Iris Flower Classification

Program:

```
#Importing the packages and libraries for data visualization and predictive data analysis

import pandas as pd

import numpy as np

import matplotlib. pyplot as pt

import seaborn as sns

import sklearn


#To load the Iris flower dataset

from sklearn. datasets import load_iris

Iris= load_iris ()

Iris
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
```

```
#To print the shape of the features which prints the number of observations and number of features respectively.

print (Iris.data.shape)
```

```
print(Iris.data.shape)
```

```
(150, 4)
```

#To print the shape of the response, which prints only the number of observations.

print(Iris.target.shape)

```
print(Iris.target.shape)
```

```
(150,)
```

#To print the feature names- sepal length, petal length, sepal width, petal width

print(Iris.feature_names)

```
print(Iris.feature_names)
```

```
... ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

#To print the target names, which is the different classes, i.e., the 3 different species of the Iris flower

print(Iris.target_names)

```
print(Iris.target_names)
```

```
... ['setosa' 'versicolor' 'virginica']
```

#Summarization and description of the Iris Flower dataset

print(Iris.DESCR)

```
print(Iris.DESCR)
```

```
.. _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

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.
.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

```
#conversion of Data into the pandas Dataframe

df=pd.DataFrame(Iris.data, columns=Iris.feature_names)

df.head()
```

#this prints first 5 feature observation

```
df=pd.DataFrame(Iris.data, columns=Iris.feature_names)
df.head()
```

...

	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

#For printing the last 5 feature observations

df.tail()

```
df.tail()
```

[48]

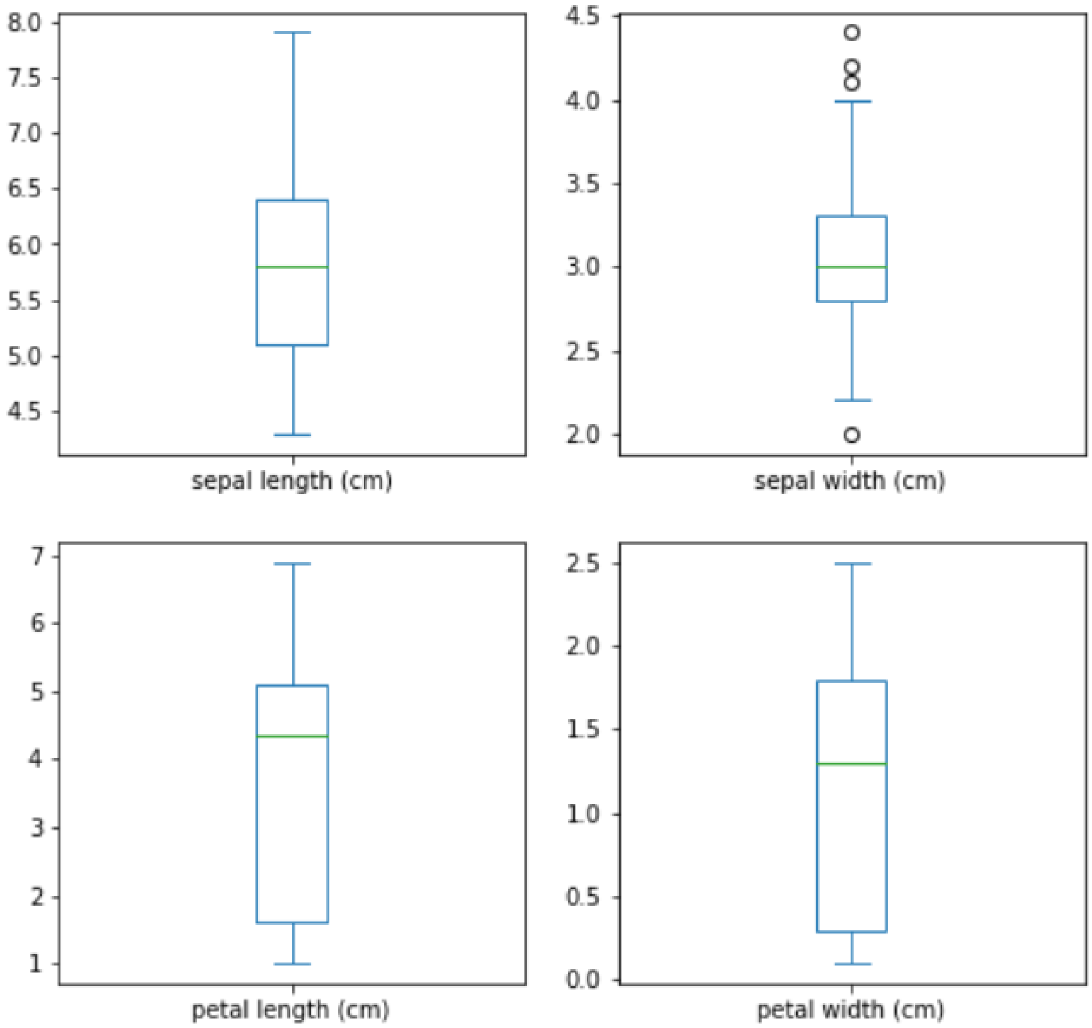
...

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
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

Data Visualizations:

#Box and whisker plots

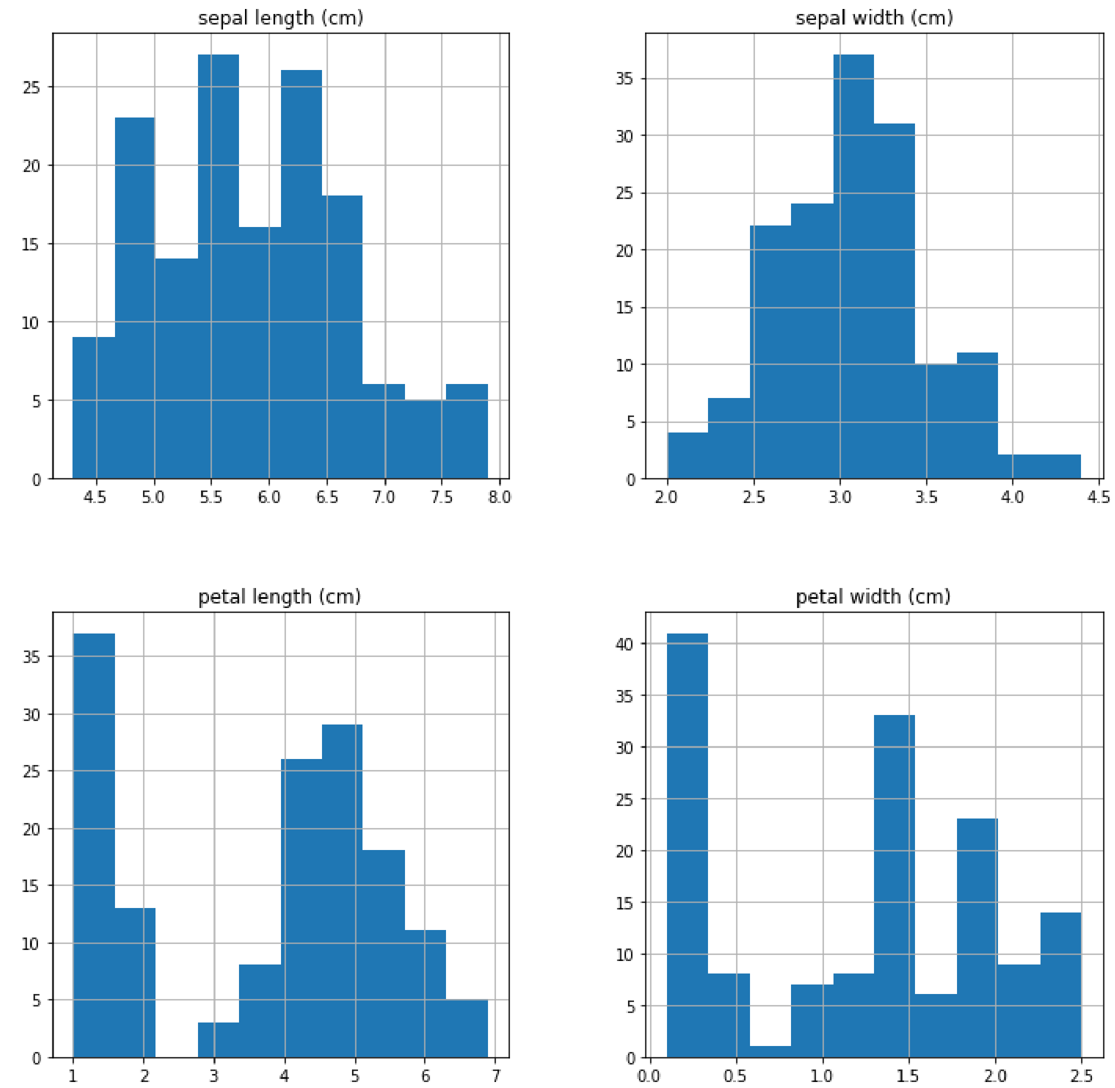
```
df.plot(kind='box', subplots=True, layout=(3,2),figsize=(8,12));
```



#Histogram

```
df.hist(figsize=(12,12))
```

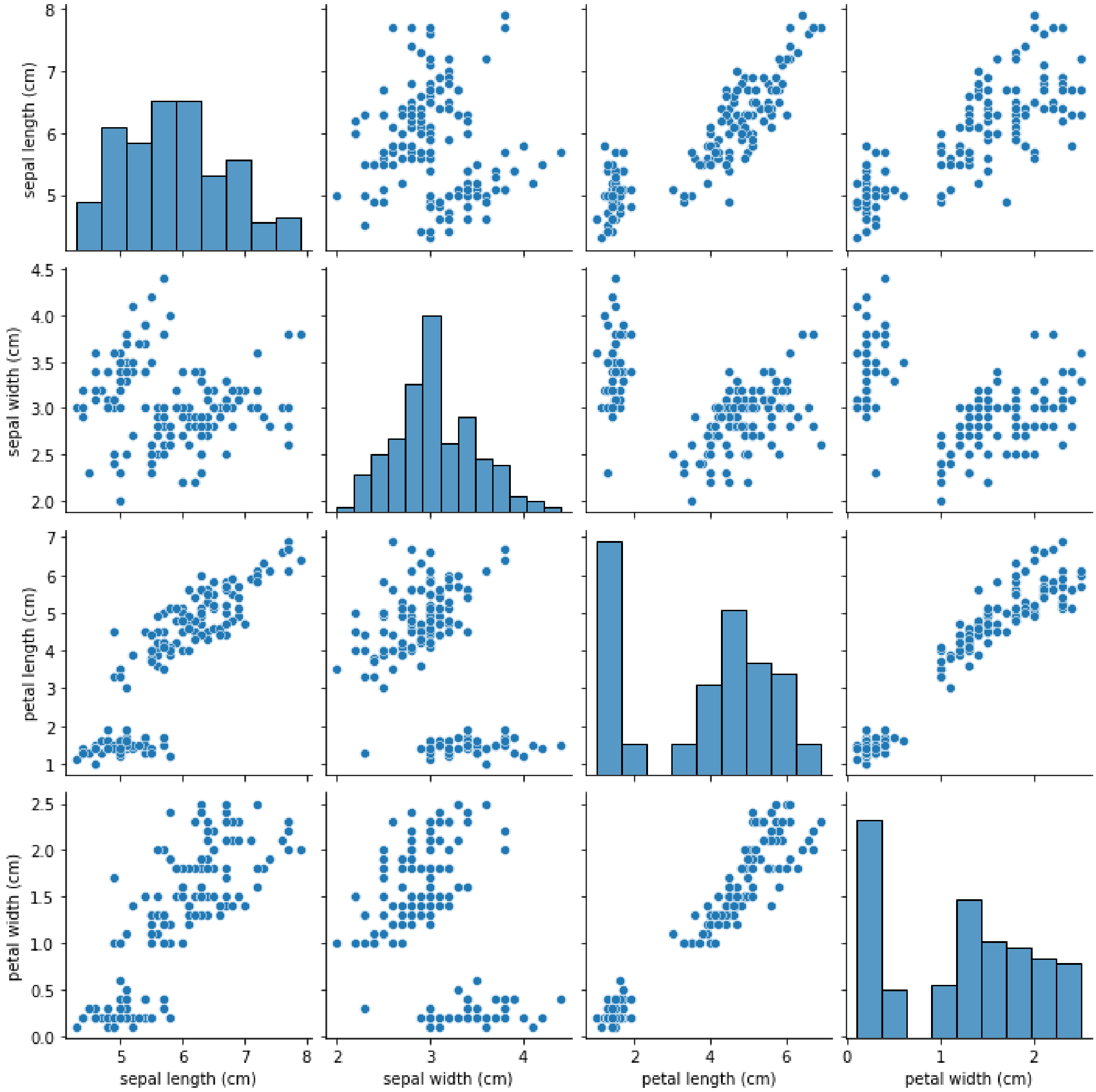
```
pt. Show()
```



#Pair plot

```
#using seaborn data visualization library
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x172605dcee0>
```



`#Correlation Plot`

This plot displays the correlation, the measure of best used in variables that demonstrate a linear relationship between each other.

Here it is demonstrated via heatmap

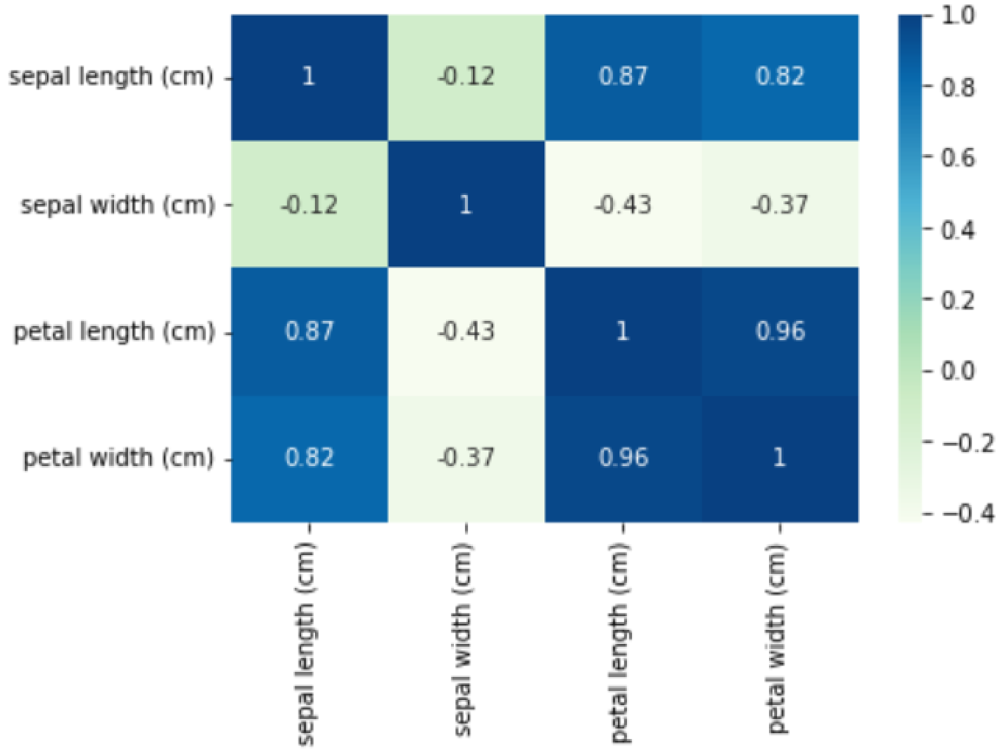
```
df.corr()
```

```
df.corr()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal length (cm)	0.871754	-0.428440	1.000000	0.962865
petal width (cm)	0.817941	-0.366126	0.962865	1.000000

```
#Heat map
sns.heatmap(df.corr(),annot=True,cmap='GnBu')
```

```
<AxesSubplot:>
```



Evaluation of Algorithms

```
#Separating data into dependent and independent variables
```

```
X=Iris.data

Y=Iris.target

print(X.shape)

print(Y.shape)
```

```
X=Iris.data
Y=Iris.target
print(X.shape)
print(Y.shape)
```

```
(150, 4)
(150,)
```

```
#Splitting the training and test data

X_train, X_test, Y_train, Y_test=
sklearn.model_selection.train_test_split(X,Y,test_size=0.25,
random_state=2)
```

Algorithms:

Logistic Regression:

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression is estimating the parameters of a logistic model (a form of binary regression).

```
from sklearn.linear_model import LogisticRegression

lorg=LogisticRegression()

print(lorg.fit(X_train, Y_train))

Y_pred=lorg.predict(X_test)

print(Y_pred)
```

```
from sklearn.linear_model import LogisticRegression
lorg=LogisticRegression()
print(lorg.fit(X_train, Y_train))
```

```
LogisticRegression()
```

```
Y_pred=lorg.predict(X_test)
print(Y_pred)
```

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2 2 0 1 2 1 0 2
```

```
#confusion matrix
```

Confusion Matrix:

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(Y_test, Y_pred))
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(Y_test, Y_pred))
```

```
[[16  0  0]
 [ 0 10  1]
 [ 0  0 11]]
```

```
#Accuracy score
```

Accuracy Score:

It is the ratio of number of correct predictions to the total number of input samples.

```
from sklearn.metrics import accuracy_score

print("Algorithm:Logistic Regression")

print("Accuracy of the model is",accuracy_score(Y_test, Y_pred))
```

```
from sklearn.metrics import accuracy_score
print("Algorithm:Logistic Regression")
print("Accuracy of the model is",accuracy_score(Y_test, Y_pred))
```

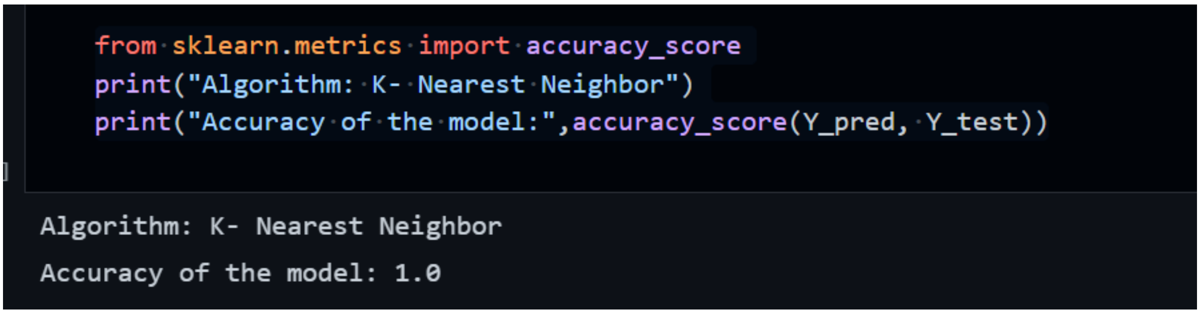
```
Algorithm:Logistic Regression
Accuracy of the model is 0.9736842105263158
```

The accuracy shown by the Logistic Regression Classifier is 0.97

K- Nearest Neighbors:

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems.

```
from sklearn.neighbors import KNeighborsClassifier

kn=KNeighborsClassifier()

kn.fit(X_train, Y_train)

Y_pred=kn.predict(X_test)

print(Y_pred)

from sklearn.metrics import confusion_matrix

print(confusion_matrix(Y_test, Y_pred))

from sklearn.metrics import accuracy_score

print("Algorithm: K- Nearest Neighbor")

print("Accuracy of the model:",accuracy_score(Y_pred, Y_test))
```



The accuracy shown by the KNN Classifier is 1.0

Support Vector Machine:

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. ... The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features.

```
from sklearn.svm import SVC

sm=SVC()

sm.fit(X_train, Y_train)

Y_pred=sm.predict(X_test)

print(Y_pred)
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(Y_test, Y_pred))
from sklearn.metrics import accuracy_score
print("Algorithm: Support Vector Machine")
print("Accuracy of the model:",accuracy_score(Y_pred, Y_test))
```

```
from sklearn.metrics import accuracy_score
print("Algorithm: Support Vector Machine")
print("Accuracy of the model:",accuracy_score(Y_pred, Y_test))
```

```
Algorithm: Support Vector Machine
Accuracy of the model: 0.9736842105263158
```

The accuracy shown by the SVM Classifier is 0.97

Decision Tree

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements

```
from sklearn.tree import DecisionTreeClassifier
dt= DecisionTreeClassifier()
dt.fit(X_train, Y_train)
Y_pred= dt.predict(X_test)
print(Y_pred)
from sklearn.metrics import confusion_matrix
print(confusion_matrix(Y_test, Y_pred))
from sklearn.metrics import accuracy_score
print("Algorithm: Decision Tree")
print("Accuracy of the model:",accuracy_score(Y_pred, Y_test))
```

```
from sklearn.metrics import accuracy_score
print("Algorithm: Decision Tree")
print("Accuracy of the model:",accuracy_score(Y_pred, Y_test))
```

```
Algorithm: Decision Tree
Accuracy of the model: 0.9473684210526315
```

The accuracy shown by the Decision tree Classifier is 0.94

Gaussian Naïve Bayes:

Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. Naive Bayes are a group of supervised machine learning classification algorithms based on the Bayes theorem

```
from sklearn.naive_bayes import GaussianNB

nb= GaussianNB()

nb.fit(X_train, Y_train)

Y_pred= nb.predict(X_test)

print(Y_pred)

from sklearn.metrics import confusion_matrix

print(confusion_matrix(Y_test, Y_pred))

from sklearn.metrics import accuracy_score

print("Algorithm: Gaussian Naive Bayes")

print("Accuracy of the model:",accuracy_score(Y_pred, Y_test))
```

```
from sklearn.metrics import accuracy_score
print("Algorithm: Gaussian Naive Bayes")
print("Accuracy of the model:",accuracy_score(Y_pred, Y_test))
```

```
Algorithm: Gaussian Naive Bayes
Accuracy of the model: 0.9736842105263158
```

The accuracy shown by the Gaussian Naïve bayes Classifier is 0.97

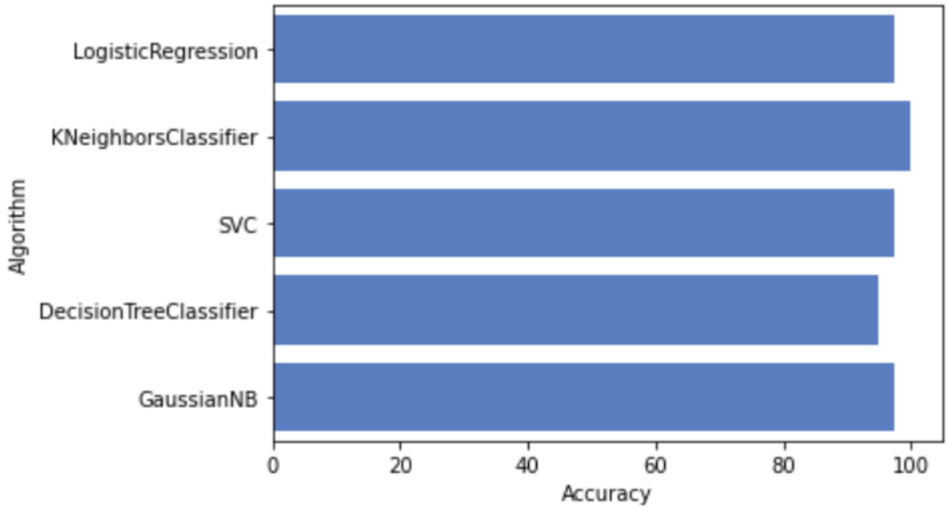
```
#For comparison and visualization

from sklearn.metrics import accuracy_score, log_loss

Algo=[LogisticRegression(), KNeighborsClassifier(), SVC(),
DecisionTreeClassifier(), GaussianNB()]

log_cols= ["Algorithm","Accuracy","Log loss"]

log=pd.DataFrame(columns=log_cols)
```

```
for a in Algo:

    a.fit(X_train, Y_train)

    name = a.__class__.__name__

    print(name)

    print('Results:')

    train_predictions = a.predict(X_test)

    acc = accuracy_score(Y_test, train_predictions)

    print("Accuracy: {:.4%}".format(acc))

    log_entry = pd.DataFrame([[name, acc*100, 11]],
columns=log_cols)

    log = log.append(log_entry)

    print("_"*30)
```

```
LogisticRegression
Results:
Accuracy: 97.3684%
______________________________
KNeighborsClassifier
Results:
Accuracy: 100.0000%
______________________________
SVC
Results:
Accuracy: 97.3684%
______________________________
DecisionTreeClassifier
Results:
Accuracy: 94.7368%
______________________________
GaussianNB
Results:
Accuracy: 97.3684%
______________________________
```

```
#Visualization using bar graph

sns.barplot(x='Accuracy', y='Algorithm', data=log, color="b")
```

```
<AxesSubplot:xlabel='Accuracy', ylabel='Algorithm'>
```



It is obvious from the data analysis that the K- Nearest Neighbors classifier shows the highest accuracy of 100%