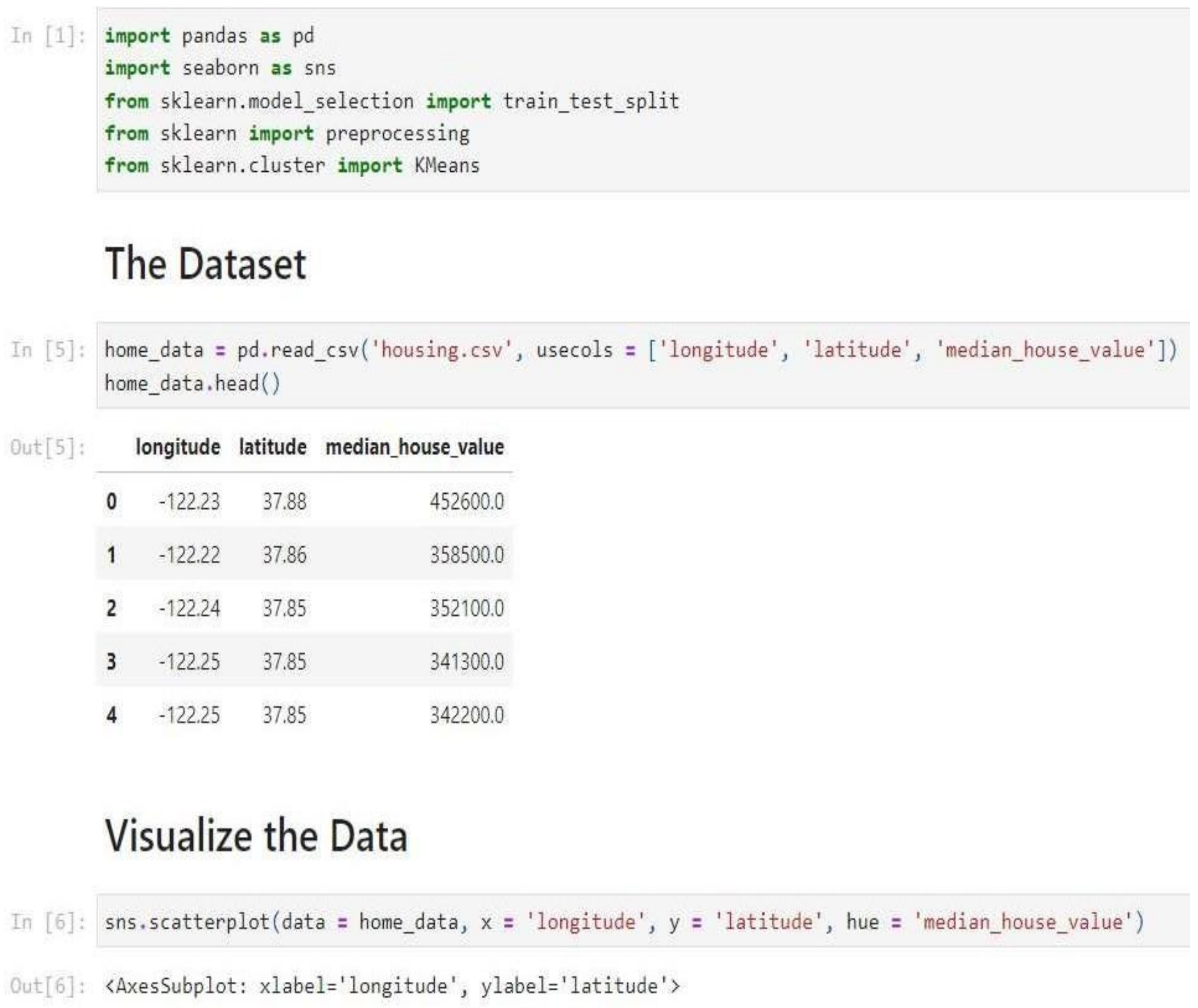
In [8]:
```
for train_index, test_index in loo.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Leave-One-Out Accuracy:",accuracy)
```

```
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
Leave-One-Out Accuracy: 1.0
```

Practical 10

Aim :- Write a program to implement k-Means clustering algorithm for a sample training data set stored as a .CSV file.

Sol.



In [1]:
```python
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.cluster import KMeans
```

The Dataset

In [5]:
```python
home_data = pd.read_csv('housing.csv', usecols = ['longitude', 'latitude', 'median_house_value'])
home_data.head()
```

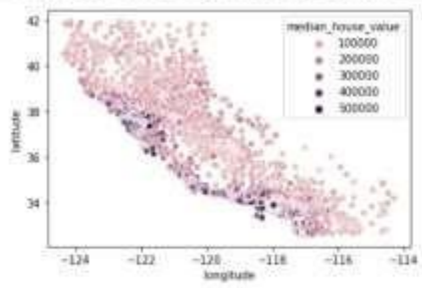
Out[5]:

	longitude	latitude	median_house_value
0	-122.23	37.88	452600.0
1	-122.22	37.86	358500.0
2	-122.24	37.85	352100.0
3	-122.25	37.85	341300.0
4	-122.25	37.85	342200.0

Visualize the Data

In [6]:
```python
sns.scatterplot(data = home_data, x = 'longitude', y = 'latitude', hue = 'median_house_value')
```

Out[6]:
```
<AxesSubplot: xlabel='longitude', ylabel='latitude'>
```



Normalizing the Data

```
In [7]: X_train, X_test, y_train, y_test = train_test_split(home_data[['latitude', 'longitude']], home_data[['median_house_value']], test_size=0.33, random_state=0)
```

```
In [8]: X_train_norm = preprocessing.normalize(X_train)
        X_test_norm = preprocessing.normalize(X_test)
```

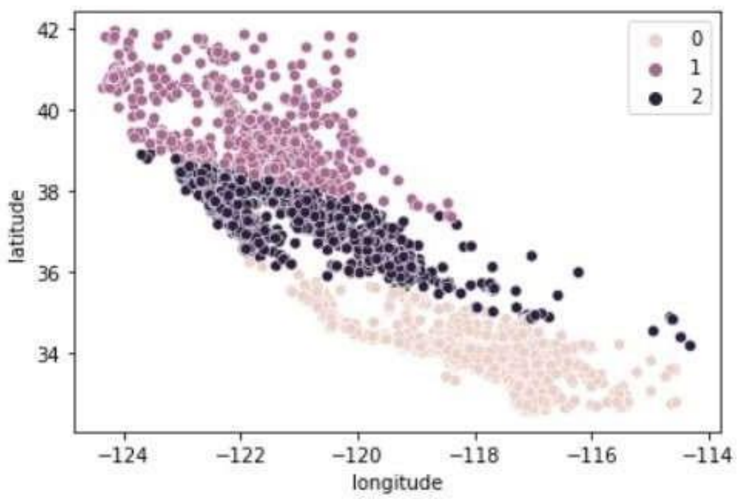
Fitting and Evaluating the Model

```
In [9]: kmeans = KMeans(n_clusters = 3, random_state = 0, n_init='auto')
        kmeans.fit(X_train_norm)
```

```
Out[9]: KMeans
        KMeans(n_clusters=3, n_init='auto', random_state=0)
```

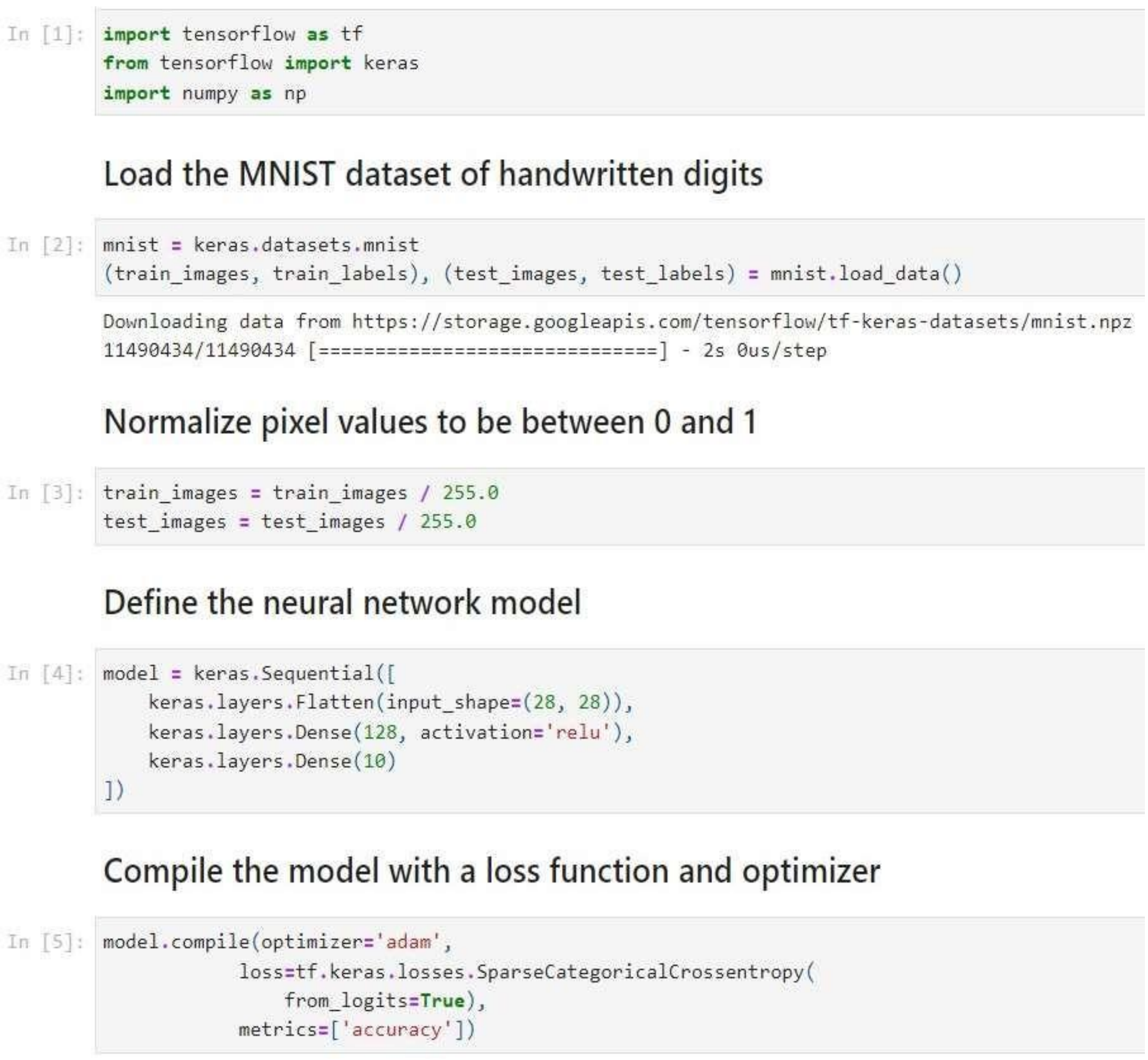
```
In [10]: sns.scatterplot(data = X_train, x = 'longitude', y = 'latitude', hue = kmeans.labels_)
```

```
Out[10]: <AxesSubplot: xlabel='longitude', ylabel='latitude'>
```



Practical 11

Aim :- Write a program to recognize handwritten digit using Artificial Neural Network.
Sol.



In [1]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
```

Load the MNIST dataset of handwritten digits

In [2]:
```
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

Normalize pixel values to be between 0 and 1

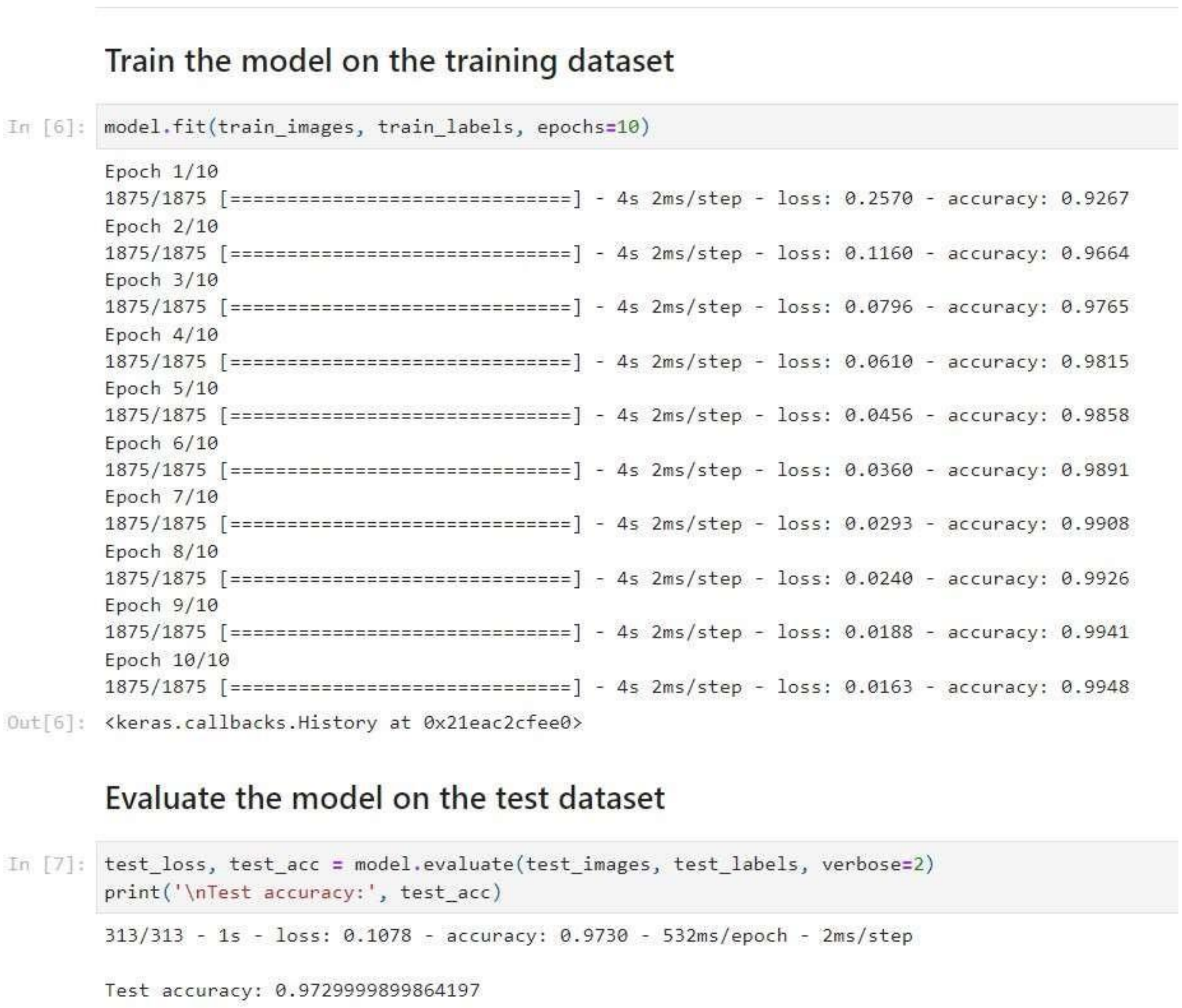
In [3]:
```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Define the neural network model

In [4]:
```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10)
])
```

Compile the model with a loss function and optimizer

In [5]:
```
model.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(
                from_logits=True),
            metrics=['accuracy'])
```



Train the model on the training dataset

```
In [6]: model.fit(train_images, train_labels, epochs=10)
```

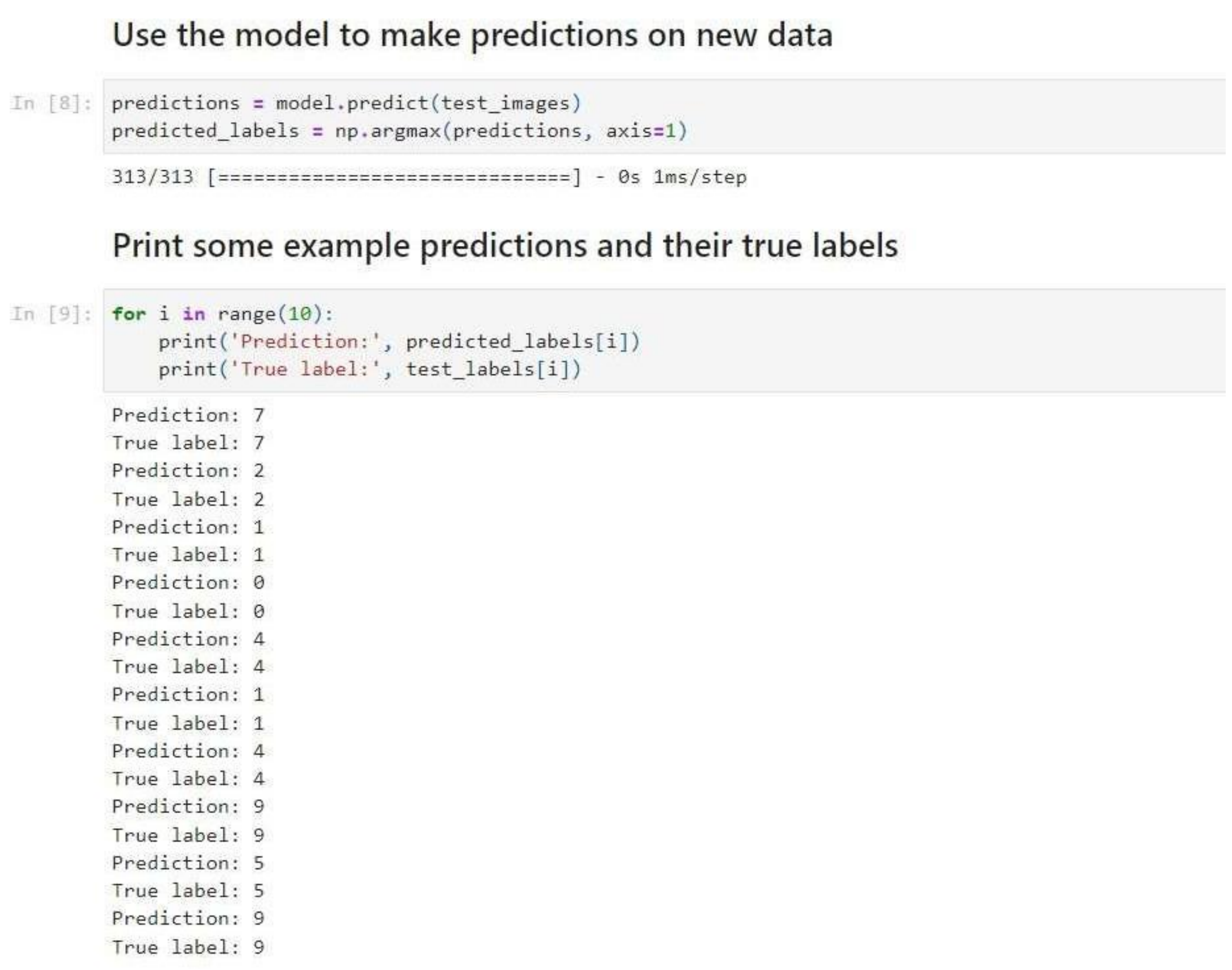
```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2570 - accuracy: 0.9267
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1160 - accuracy: 0.9664
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0796 - accuracy: 0.9765
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0610 - accuracy: 0.9815
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0456 - accuracy: 0.9858
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0360 - accuracy: 0.9891
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0293 - accuracy: 0.9908
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0240 - accuracy: 0.9926
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0188 - accuracy: 0.9941
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0163 - accuracy: 0.9948
```

```
Out[6]: <keras.callbacks.History at 0x21eac2cfee0>
```

Evaluate the model on the test dataset

```
In [7]: test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
        print('\nTest accuracy:', test_acc)
```

```
313/313 - 1s - loss: 0.1078 - accuracy: 0.9730 - 532ms/epoch - 2ms/step

Test accuracy: 0.9729999899864197
```



Use the model to make predictions on new data

In [8]:
```
predictions = model.predict(test_images)
predicted_labels = np.argmax(predictions, axis=1)
```

```
313/313 [==============================] - 0s 1ms/step
```

Print some example predictions and their true labels

In [9]:
```
for i in range(10):
    print('Prediction:', predicted_labels[i])
    print('True label:', test_labels[i])
```

```
Prediction: 7
True label: 7
Prediction: 2
True label: 2
Prediction: 1
True label: 1
Prediction: 0
True label: 0
Prediction: 4
True label: 4
Prediction: 1
True label: 1
Prediction: 4
True label: 4
Prediction: 9
True label: 9
Prediction: 5
True label: 5
Prediction: 9
True label: 9
```

Practical 12

Aim :- Study Weka toolkit for demonstration of regression, classification and clustering models on it.

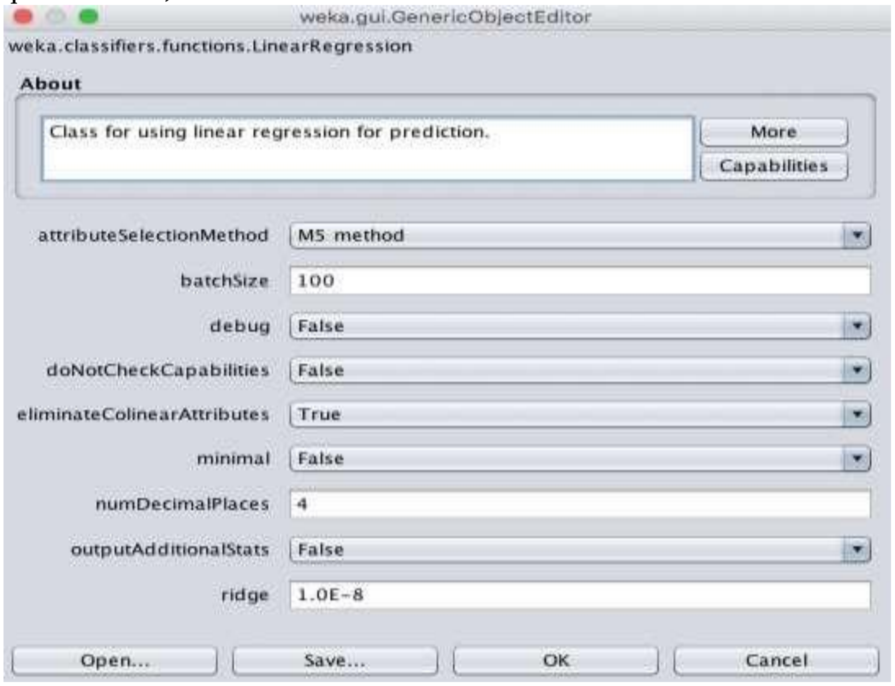
Sol.

Weka is a popular open-source machine learning toolkit that provides a wide range of algorithms for data analysis, including regression, classification, and clustering models. In this answer, we will provide a brief overview of how to use Weka to train and evaluate these types of models.

Regression:

Weka provides several regression algorithms, such as Linear Regression, Multilayer Perceptron, and Support Vector Regression. To demonstrate how to use Weka for regression, let's use the 'houses' dataset from the UCI Machine Learning Repository, which contains information about houses in the suburbs of Boston.

Load the data into Weka. You can either use the built-in datasets in Weka or import your own data. In this case, you can download the dataset in CSV format from the UCI Machine Learning Repository and then load it into Weka using the 'Explorer' interface.
Preprocess the data. Depending on the dataset, you may need to preprocess it by cleaning, transforming, or scaling the data. In this case, we can normalize the input variables by selecting the 'Normalize' filter from the 'Filter' panel in the 'Explorer' interface.
Select a regression algorithm. In this case, let's use the Linear Regression algorithm by selecting it from the 'Classify' panel in the 'Explorer' interface. Train the model. To train the model, you can either use the 'Run' button in the 'Classifier' panel or run the regression algorithm from the command line using the Weka command-line interface (CLI).
Evaluate the model. Once the model is trained, you can evaluate its performance using the 'Evaluate' panel in the 'Explorer' interface. This will give you metrics such as Mean Absolute Error, Root Mean Squared Error, and Coefficient of Determination.

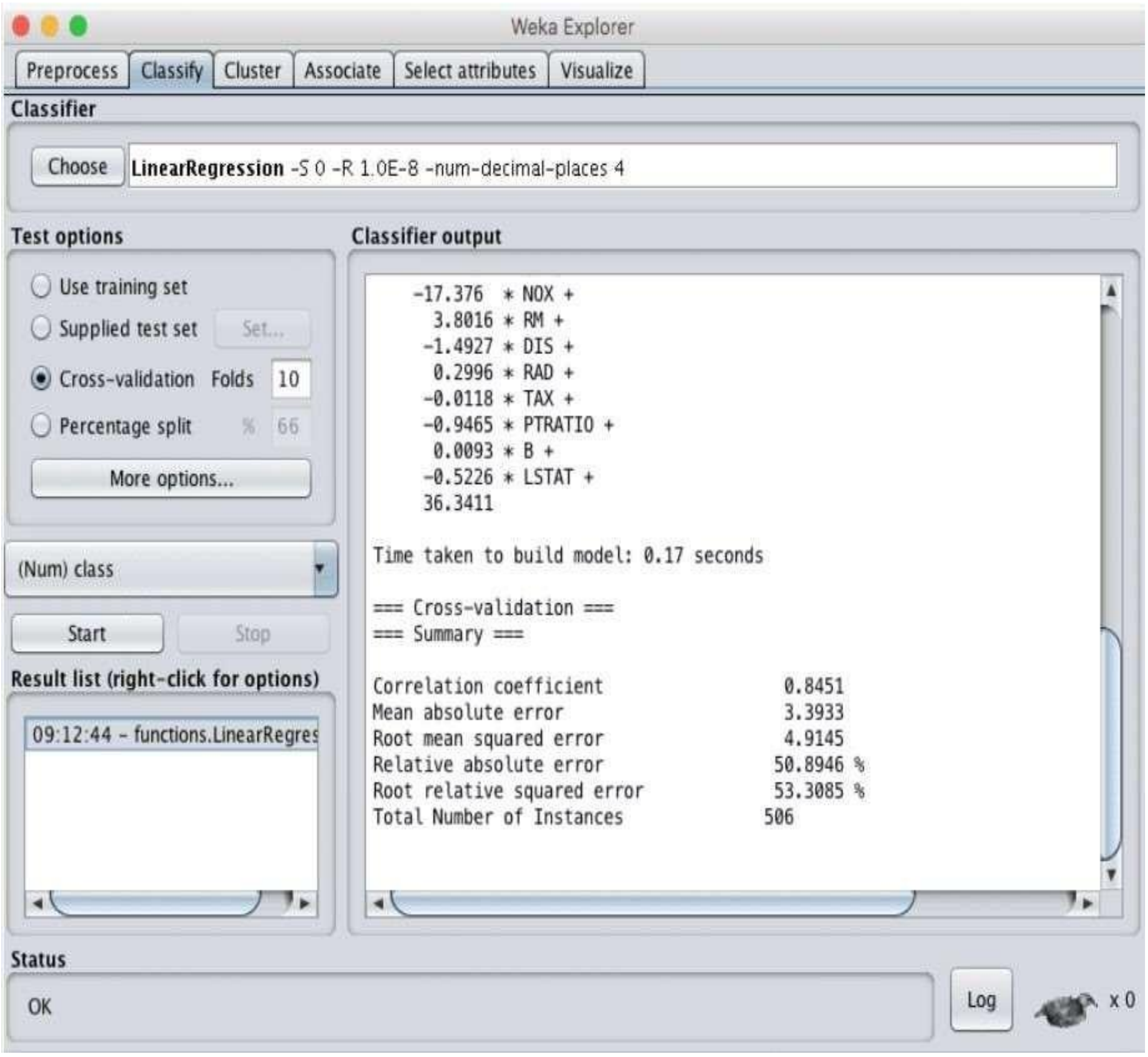

weka.gui.GenericObjectEditor

weka.classifiers.functions.LinearRegression

About

Class for using linear regression for prediction.

attributeSelectionMethod	M5 method
batchSize	100
debug	False
doNotCheckCapabilities	False
eliminateColinearAttributes	True
minimal	False
numDecimalPlaces	4
outputAdditionalStats	False
ridge	1.0E-8



LinearRegression -S 0 -R 1.0E-8 -num-decimal-places 4

Classifier output

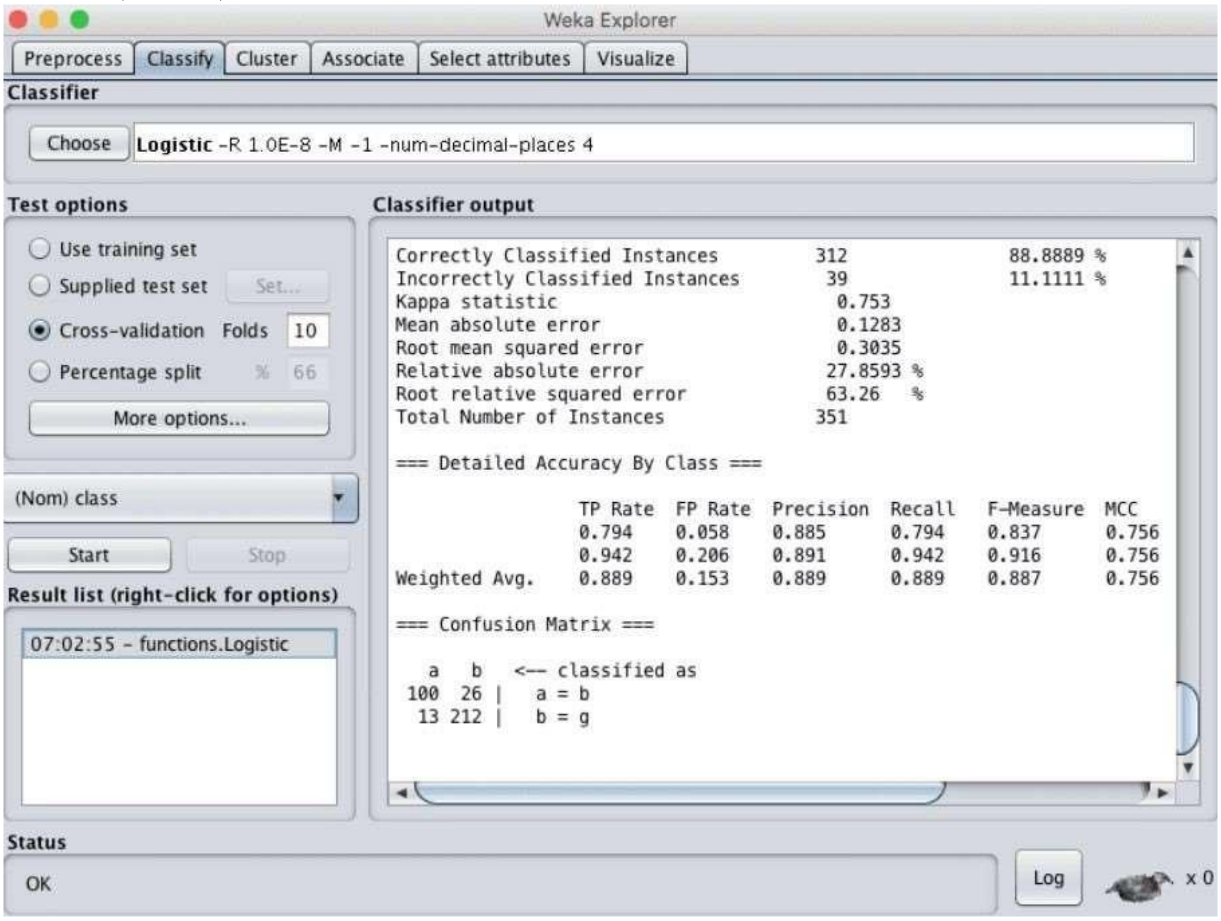
```
    -17.376  * NOX +
      3.8016 * RM +
     -1.4927 * DIS +
      0.2996 * RAD +
     -0.0118 * TAX +
     -0.9465 * PTRATIO +
      0.0093 * B +
     -0.5226 * LSTAT +
     36.3411

Time taken to build model: 0.17 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8451
Mean absolute error                      3.3933
Root mean squared error                  4.9145
Relative absolute error                 50.8946 %
Root relative squared error             53.3085 %
Total Number of Instances              506
```

Classification:

Weka provides a wide range of classification algorithms, such as Decision Trees, Random Forests, and Naive Bayes. To demonstrate how to use Weka for classification, let's use the 'iris' dataset, which contains measurements of flowers and their species.

Load the data into Weka. You can either use the built-in datasets in Weka or import your own data. In this case, you can use the 'iris.arff' file, which is a standard dataset that comes with Weka.
Select a classification algorithm. In this case, let's use the J48 decision tree algorithm by selecting it from the 'Classify' panel in the 'Explorer' interface. Train the model. To train the model, you can either use the 'Run' button in the 'Classifier' panel or run the classification algorithm from the command line using the Weka CLI.

Evaluate the model. Once the model is trained, you can evaluate its performance using the 'Evaluate' panel in the 'Explorer' interface. This will give you metrics such as Accuracy, Precision, Recall, and F-Measure.



Classifier: Logistic -R 1.0E-8 -M -1 -num-decimal-places 4

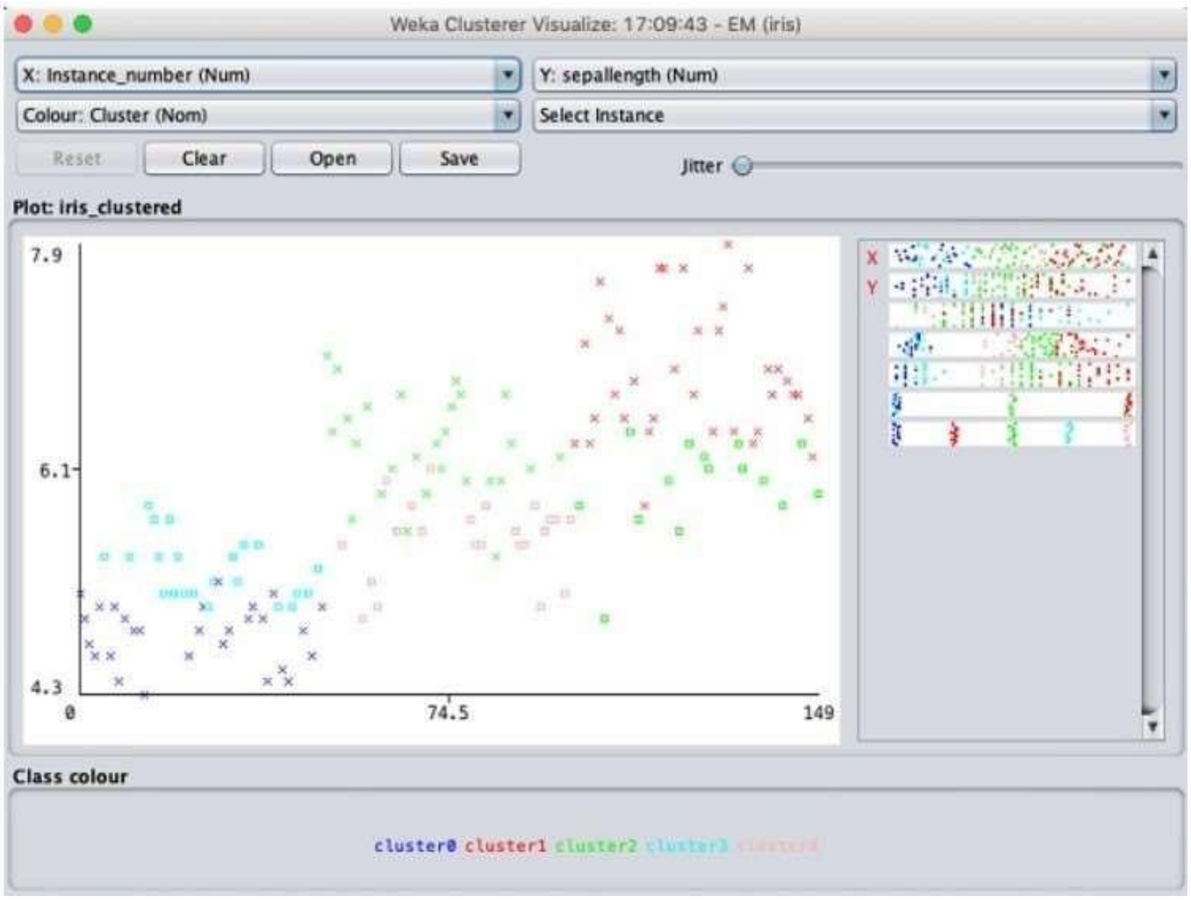
Classifier output

```
Correctly Classified Instances         312               88.8889 %
Incorrectly Classified Instances        39               11.1111 %
Kappa statistic                          0.753
Mean absolute error                      0.1283
Root mean squared error                  0.3035
Relative absolute error                 27.8593 %
Root relative squared error             63.26   %
Total Number of Instances              351

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC
                 0.794    0.058    0.885      0.794    0.837      0.756
                 0.942    0.206    0.891      0.942    0.916      0.756
Weighted Avg.    0.889    0.153    0.889      0.889    0.887      0.756

=== Confusion Matrix ===

   a   b   <-- classified as
 100  26 |   a = b
  13 212 |   b = g
```

Clustering:

Weka provides several clustering algorithms, such as k-Means, Hierarchical Clustering, and DBSCAN. To demonstrate how to use Weka for clustering, let's use the 'iris' dataset again.

Load the data into Weka. You can use the same 'iris.arff' file as before.
Select a clustering algorithm. In this case, let's use the k-Means algorithm by selecting it from the 'Cluster' panel in the 'Explorer' interface.
Configure the algorithm. Depending on the clustering algorithm, you may need to configure parameters such as the number of clusters or the distance metric. In this case, we can set the number of clusters to 3 and leave the other parameters at their default values. Run the algorithm


Weka Clusterer Visualize: 17:09:43 - EM (iris)
X: Instance_number (Num)
Y: sepallength (Num)
Colour: Cluster (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: iris_clustered
7.9
6.1
4.3
0
74.5
149
X
Y
Class colour
cluster0
cluster1
cluster2
cluster3
cluster4