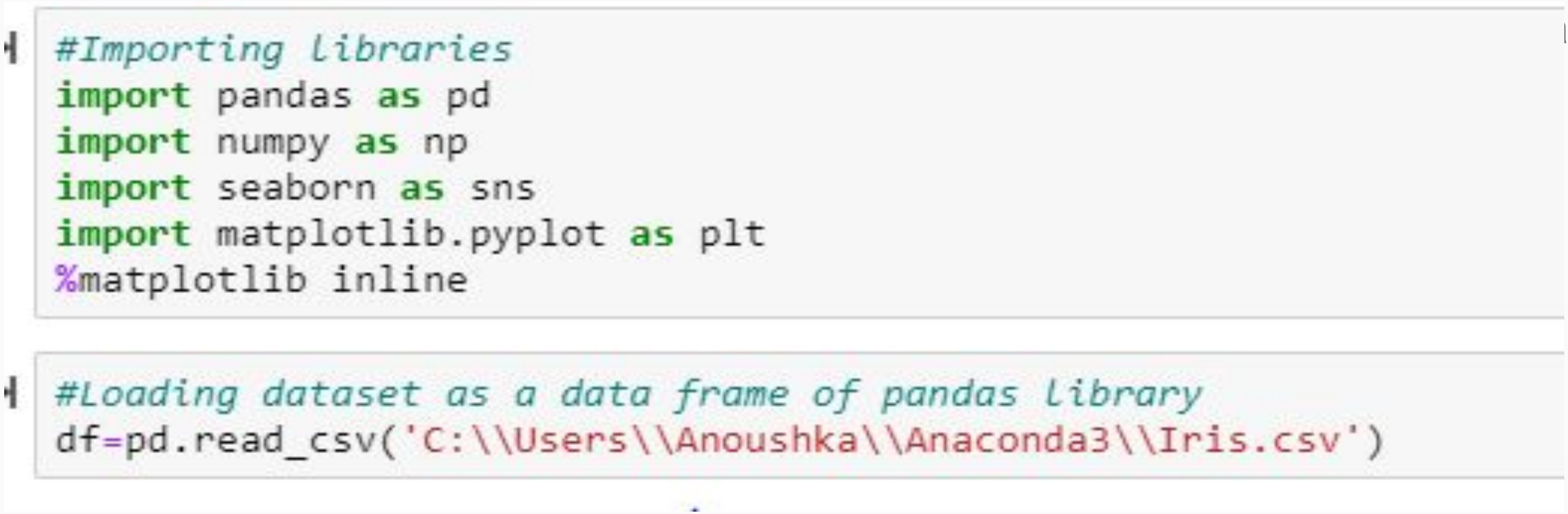
EDA on Iris Dataset using python.

Aim of the EDA

:To group a new Iris-Flower into one of the three species.

Importing python libraries and loading dataset in dataframe.



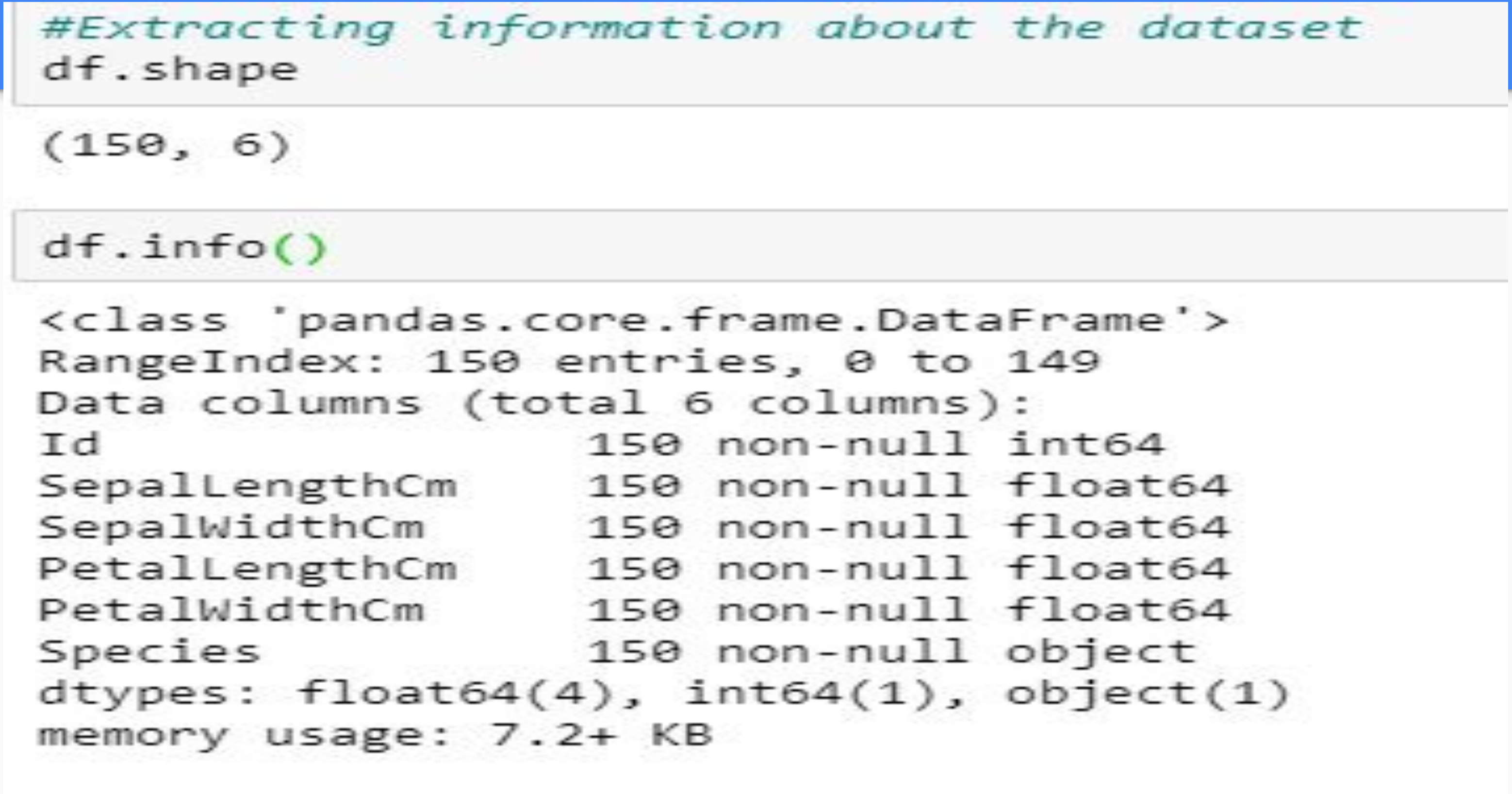
```
#Importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#Loading dataset as a data frame of pandas library
df=pd.read_csv('C:\\Users\\Anoushka\\Anaconda3\\Iris.csv')
```

Finding information about dataset :

The dataset has 150 rows and 6 columns.

Using the function info() we can see the data types of the columns.



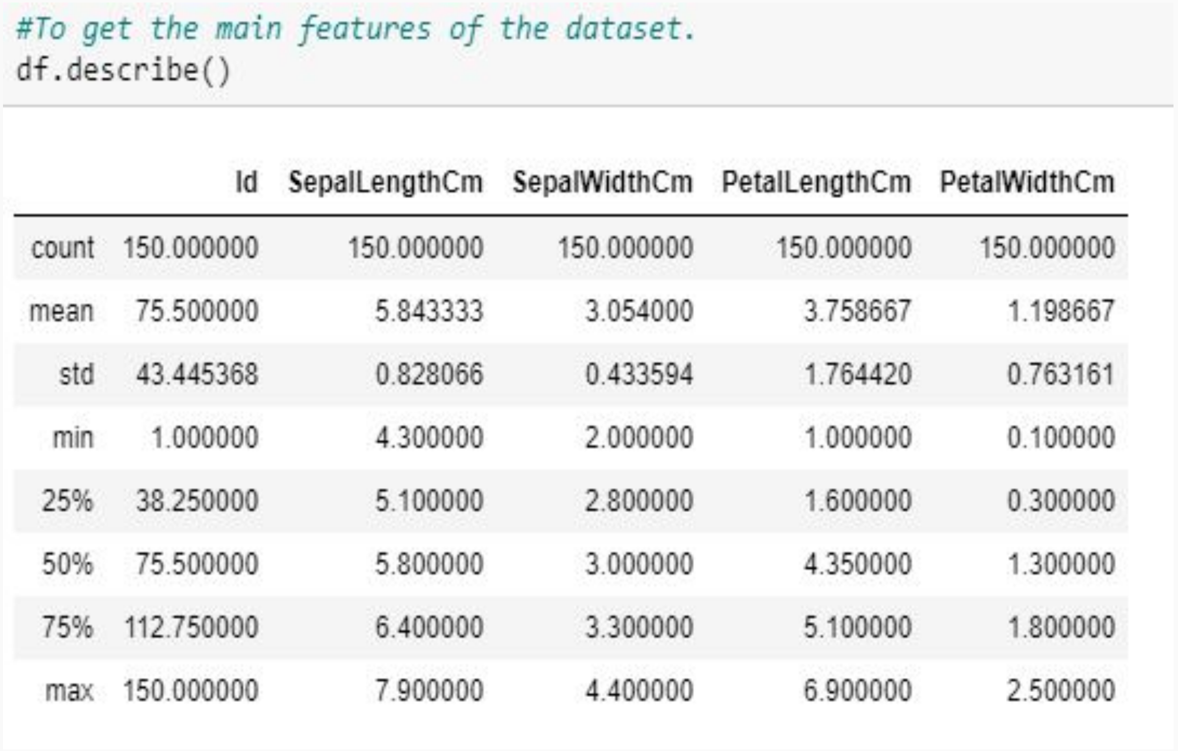
```
#Extracting information about the dataset
df.shape
```

```
(150, 6)
```

```
df.info()
```

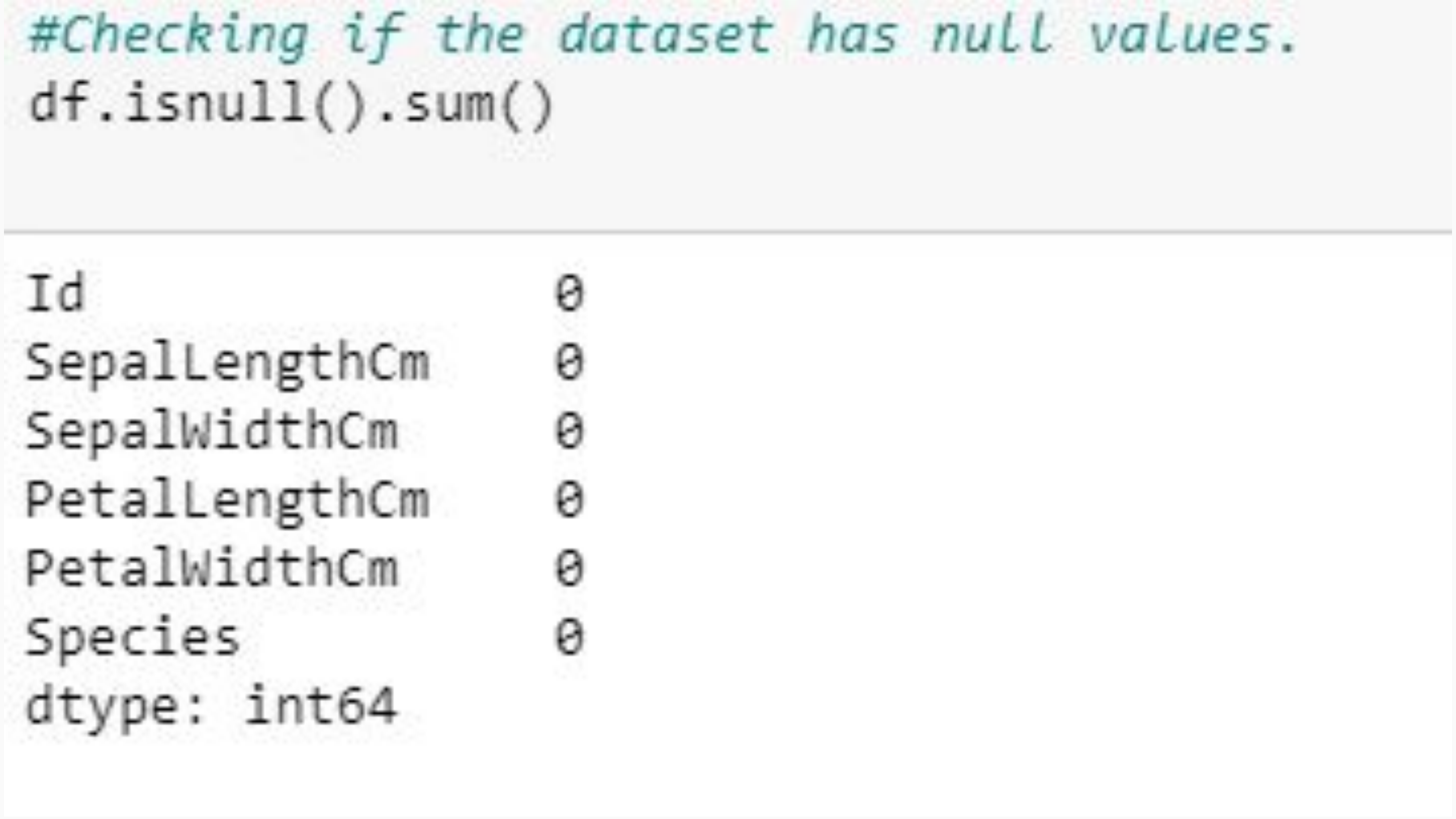
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
Id               150 non-null int64
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Using describe() function to get the various features like mean ,standard deviation and quartile ranges of dataset.



```
#To get the main features of the dataset.
df.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

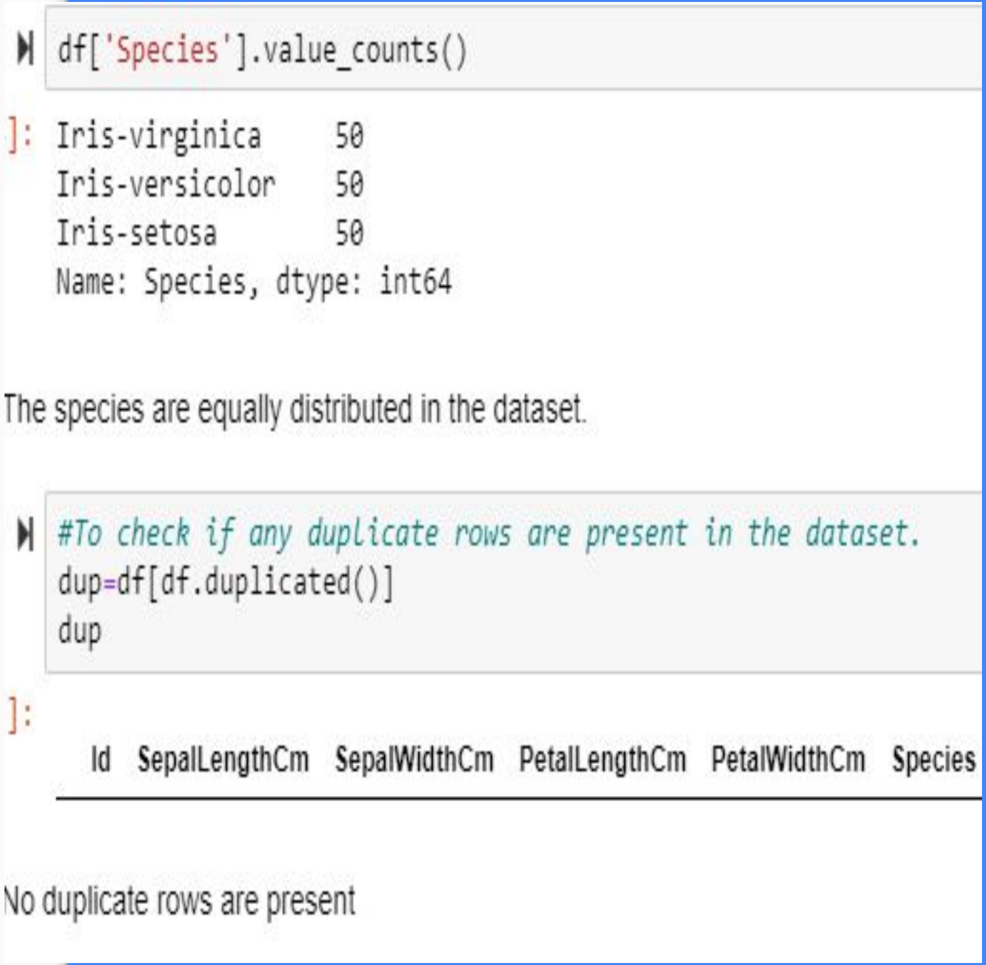


```python
#Checking if the dataset has null values.
df.isnull().sum()
```

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

The dataset has no null values.

The species in dataset are distributed equally and there are no duplicate columns in the dataset.



```
df['Species'].value_counts()
```

```
]: Iris-virginica     50
   Iris-versicolor    50
   Iris-setosa        50
   Name: Species, dtype: int64
```

The species are equally distributed in the dataset.

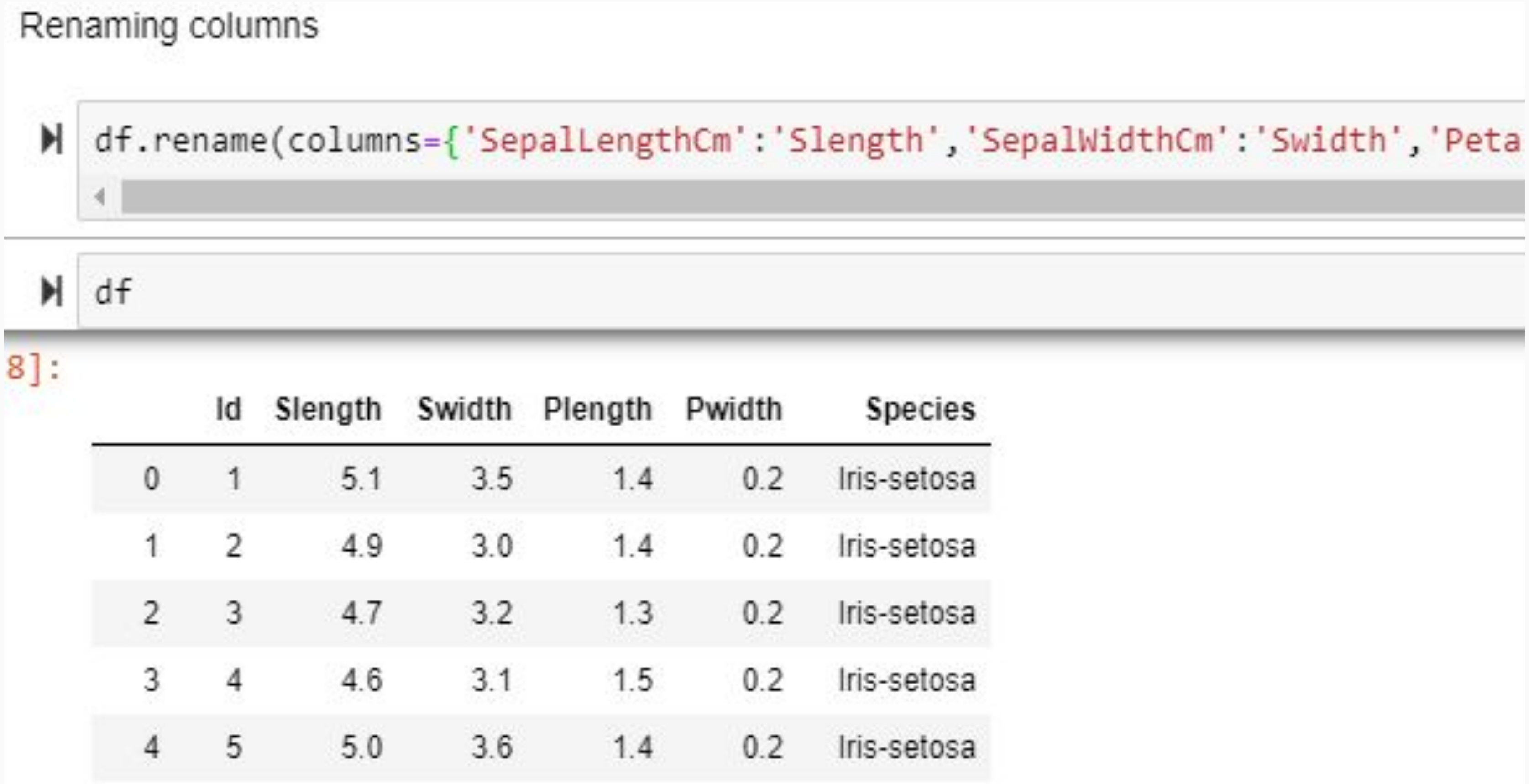
```
#To check if any duplicate rows are present in the dataset.
dup=df[df.duplicated()]
dup
```

]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species

No duplicate rows are present

Renaming columns for convenience.



Renaming columns

```
df.rename(columns={'SepalLengthCm':'Slength','SepalWidthCm':'Swidth','Peta
```

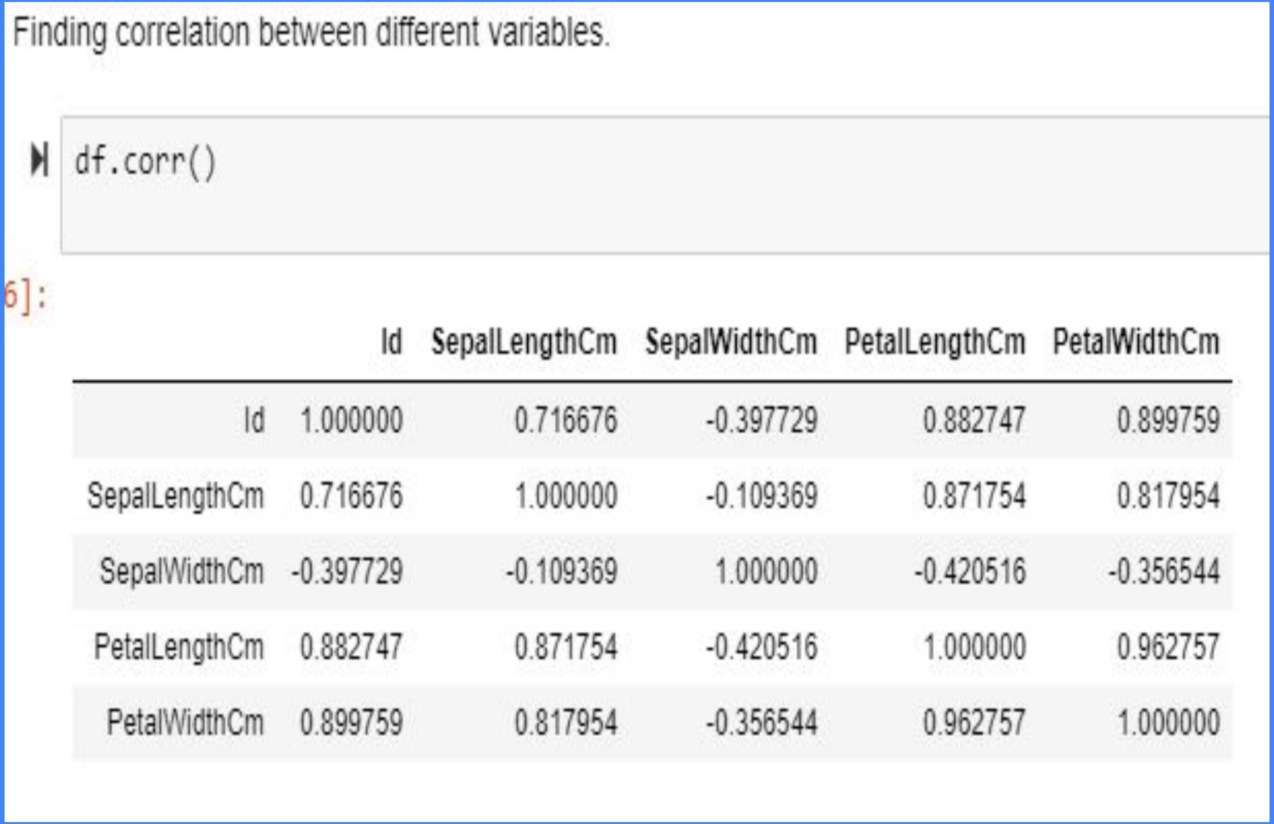
```
df
```

8]:

	Id	Slength	Swidth	Plength	Pwidth	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Finding the correlation between the features:

Values closer to +1 or -1 show a strong positive and negative correlation respectively.



Finding correlation between different variables.

```
df.corr()
```

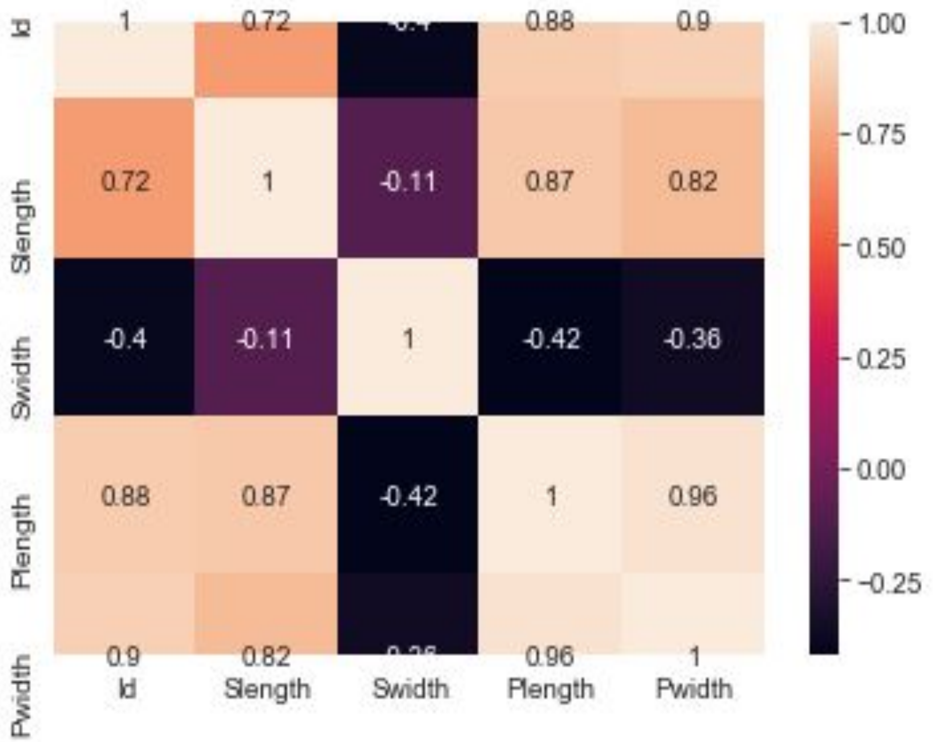
6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

Heatmap displaying the correlation between the different features.

```
c=df.corr()
sns.heatmap(c,annot=True)
```

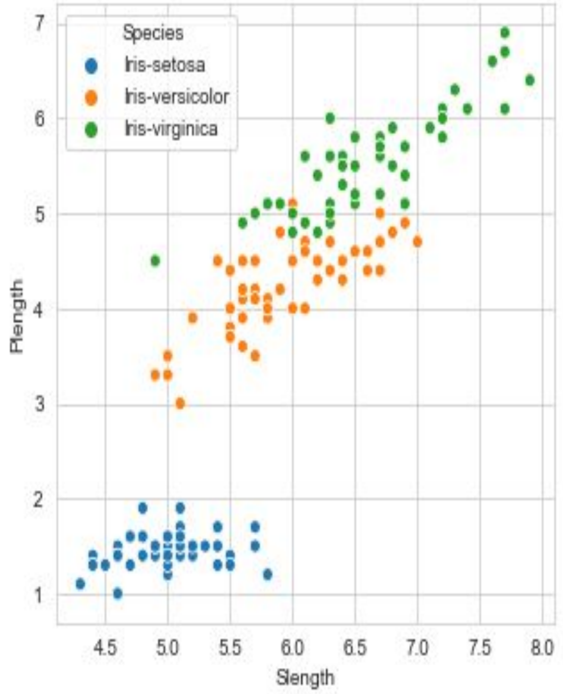
```
<matplotlib.axes._subplots.AxesSubplot at 0x20255694708>
```



Scatter plots to depict the correlation between a pair of variables.

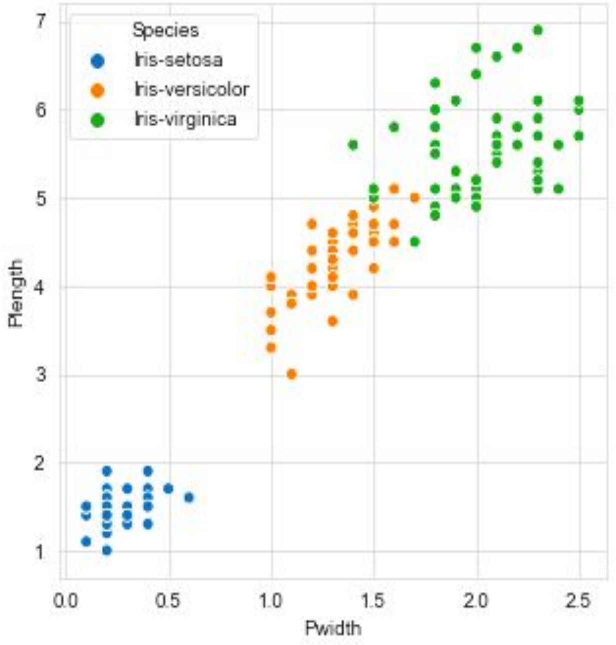
```
plt.figure(figsize=(5,5))
sns.scatterplot(x=df['Slength'],y=df['Plength'],hue=df['Species'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2025993d808>
```



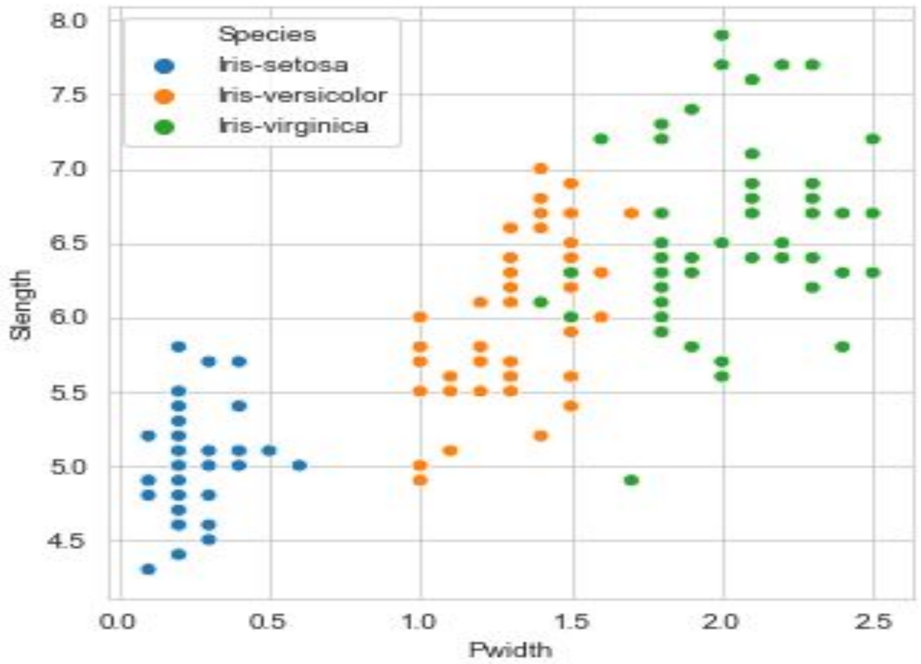
```
sns.set_style("whitegrid")
plt.figure(figsize=(5,5))
sns.scatterplot(x=df['Pwidth'],y=df['Plength'],hue=df['Species'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20259a83808>
```



```
sns.set_style("whitegrid")
plt.figure(figsize=(5,5))
sns.scatterplot(x=df['Pwidth'],y=df['Slength'],hue=df['Species'])
```

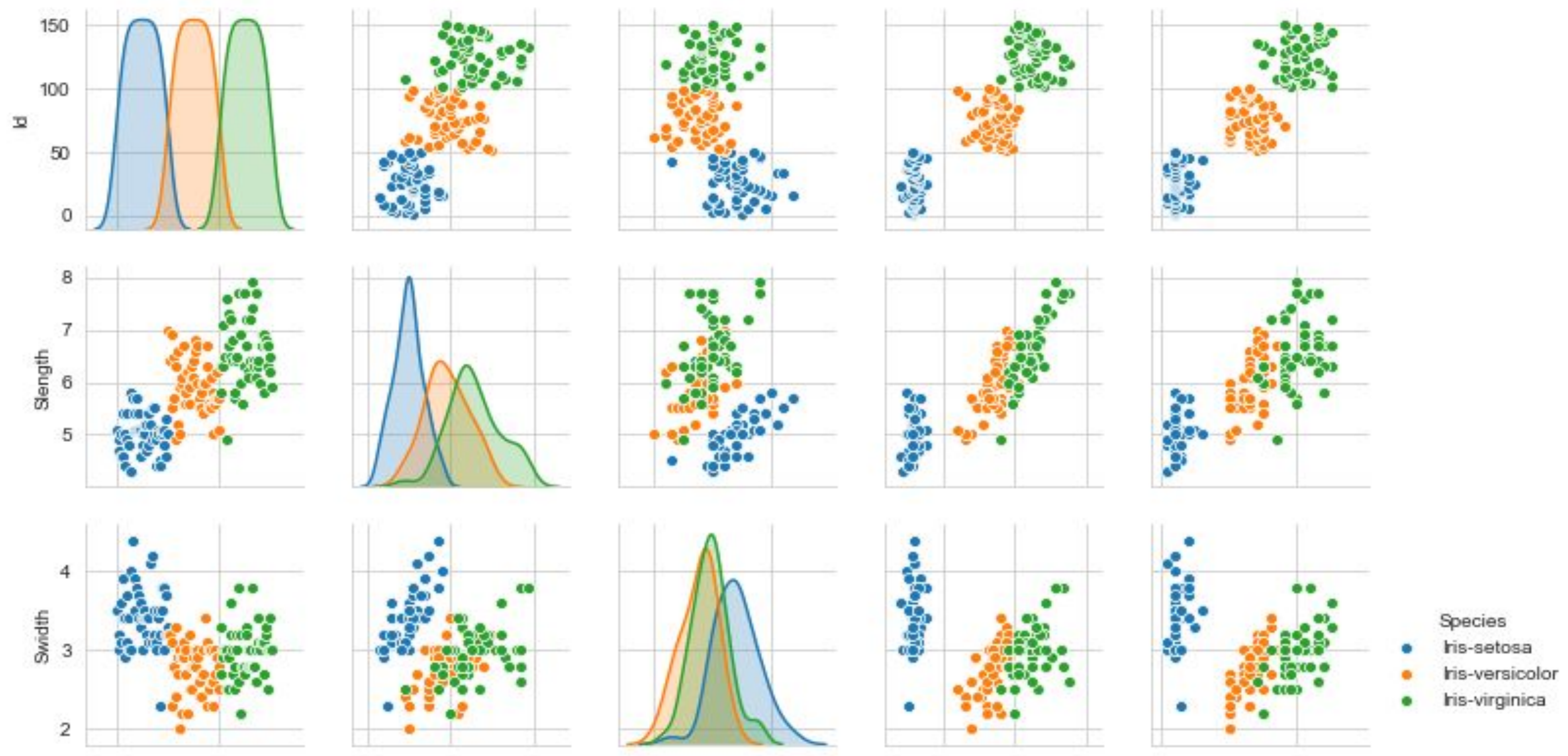
```
<matplotlib.axes._subplots.AxesSubplot at 0x20259ca1bc8>
```

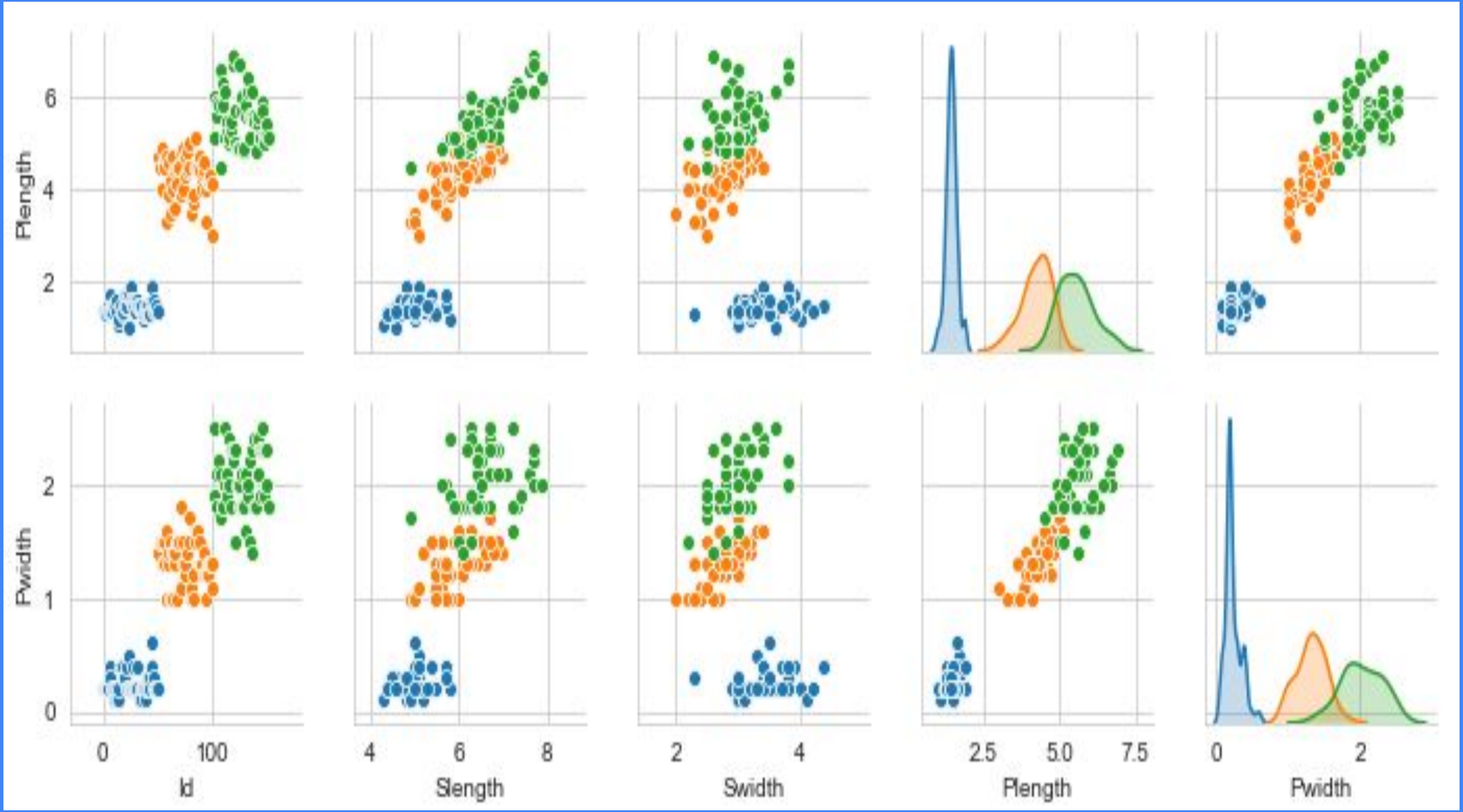


Pair plot depicting a pairwise relationship between the features.

```
sns.pairplot(data=df,hue='Species',height=2)
```

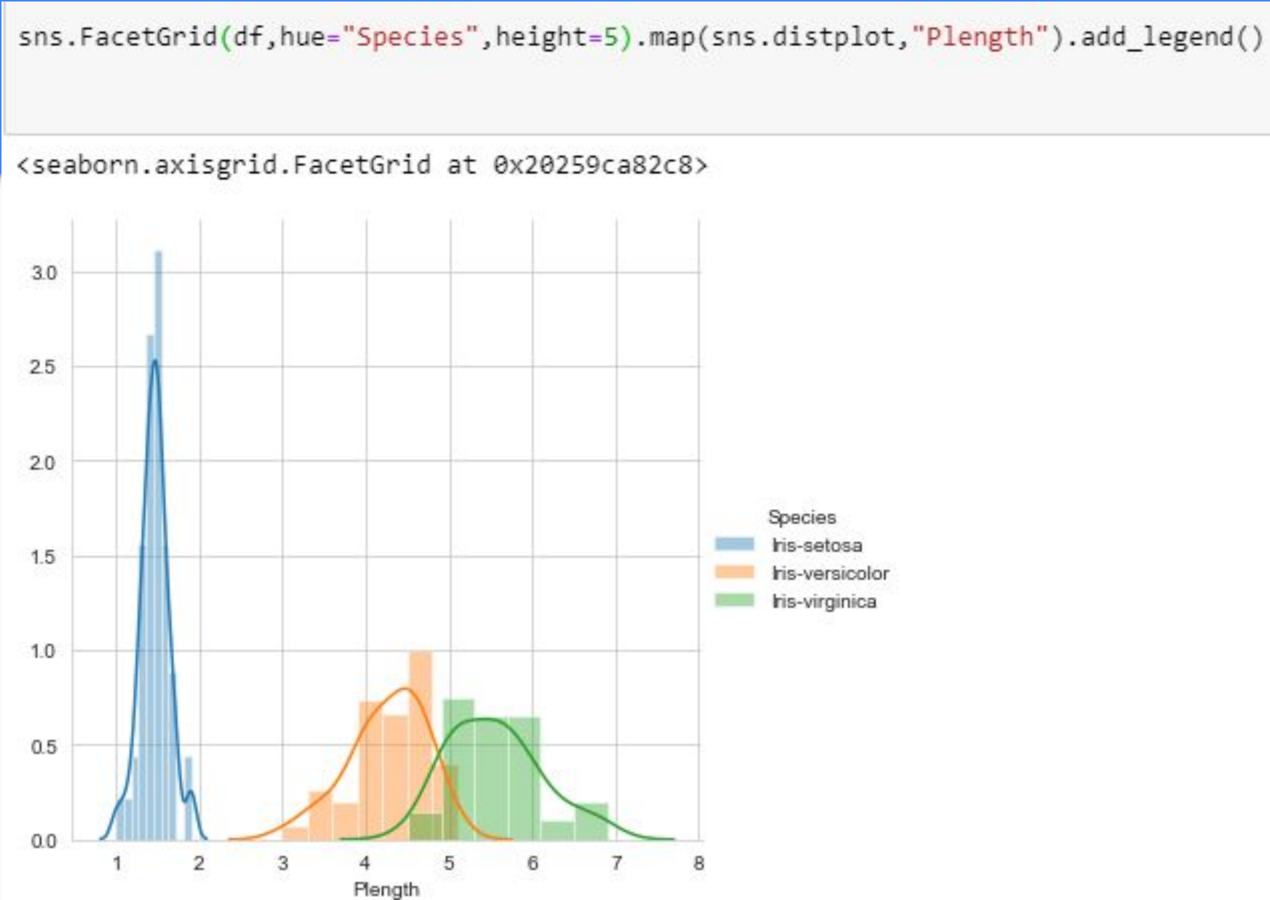
```
<seaborn.axisgrid.PairGrid at 0x2025c208ec8>
```




Plength
Pwidth
Id
Slength
Swidth
Plength
Pwidth
0
100
4
6
8
2
4
2.5
5.0
7.5
0
2
2
4
6
0
1
2

Histograms

Histograms show the probability density function of the features Petal width and petal length with respect to the species.



```
sns.FacetGrid(df,hue="Species",height=5).map(sns.distplot,"Plength").add_legend()
```

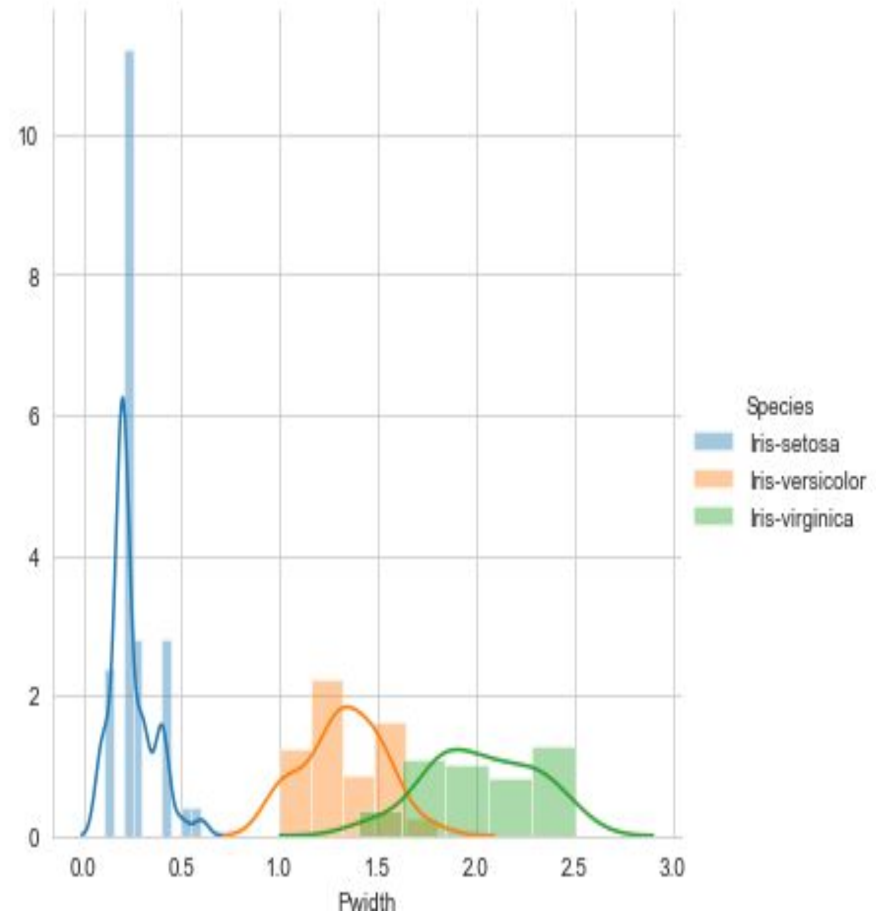
```
<seaborn.axisgrid.FacetGrid at 0x20259ca82c8>
```

From the plots we can see that iris setosa can be separated from the other two species using Petal length and Petal width.

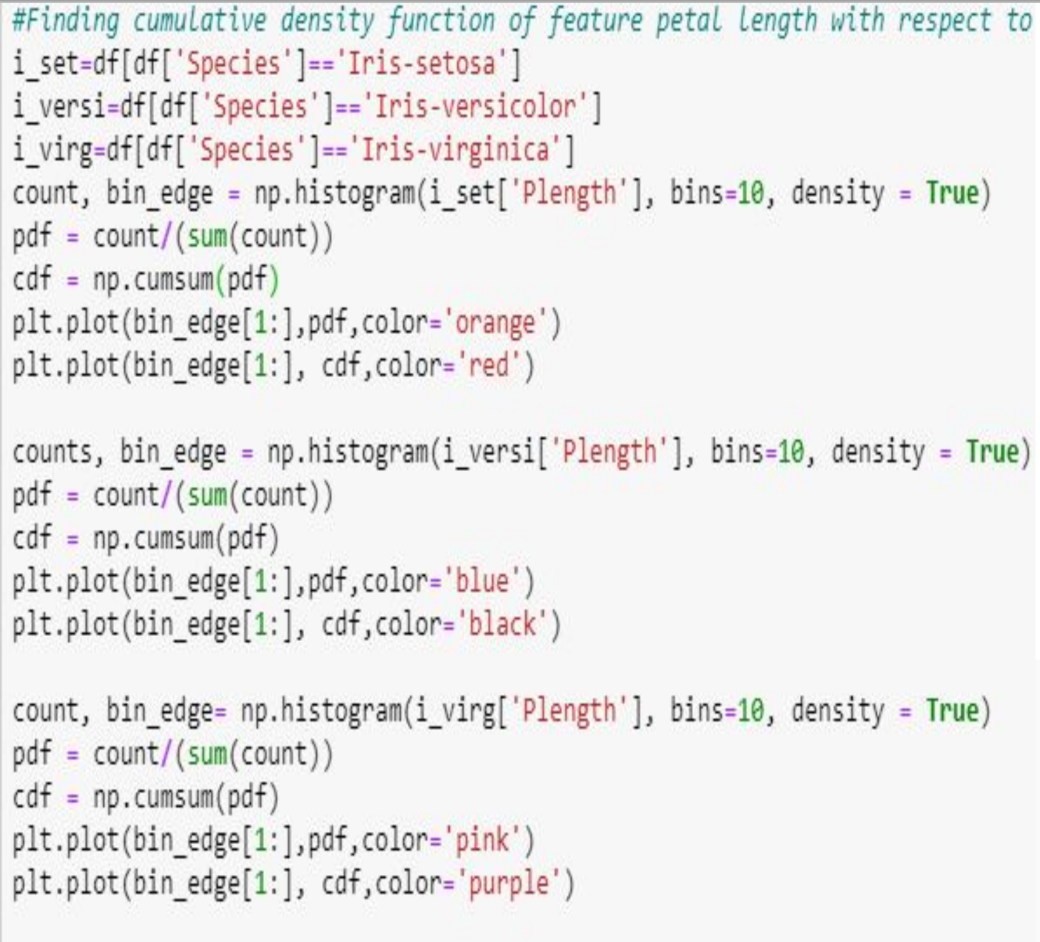
Petal length is slightly better than petal width because the distributions are better separated.

```
sns.FacetGrid(df,hue="Species",height=5).map(sns.distplot,"Pwidth").add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x20259ead088>
```

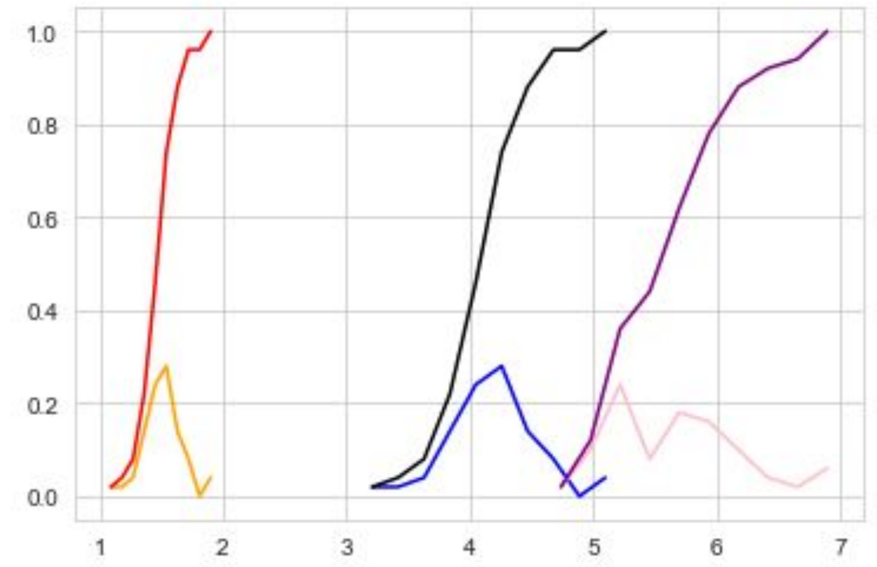


Cumulative Density Function

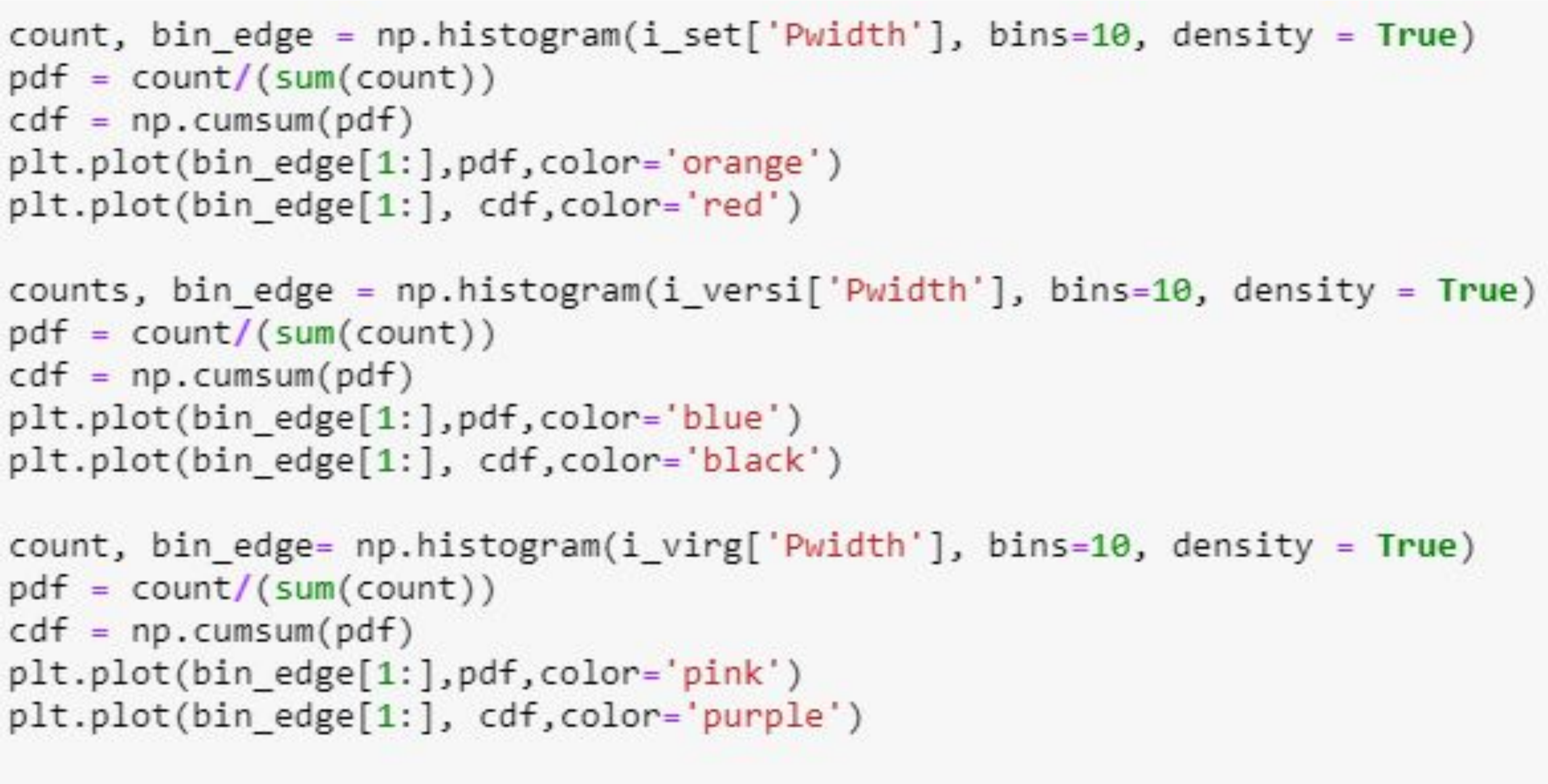


```
#Finding cumulative density function of feature petal length with respect to
i_set=df[df['Species']=='Iris-setosa']
i_versi=df[df['Species']=='Iris-versicolor']
i_virg=df[df['Species']=='Iris-virginica']
count, bin_edge = np.histogram(i_set['Plength'], bins=10, density = True)
pdf = count/(sum(count))
cdf = np.cumsum(pdf)
plt.plot(bin_edge[1:],pdf,color='orange')
plt.plot(bin_edge[1:], cdf,color='red')

counts, bin_edge = np.histogram(i_versi['Plength'], bins=10, density = True)
pdf = count/(sum(count))
cdf = np.cumsum(pdf)
plt.plot(bin_edge[1:],pdf,color='blue')
plt.plot(bin_edge[1:], cdf,color='black')

count, bin_edge= np.histogram(i_virg['Plength'], bins=10, density = True)
pdf = count/(sum(count))
cdf = np.cumsum(pdf)
plt.plot(bin_edge[1:],pdf,color='pink')
plt.plot(bin_edge[1:], cdf,color='purple')
```

```
[<matplotlib.lines.Line2D at 0x20258da6908>]
```



Finding cumulative density function of feature petal length with respect to the species to see what percent of flowers fall in which species if we consider their petal lengths



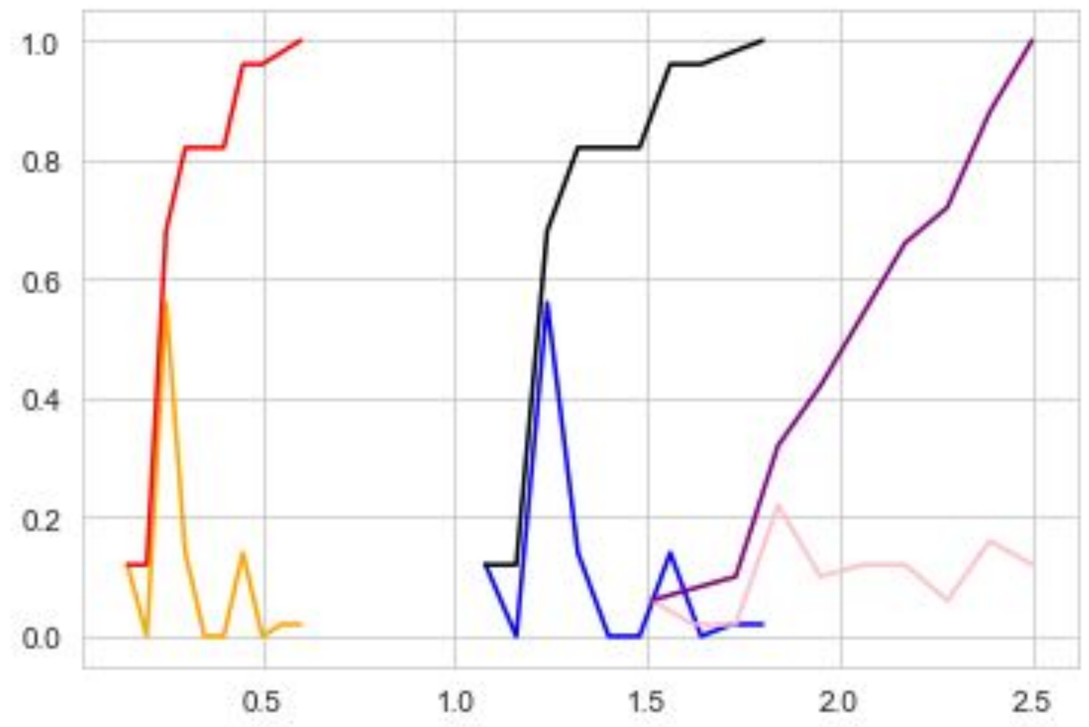
```
count, bin_edge = np.histogram(i_set['Pwidth'], bins=10, density = True)
pdf = count/(sum(count))
cdf = np.cumsum(pdf)
plt.plot(bin_edge[1:],pdf,color='orange')
plt.plot(bin_edge[1:], cdf,color='red')

counts, bin_edge = np.histogram(i_versi['Pwidth'], bins=10, density = True)
pdf = count/(sum(count))
cdf = np.cumsum(pdf)
plt.plot(bin_edge[1:],pdf,color='blue')
plt.plot(bin_edge[1:], cdf,color='black')

count, bin_edge= np.histogram(i_virg['Pwidth'], bins=10, density = True)
pdf = count/(sum(count))
cdf = np.cumsum(pdf)
plt.plot(bin_edge[1:],pdf,color='pink')
plt.plot(bin_edge[1:], cdf,color='purple')
```

Cdf using petal width

From the CDF of petal length it is clear that more than 95 percent of versicolor flowers have petal length less than or equal to 5. Hence we can somewhat distinguish the two species using this information.

Cdf using petal width has some overlap after the 1.5cm region,hence petal length is better.

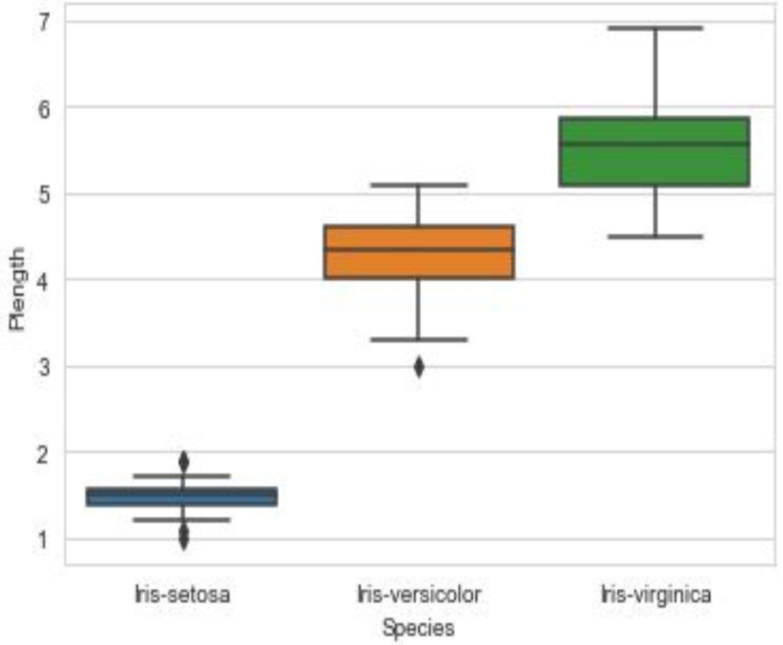
```
[<matplotlib.lines.Line2D at 0x202593931c8>]
```



Box plots representing the median and quartiles of the data.

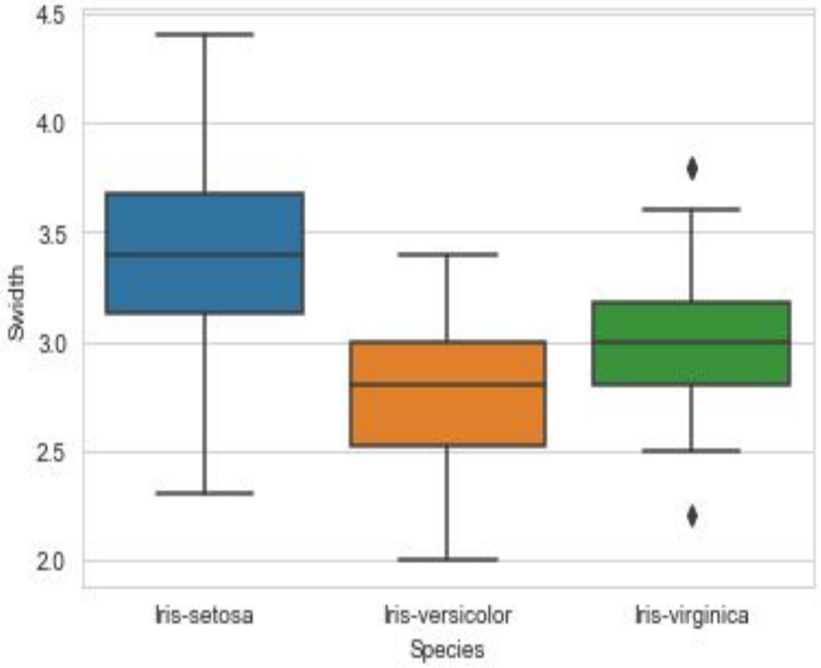
```
sns.boxplot(x="Species",y="Plength", data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20258dc9c88>
```



```
sns.boxplot(x="Species",y="Swidth", data=df)
```

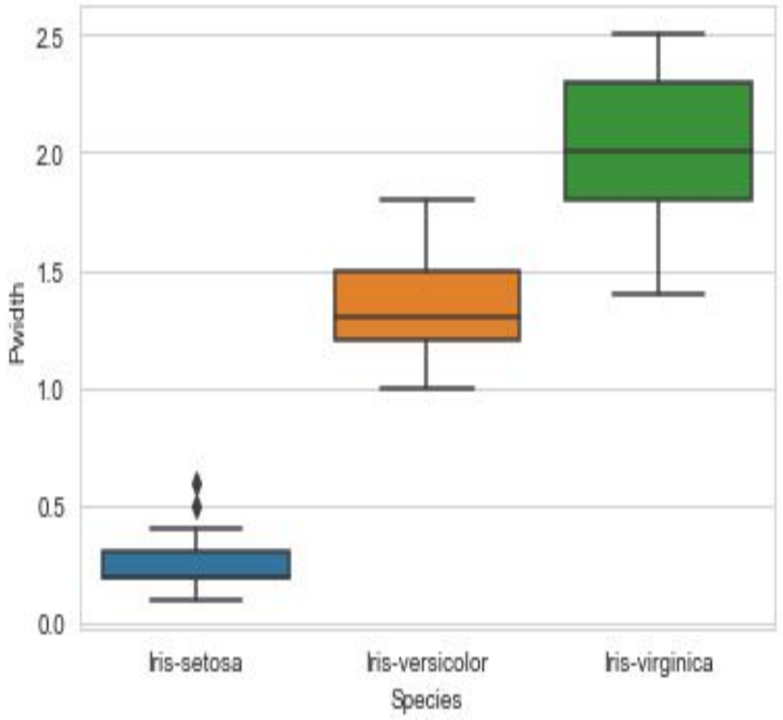
```
<matplotlib.axes._subplots.AxesSubplot at 0x20258e924
```



Species wise box plot of Petal length and Sepal width

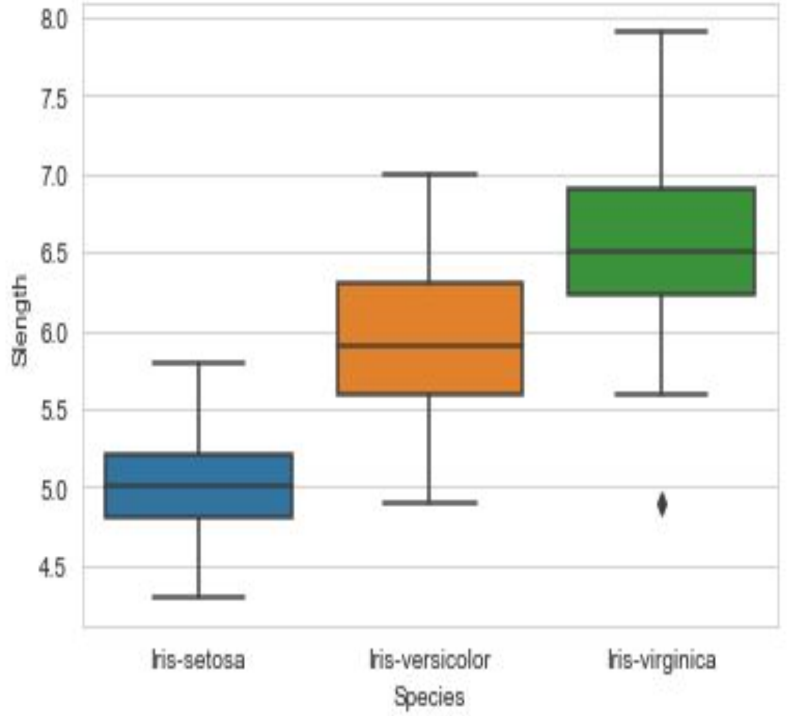
```
sns.boxplot(x="Species",y="Pwidth", data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20258f13c88>
```



```
sns.boxplot(x="Species",y="Slength", data=df)
```

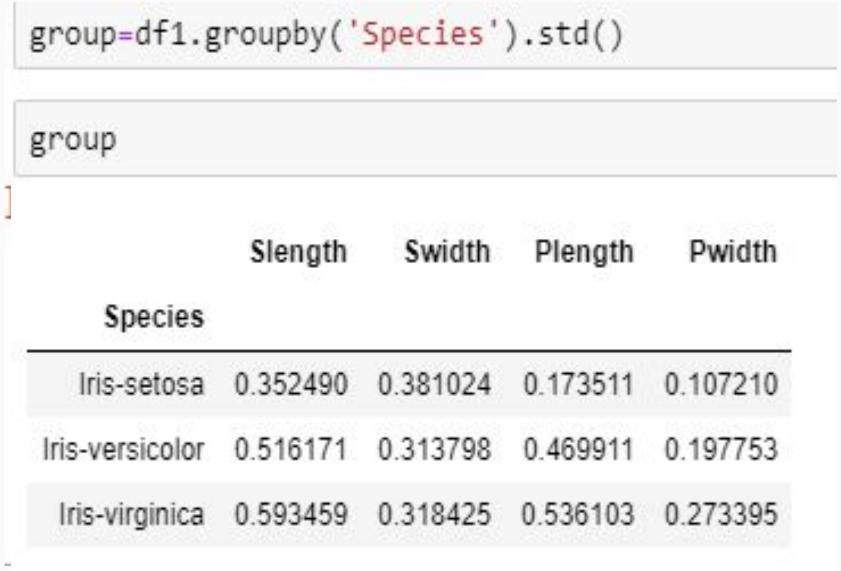
```
<matplotlib.axes._subplots.AxesSubplot at 0x202590264c8>
```



Species wise box plot of Petal width and Sepal length

Species wise means and standard deviation of the features.

Mean

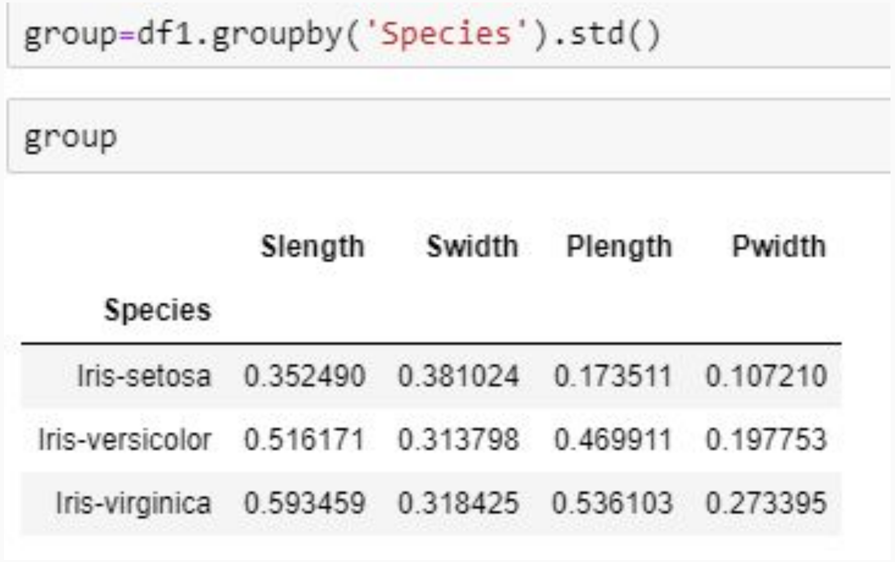


```
group=df1.groupby('Species').std()
```

```
group
```

Species	Slength	Swidth	Plength	Pwidth
Iris-setosa	0.352490	0.381024	0.173511	0.107210
Iris-versicolor	0.516171	0.313798	0.469911	0.197753
Iris-virginica	0.593459	0.318425	0.536103	0.273395

Standard Deviation



```
group=df1.groupby('Species').std()
```

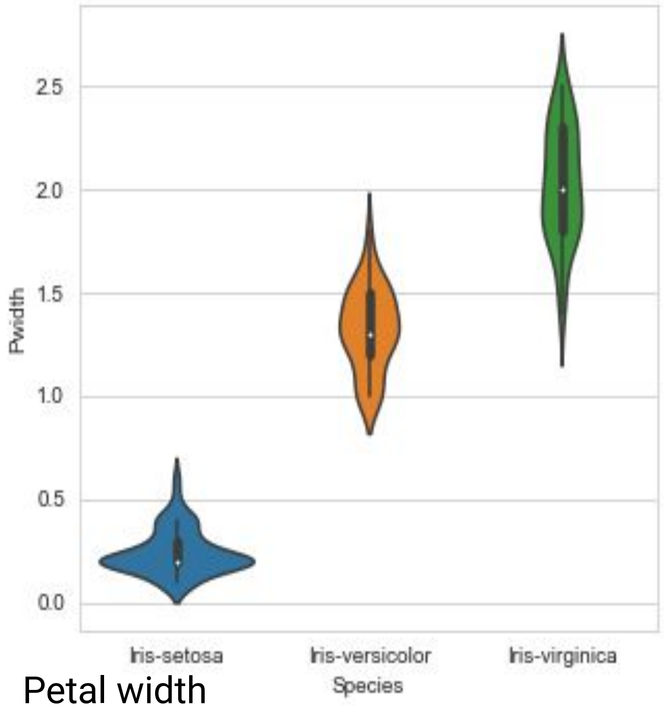
```
group
```

Species	Slength	Swidth	Plength	Pwidth
Iris-setosa	0.352490	0.381024	0.173511	0.107210
Iris-versicolor	0.516171	0.313798	0.469911	0.197753
Iris-virginica	0.593459	0.318425	0.536103	0.273395

Petal length and petal width of iris setosa is lesser than the other two species

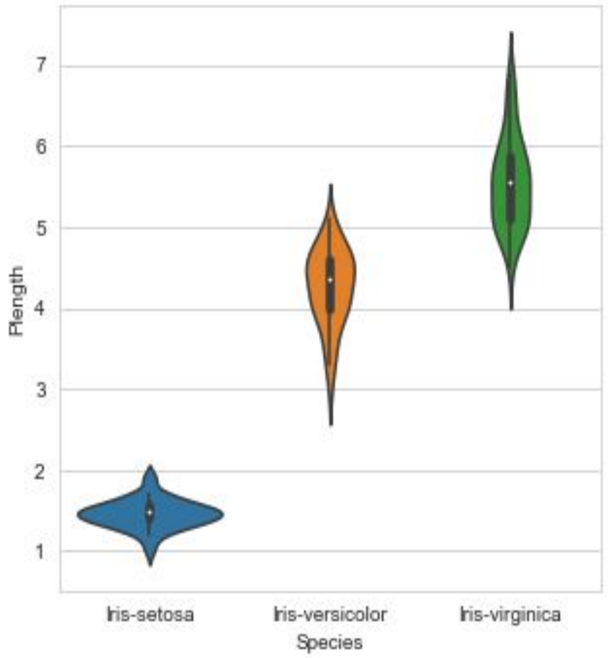
Violin plots that represent the quartiles and pdfs of the features.

```
plt.figure(figsize=(5,5))
sns.violinplot(x="Species", y="Pwidth", data=df, height=10)
plt.show()
```



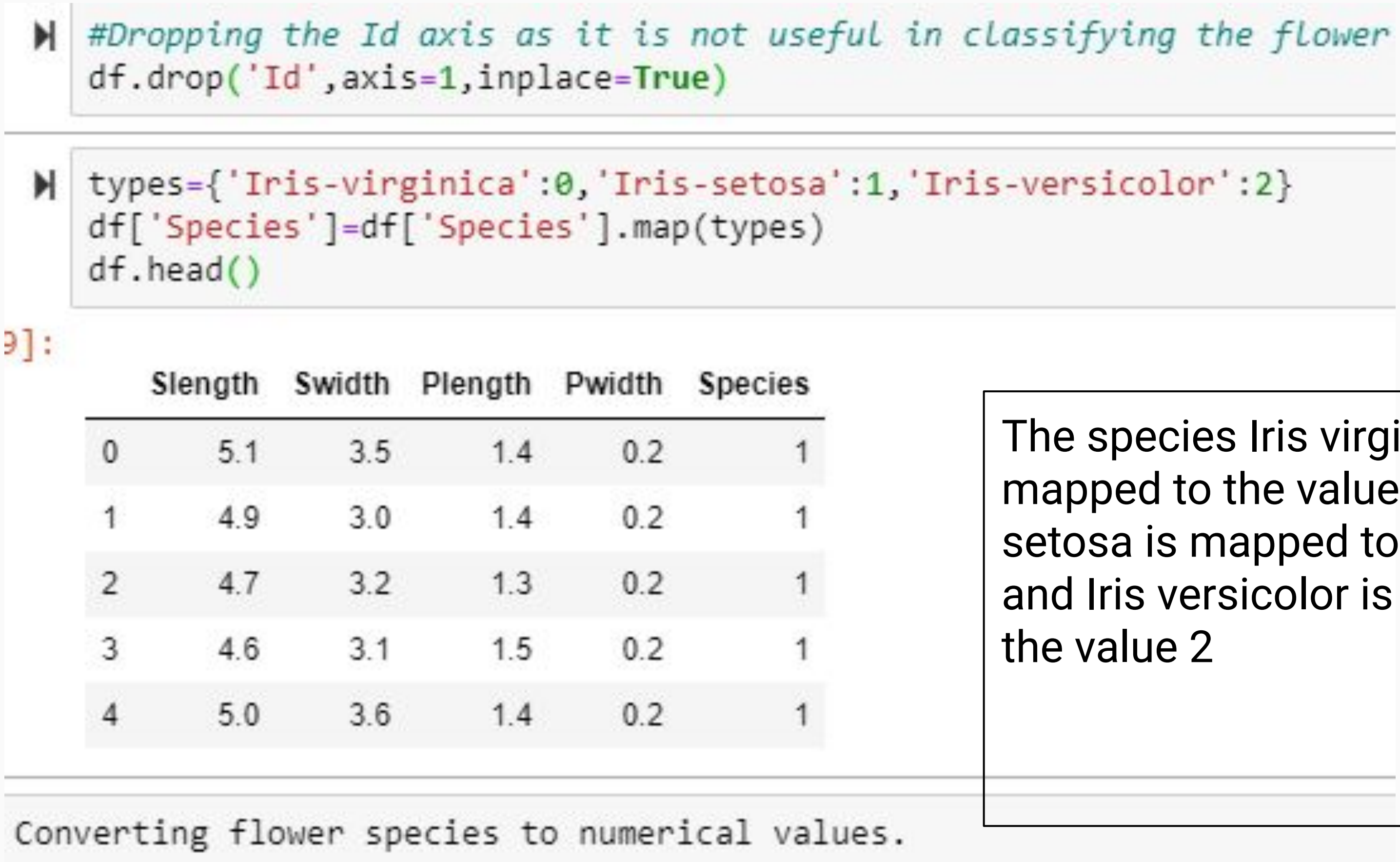
Petal width

```
plt.figure(figsize=(5,5))
sns.violinplot(x="Species", y="Plength", data=df, height=10)
plt.show()
```



Petal length

Dropping ID column as it is not useful in flower classification and converting the flower types into numerical values.



```
#Dropping the Id axis as it is not useful in classifying the flower
df.drop('Id',axis=1,inplace=True)
```

```
types={'Iris-virginica':0,'Iris-setosa':1,'Iris-versicolor':2}
df['Species']=df['Species'].map(types)
df.head()
```

```
9]:
```

	Slength	Swidth	Plength	Pwidth	Species
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

```
Converting flower species to numerical values.
```

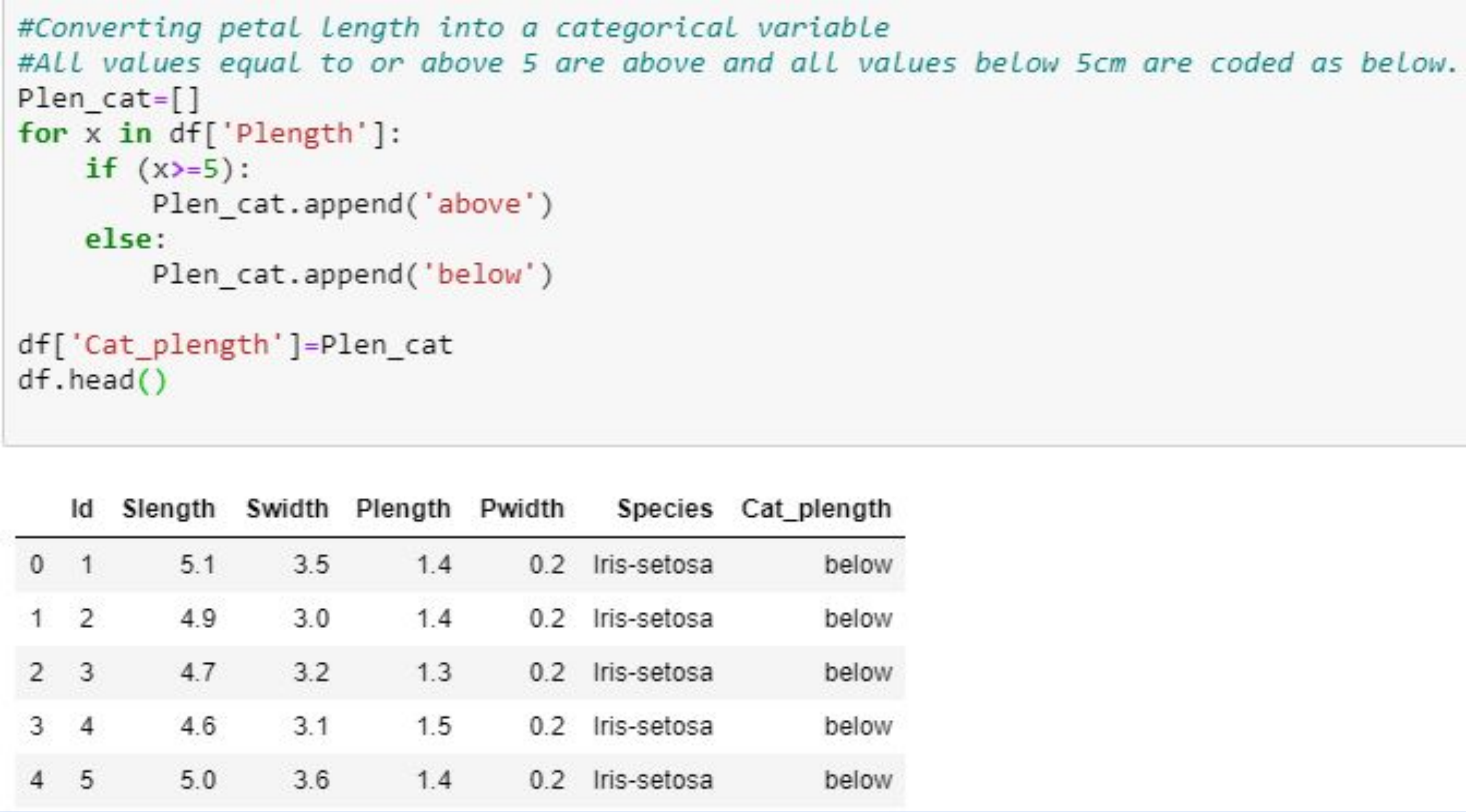
The species Iris virginica is mapped to the value 0,Iris setosa is mapped to the value 1 and Iris versicolor is mapped to the value 2

Hypothesis Testing

Hypothesis -Flowers with petal lengths more than 5cm are Iris Virginica and hence petal length can differentiate between the flowers species.If we convert petal length into a categorical variable such that flowers with petal length more than or equal to 5cm are labelled as above and those with less than 5 are labelled below, then these two variables will be highly correlated.

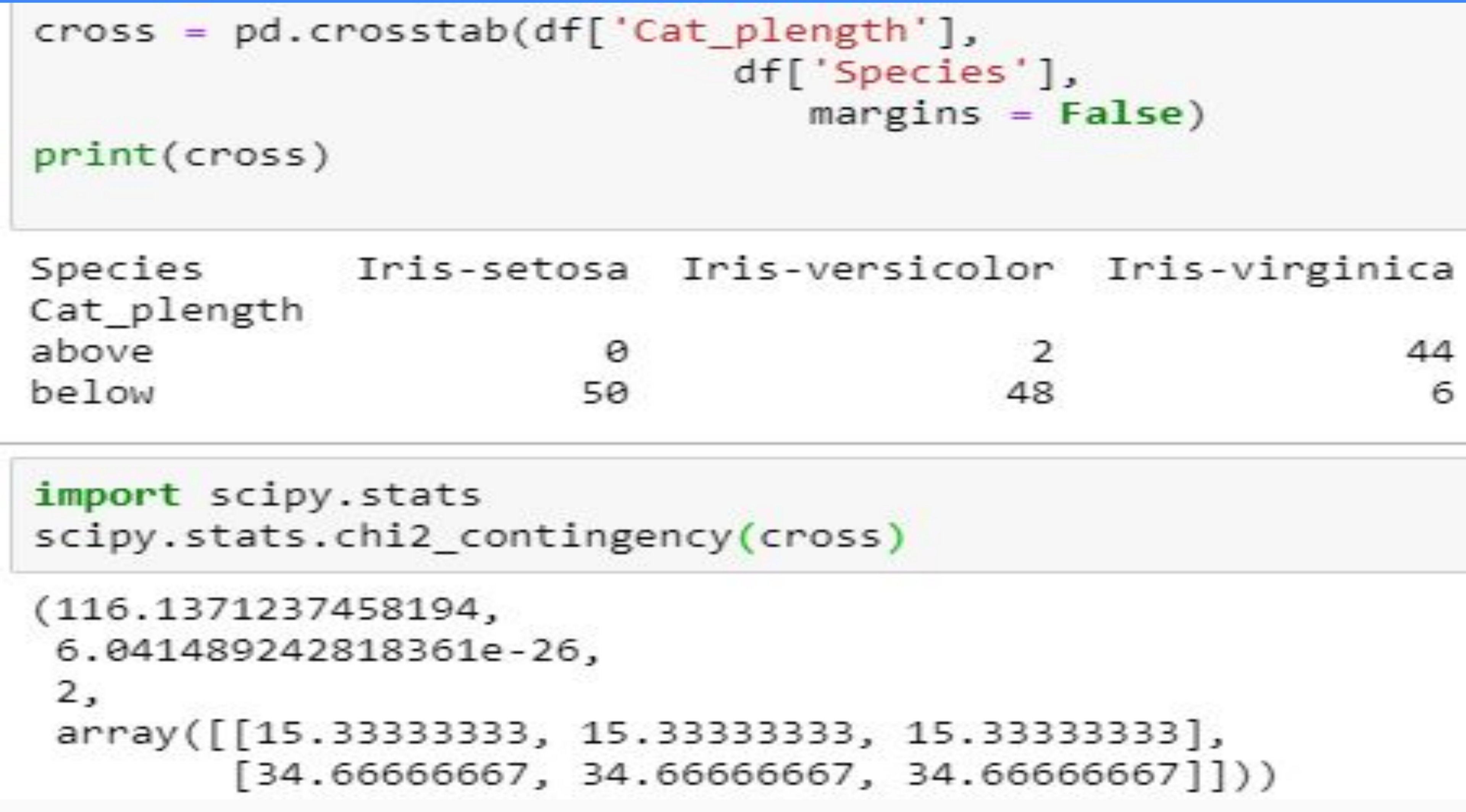
Null Hypothesis-Categorical petal length and species are independent.

Alternate hypothesis- Categorical petal length and species variables are dependent.



```python
#Converting petal length into a categorical variable
#All values equal to or above 5 are above and all values below 5cm are coded as below.
Plen_cat=[]
for x in df['Plength']:
    if (x>=5):
        Plen_cat.append('above')
    else:
        Plen_cat.append('below')

df['Cat_plength']=Plen_cat
df.head()
```

	Id	Slength	Swidth	Plength	Pwidth	Species	Cat_plength
0	1	5.1	3.5	1.4	0.2	Iris-setosa	below
1	2	4.9	3.0	1.4	0.2	Iris-setosa	below
2	3	4.7	3.2	1.3	0.2	Iris-setosa	below
3	4	4.6	3.1	1.5	0.2	Iris-setosa	below
4	5	5.0	3.6	1.4	0.2	Iris-setosa	below



```
cross = pd.crosstab(df['Cat_plength'],
                             df['Species'],
                               margins = False)
print(cross)
```

```
Species       Iris-setosa  Iris-versicolor  Iris-virginica
Cat_plength
above                   0                2              44
below                  50               48               6
```

```
import scipy.stats
scipy.stats.chi2_contingency(cross)
```

```
(116.1371237458194,
 6.041489242818361e-26,
 2,
 array([[15.33333333, 15.33333333, 15.33333333],
        [34.66666667, 34.66666667, 34.66666667]]))
```

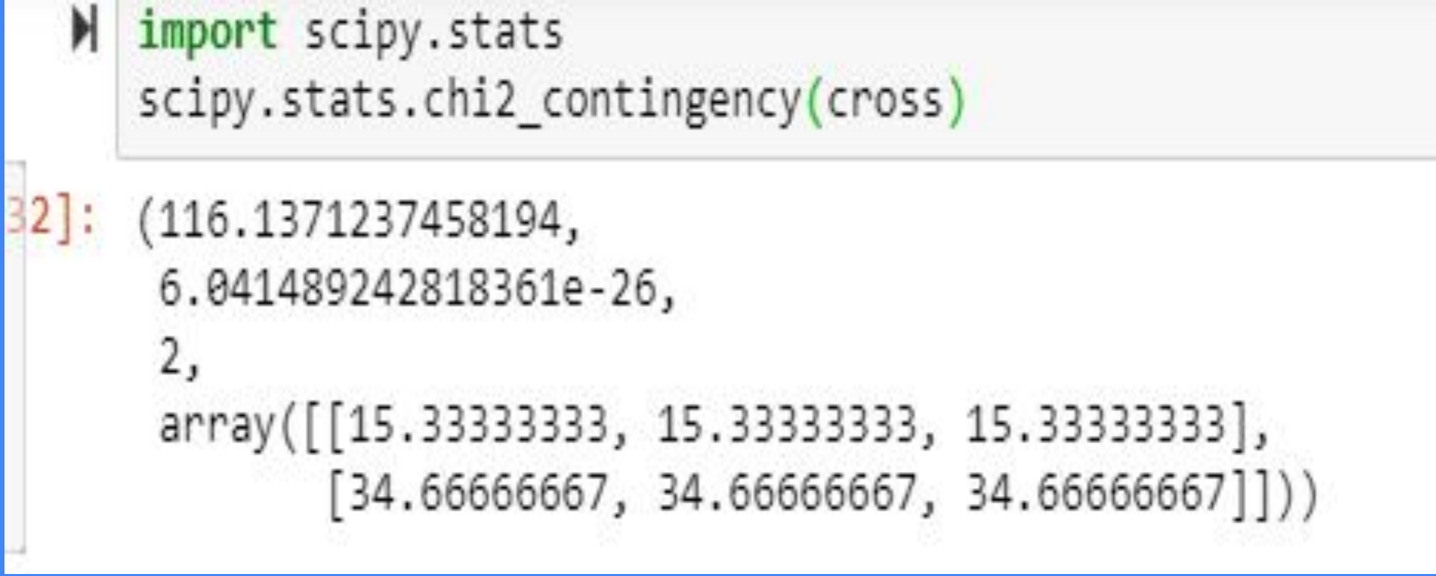
The function returns the following:

chi2 : float
The test statistic.

p : float
The p-value of the test

dof : int
Degrees of freedom

expected : ndarray, same shape as `observed`
The expected frequencies, based on the marginal sums of the table.



```
import scipy.stats
scipy.stats.chi2_contingency(cross)
```

```
32]: (116.1371237458194,
      6.041489242818361e-26,
      2,
      array([[15.33333333, 15.33333333, 15.33333333],
             [34.66666667, 34.66666667, 34.66666667]]))
```

Hence from the output it is visible that the p-value 6.041489242818361e-26 is much less than the confidence interval of 0.05 hence we can reject the null hypothesis.
That is our alternate hypothesis is true.

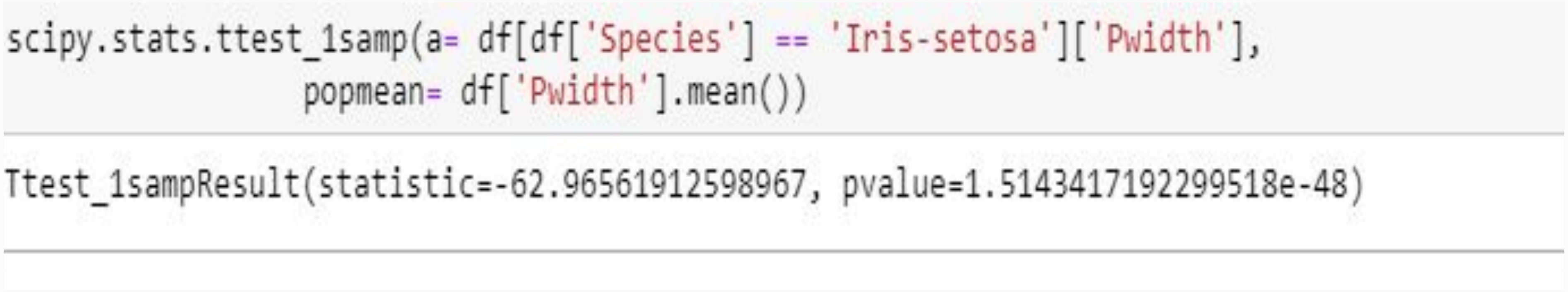
Hypothesis-2

The mean of Petal width of Iris setosa species is lesser ie different from the mean of the entire sample Petal width.

Null hypothesis-Mean petal width of iris setosa flowers is nearly equal to that of the entire data set's petal width.

Alternate Hypothesis-Mean petal width of iris setosa flowers is considerably different from that of the entire data set's petal width.

Using T-test score



```
scipy.stats.ttest_1samp(a= df[df['Species'] == 'Iris-setosa']['Pwidth'],
                        popmean= df['Pwidth'].mean())
```

```
Ttest_1sampResult(statistic=-62.96561912598967, pvalue=1.5143417192299518e-48)
```

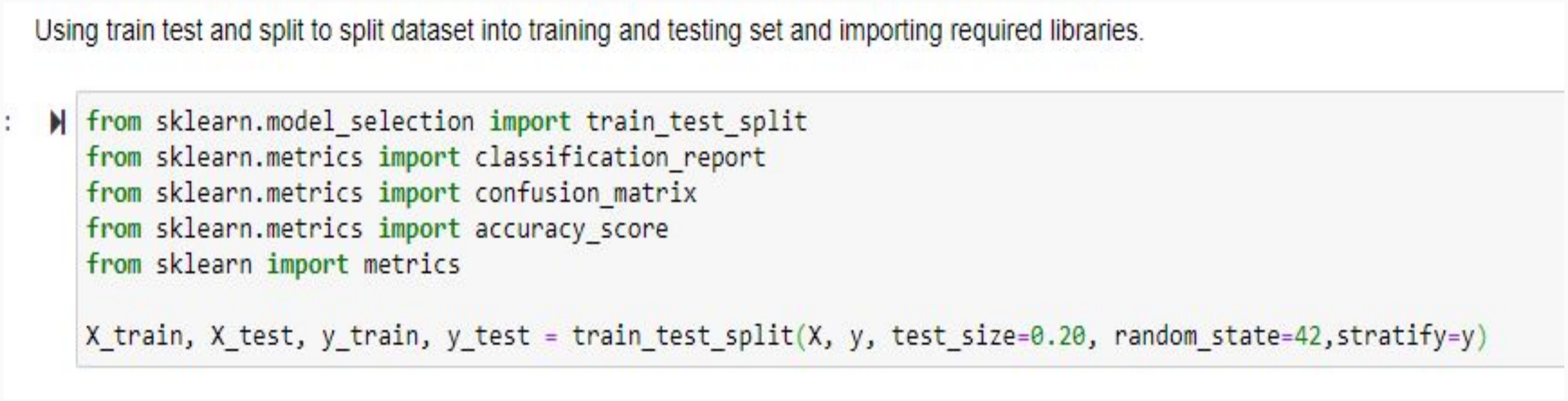
The small p value indicates that we can reject our null hypothesis, ie. petal width of Iris setosa flowers is different from the petal width mean.

Conclusion-

The EDA helps us distinguish the flowers based on the four features -Petal length,width and Sepal length and width to some extent.

Building Prediction Models:

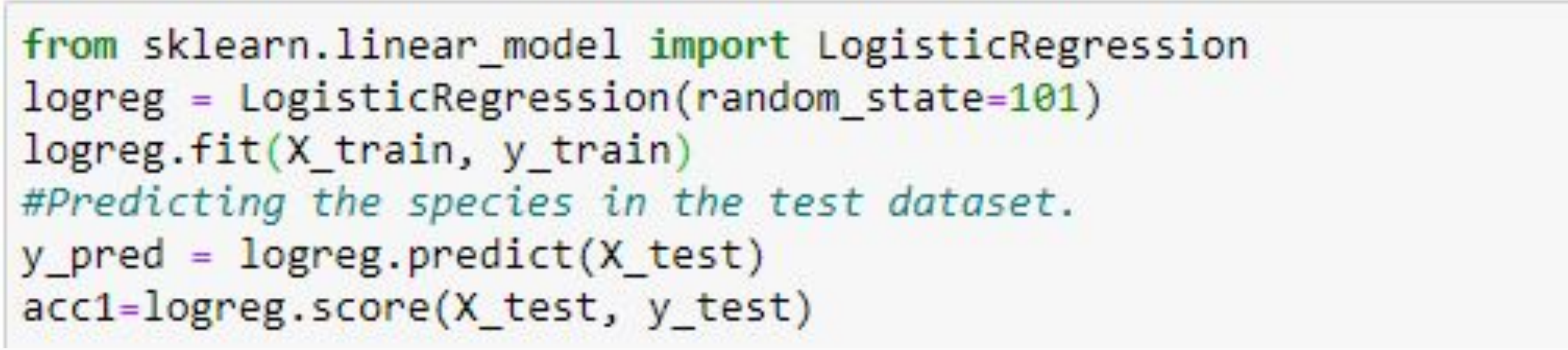
Importing required libraries and splitting data into training and testing set using sklearn



Using train test and split to split dataset into training and testing set and importing required libraries.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import metrics

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42,stratify=y)
```

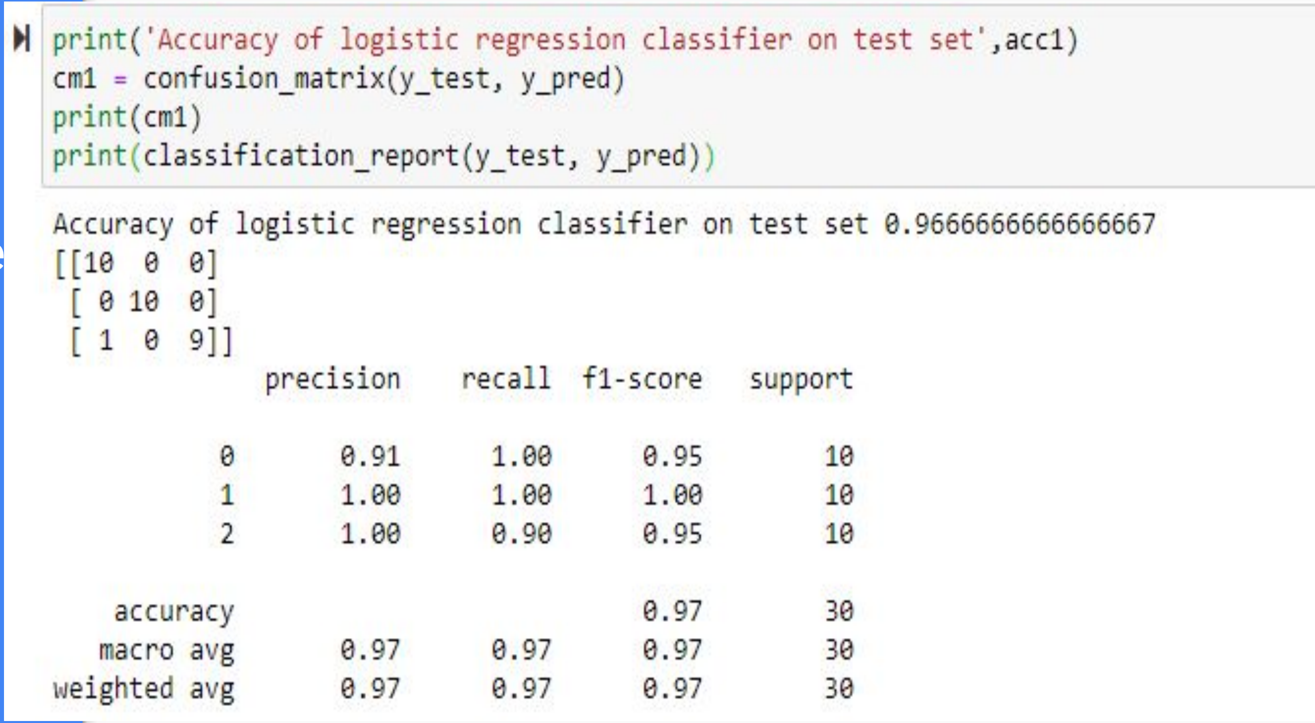
Building a logistic regression model :



```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state=101)
logreg.fit(X_train, y_train)
#Predicting the species in the test dataset.
y_pred = logreg.predict(X_test)
acc1=logreg.score(X_test, y_test)
```

Predictions

Using logistic regression algorithm,the model is able to predict the species with 97% accuracy.



```
print('Accuracy of logistic regression classifier on test set',acc1)
cm1 = confusion_matrix(y_test, y_pred)
print(cm1)
print(classification_report(y_test, y_pred))
```

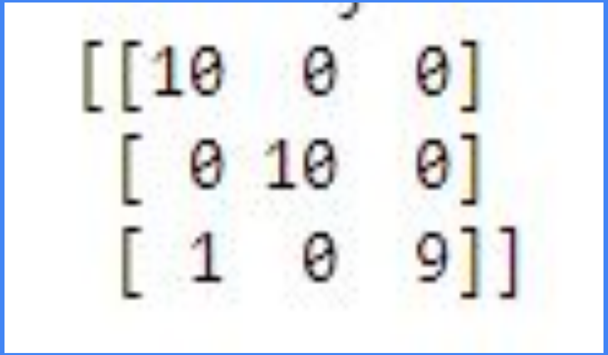
```
Accuracy of logistic regression classifier on test set 0.9666666666666667
[[10  0  0]
 [ 0 10  0]
 [ 1  0  9]]
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        10
           1       1.00      1.00      1.00        10
           2       1.00      0.90      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

Understanding the confusion matrix

Based on the 3x3 confusion matrix the columns are the predictions and the rows are the actual values. The main diagonal (10,10,9) gives the correct predictions. That is, the cases where the actual values and the model predictions are the same.The first row are the actual setosa flowers. The model predicted 10 of these correctly and didn’t predict any incorrectly..

Looking at the setosa column, of the 11 setosa flowers predicted by the model (sum of column Setosa), 10 were actually setosa flowers, while 0 where Versicolor incorrectly predicted to be Setosa and 1 was Virginica incorrectly predicted to be Setosa .

Similarly we can analyze the other columns.



$$\begin{array}{cccc} & \text{Setosa} & \text{Versicolor} & \text{Virginica} \\ \text{Setosa} & 10 & 0 & 0 \\ \text{Versicolor} & 0 & 10 & 0 \\ \text{Virginica} & 1 & 0 & 9 \end{array}$$

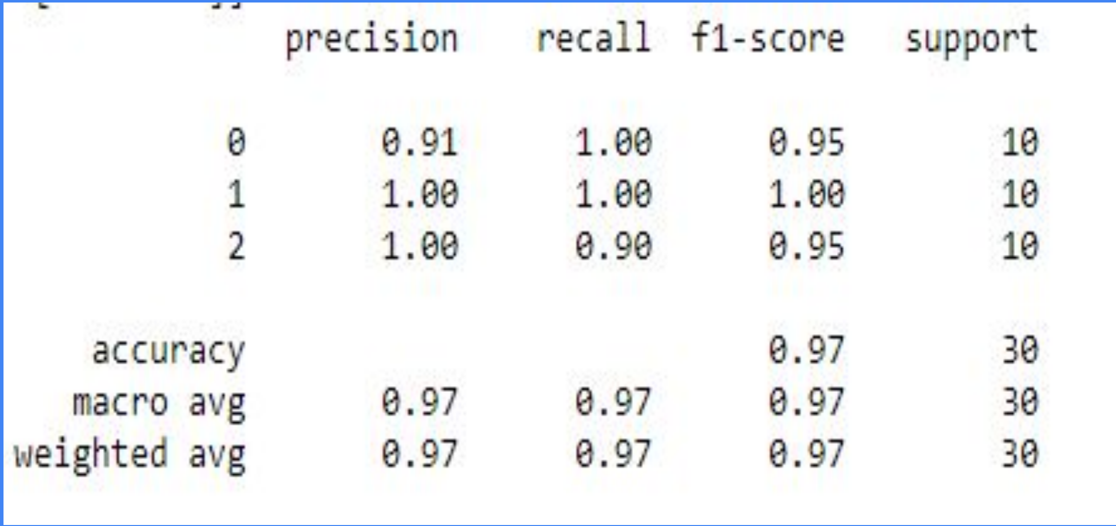
```
[[10  0  0]
 [ 0 10  0]
 [ 1  0  9]]
```

Precision measures how good the model is at assigning positive events to the positive class. That is, how accurate the species detection is

.

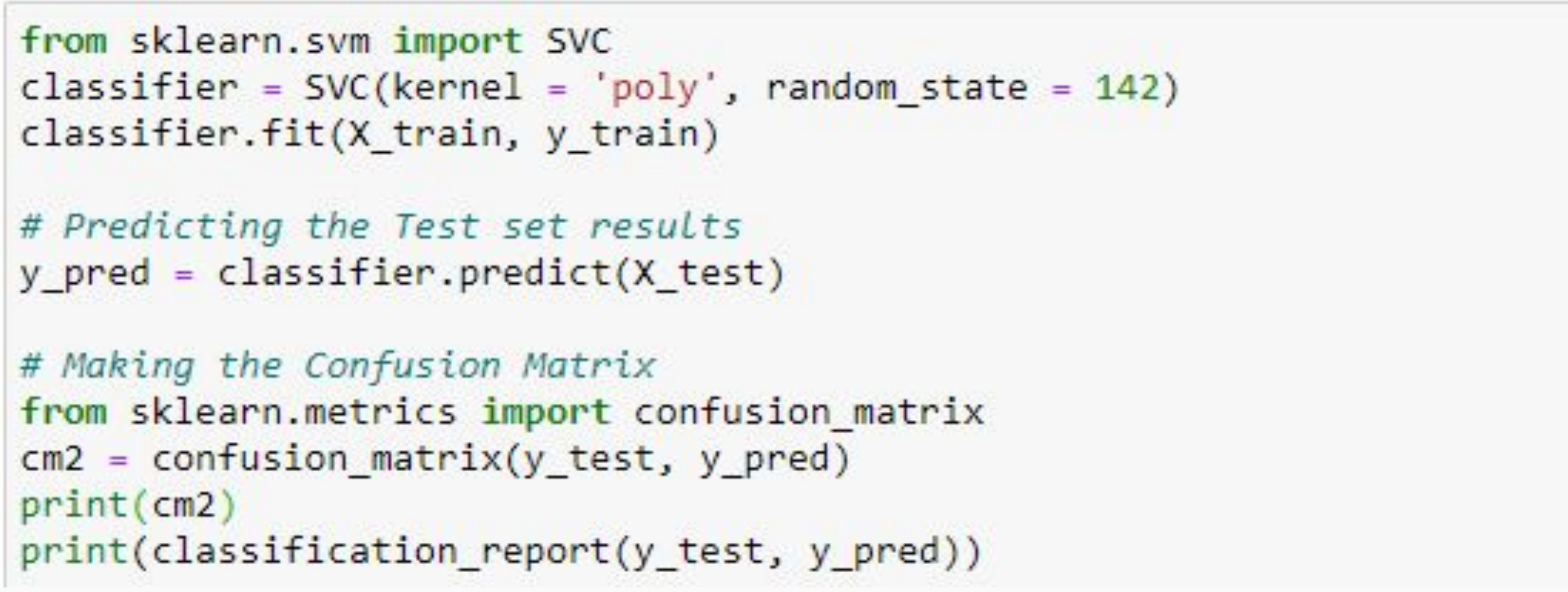
Recall measures how good the model is in detecting positive events.

Recall and precision can be reported by a measure that combines them. One example is called F-measure, which is the harmonic mean of recall and precision.



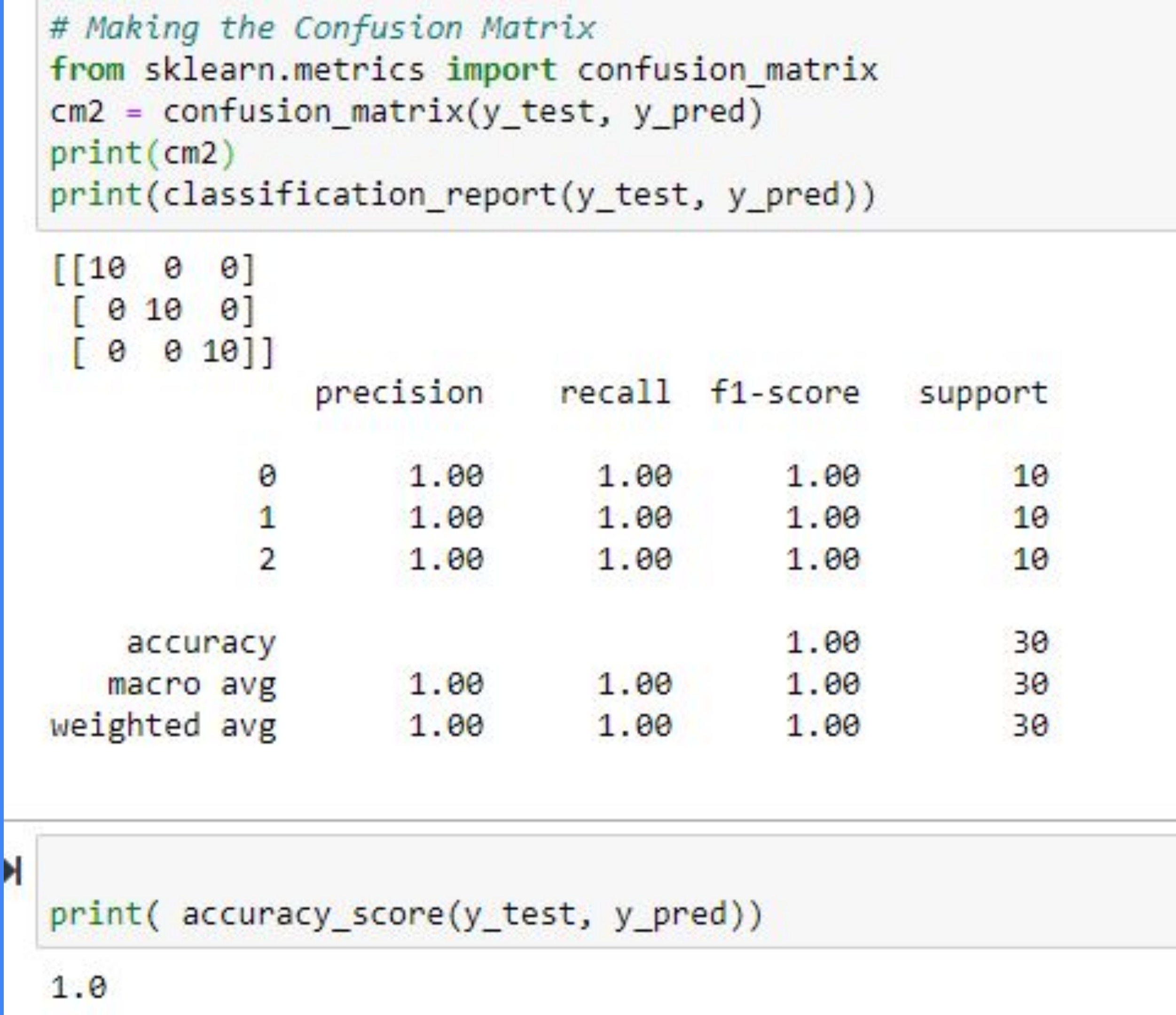
```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        10
           1       1.00      1.00      1.00        10
           2       1.00      0.90      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

Applying Support Vector Classifier Model



```
from sklearn.svm import SVC
classifier = SVC(kernel = 'poly', random_state = 142)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm2 = confusion_matrix(y_test, y_pred)
print(cm2)
print(classification_report(y_test, y_pred))
```

The support vector classifier model gives us 100% accuracy.



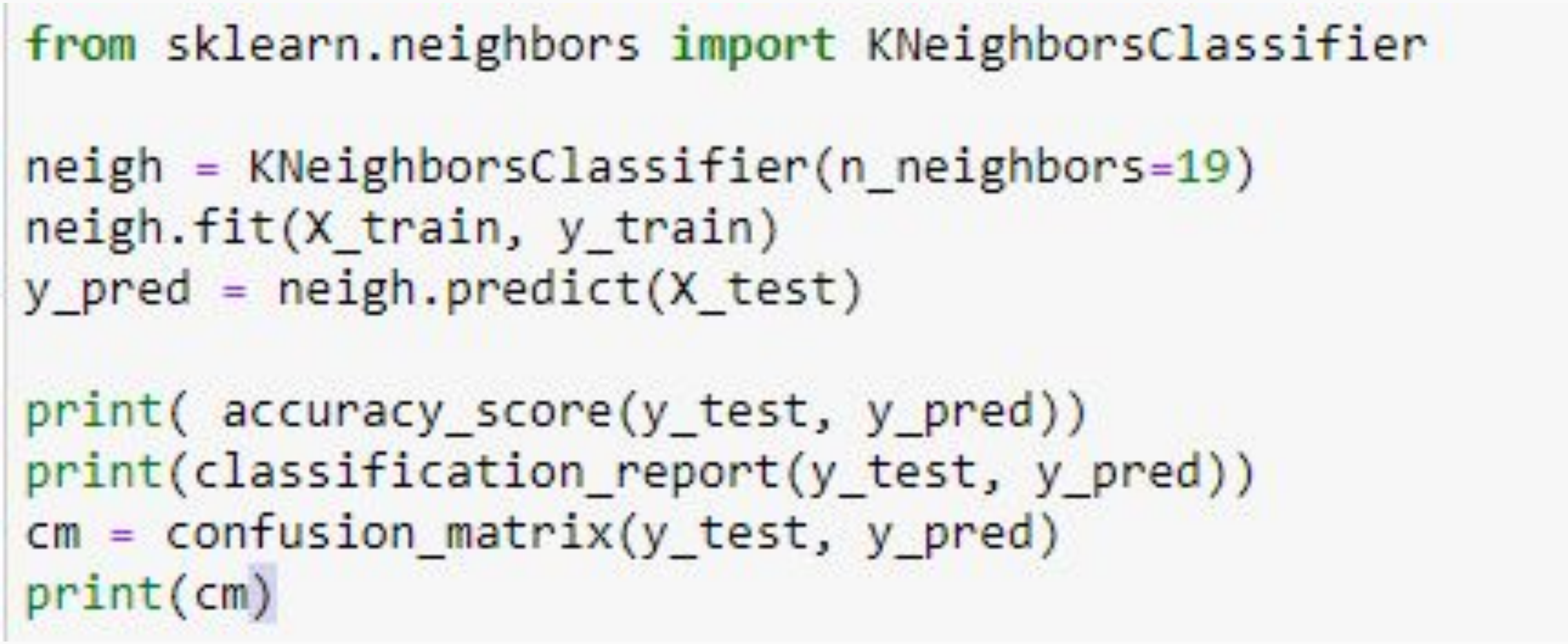
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm2 = confusion_matrix(y_test, y_pred)
print(cm2)
print(classification_report(y_test, y_pred))
```

```
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

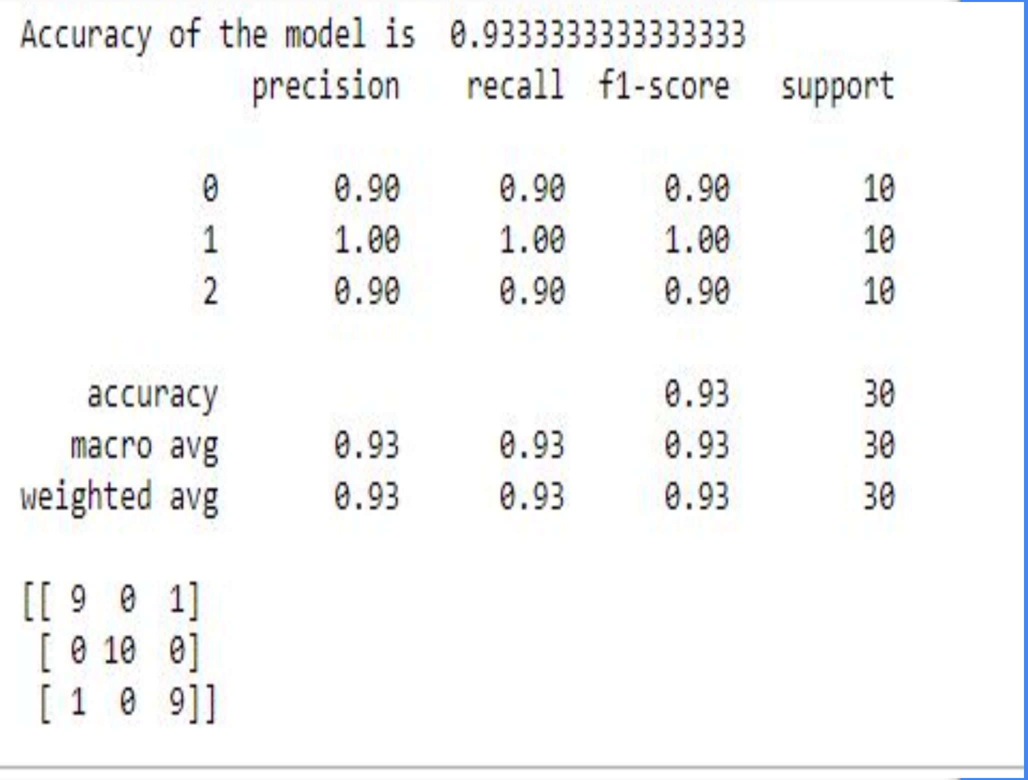
```
print( accuracy_score(y_test, y_pred))
```

```
1.0
```

Applying KNN Classifier



```
from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_neighbors=19)
neigh.fit(X_train, y_train)
y_pred = neigh.predict(X_test)

print( accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
print(cm)
```



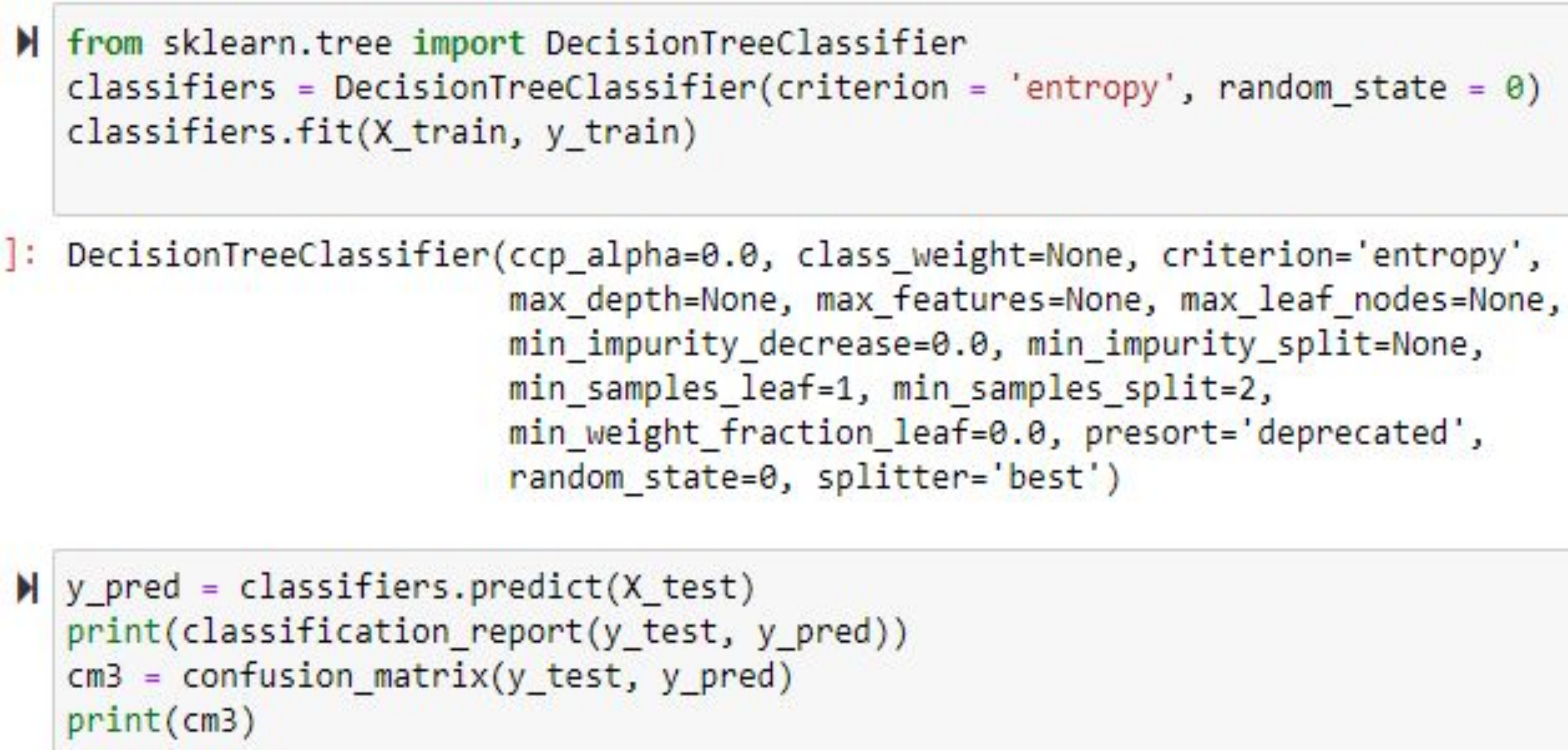
```
Accuracy of the model is  0.9333333333333333
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        10
           1       1.00      1.00      1.00        10
           2       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

[[ 9  0  1]
 [ 0 10  0]
 [ 1  0  9]]
```

Accuracy of model is 93%

The image shows the classification report of the model including:

Precision score,recall score and f1-score.

Applying Decision Tree Classifier

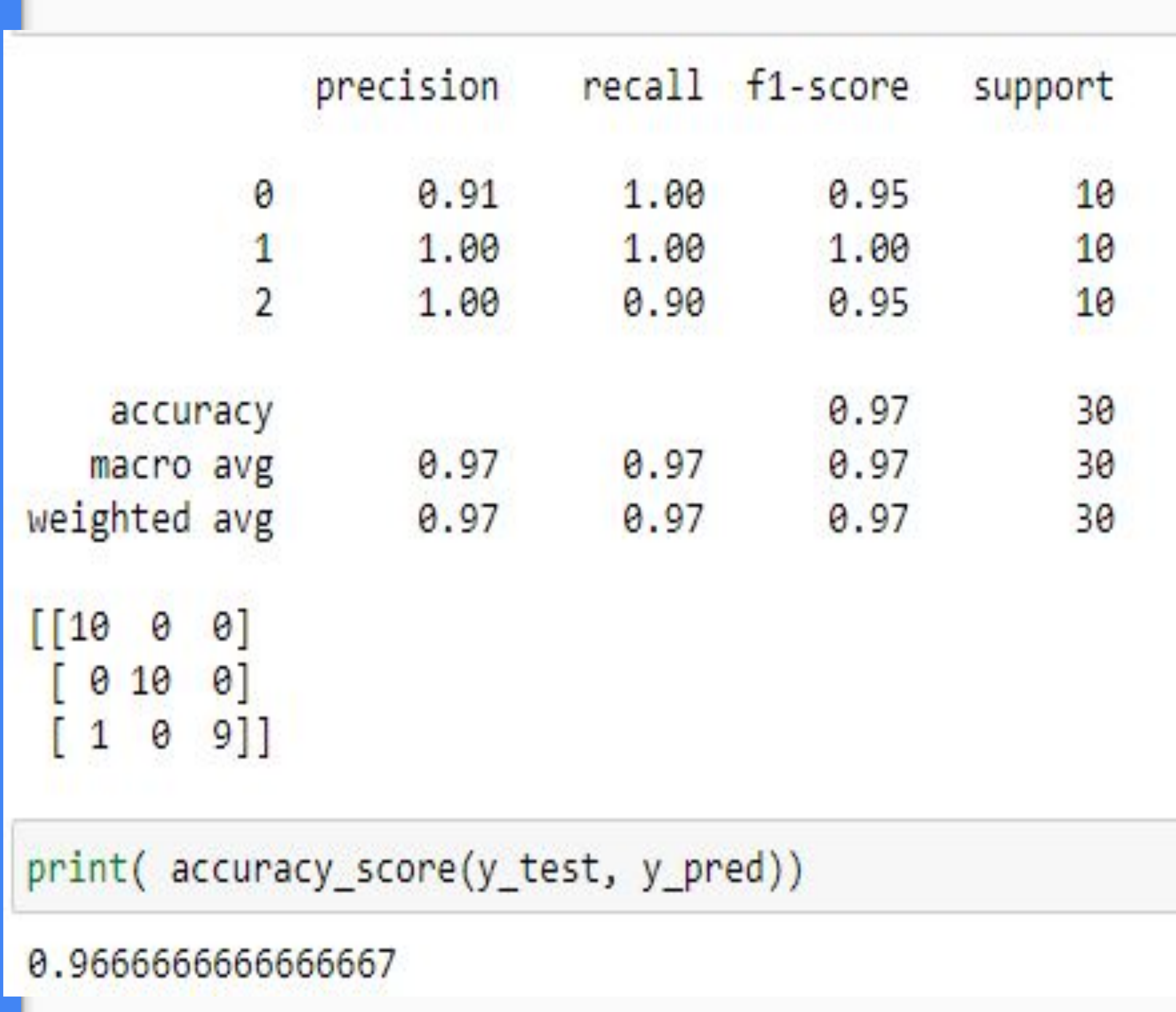


```
from sklearn.tree import DecisionTreeClassifier
classifiers = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifiers.fit(X_train, y_train)
```

```
]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                          max_depth=None, max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, presort='deprecated',
                          random_state=0, splitter='best')
```

```
y_pred = classifiers.predict(X_test)
print(classification_report(y_test, y_pred))
cm3 = confusion_matrix(y_test, y_pred)
print(cm3)
```

This model predicts the test set target values with 97% accuracy.

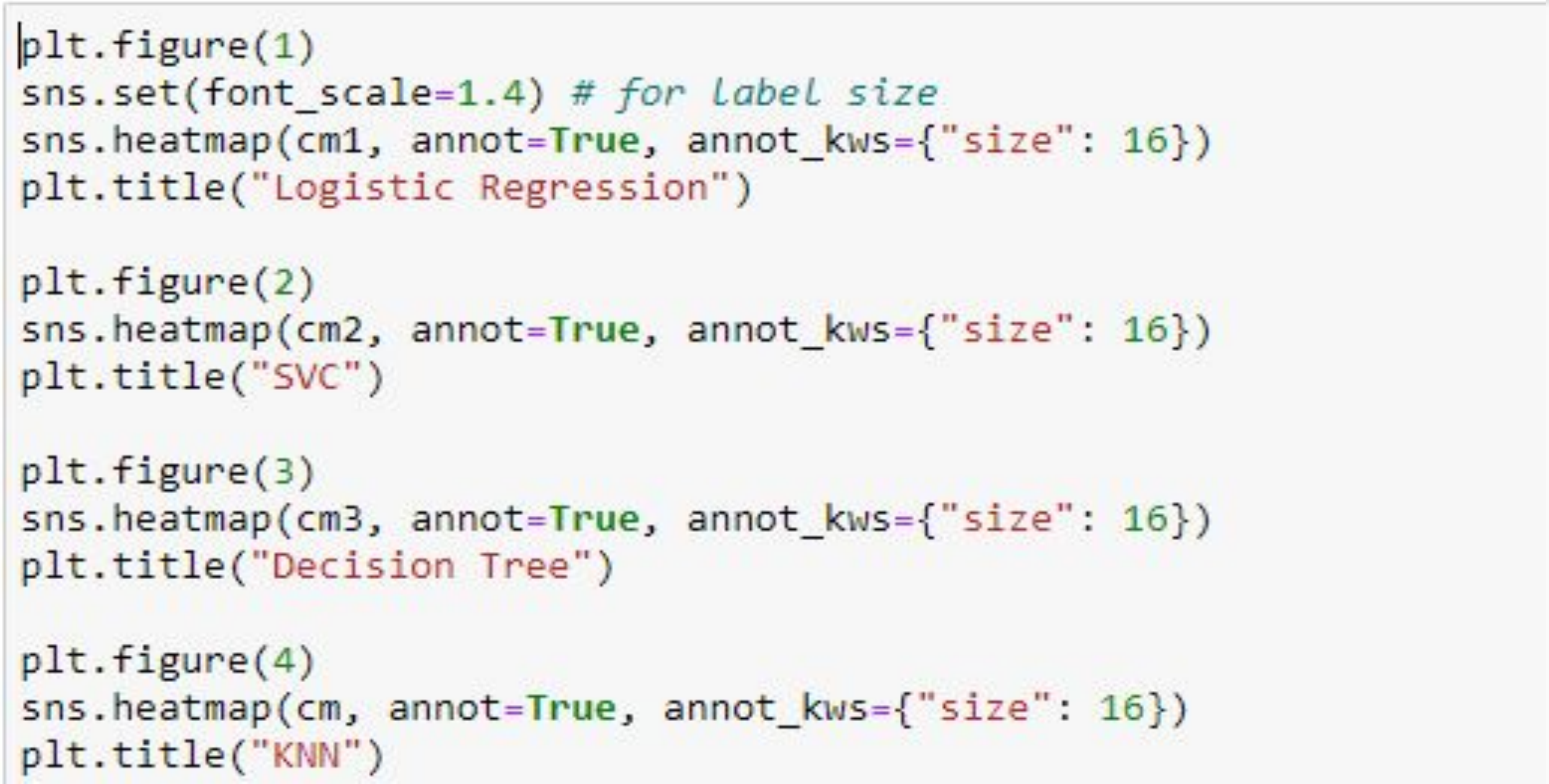


```
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        10
           1       1.00      1.00      1.00        10
           2       1.00      0.90      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

[[10  0  0]
 [ 0 10  0]
 [ 1  0  9]]
```

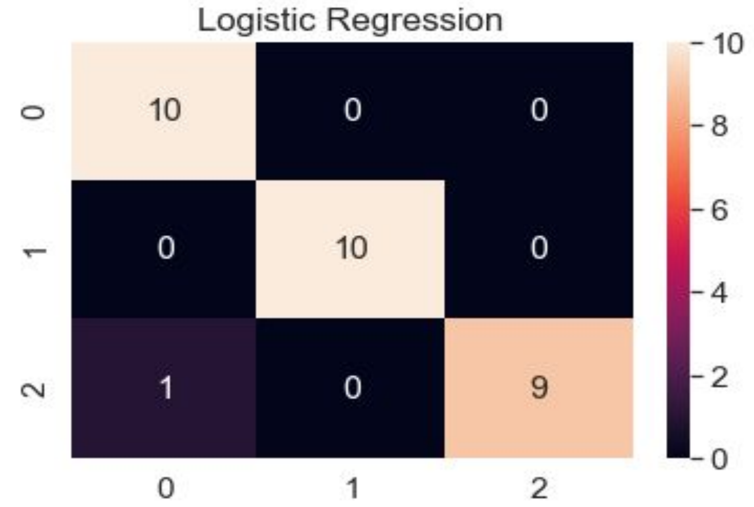
```
print( accuracy_score(y_test, y_pred))
```

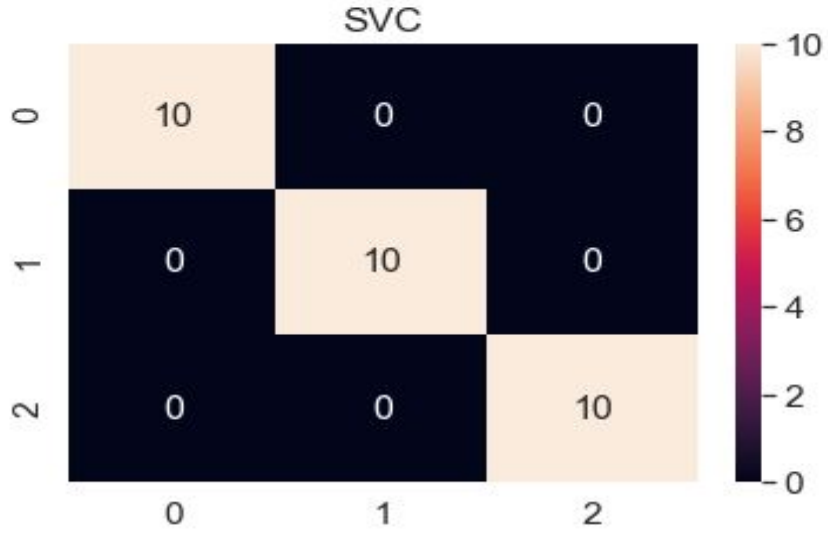
```
0.9666666666666667
```

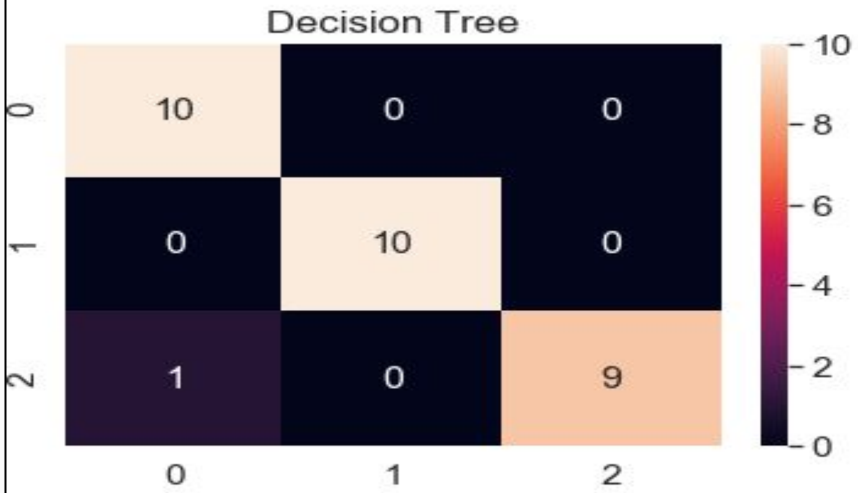
PLOTTING THE CONFUSION MATRICES FOR EACH MODEL

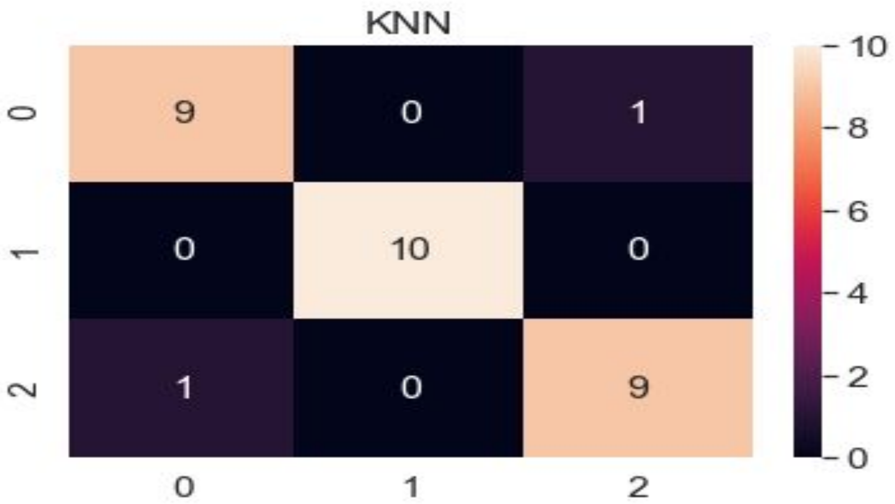


```
plt.figure(1)
sns.set(font_scale=1.4) # for label size
sns.heatmap(cm1, annot=True, annot_kws={"size": 16})
plt.title("Logistic Regression")

plt.figure(2)
sns.heatmap(cm2, annot=True, annot_kws={"size": 16})
plt.title("SVC")

plt.figure(3)
sns.heatmap(cm3, annot=True, annot_kws={"size": 16})
plt.title("Decision Tree")

plt.figure(4)
sns.heatmap(cm, annot=True, annot_kws={"size": 16})
plt.title("KNN")
```

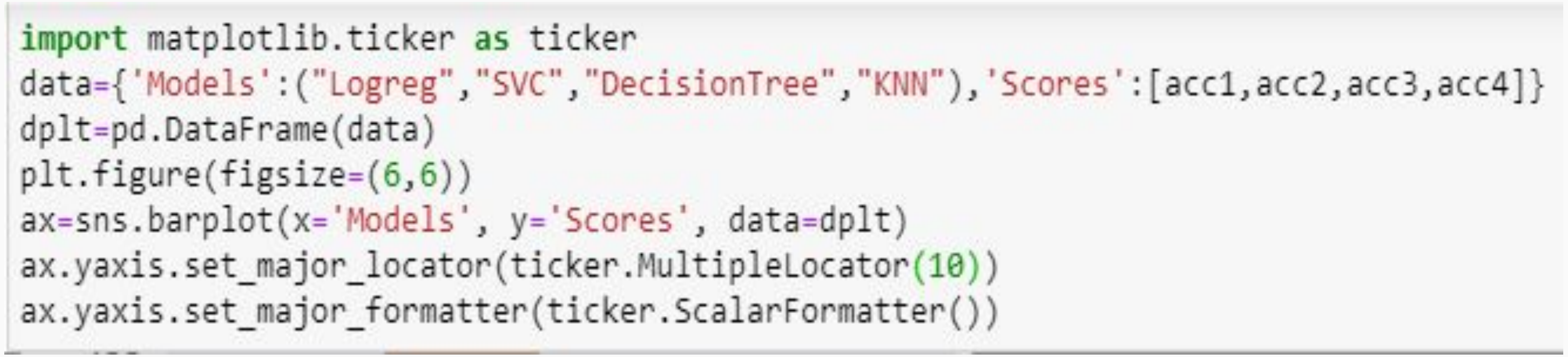

Logistic Regression
10
0
0
0
10
0
1
0
9
0
1
2
0
1
2
0
2
4
6
8
10


SVC
10
0
0
0
10
0
0
0
10
0
1
2
0
1
2
0
2
4
6
8
10


Decision Tree
10
0
0
0
10
0
1
0
9
0
1
2
0
1
2
0
2
4
6
8
10

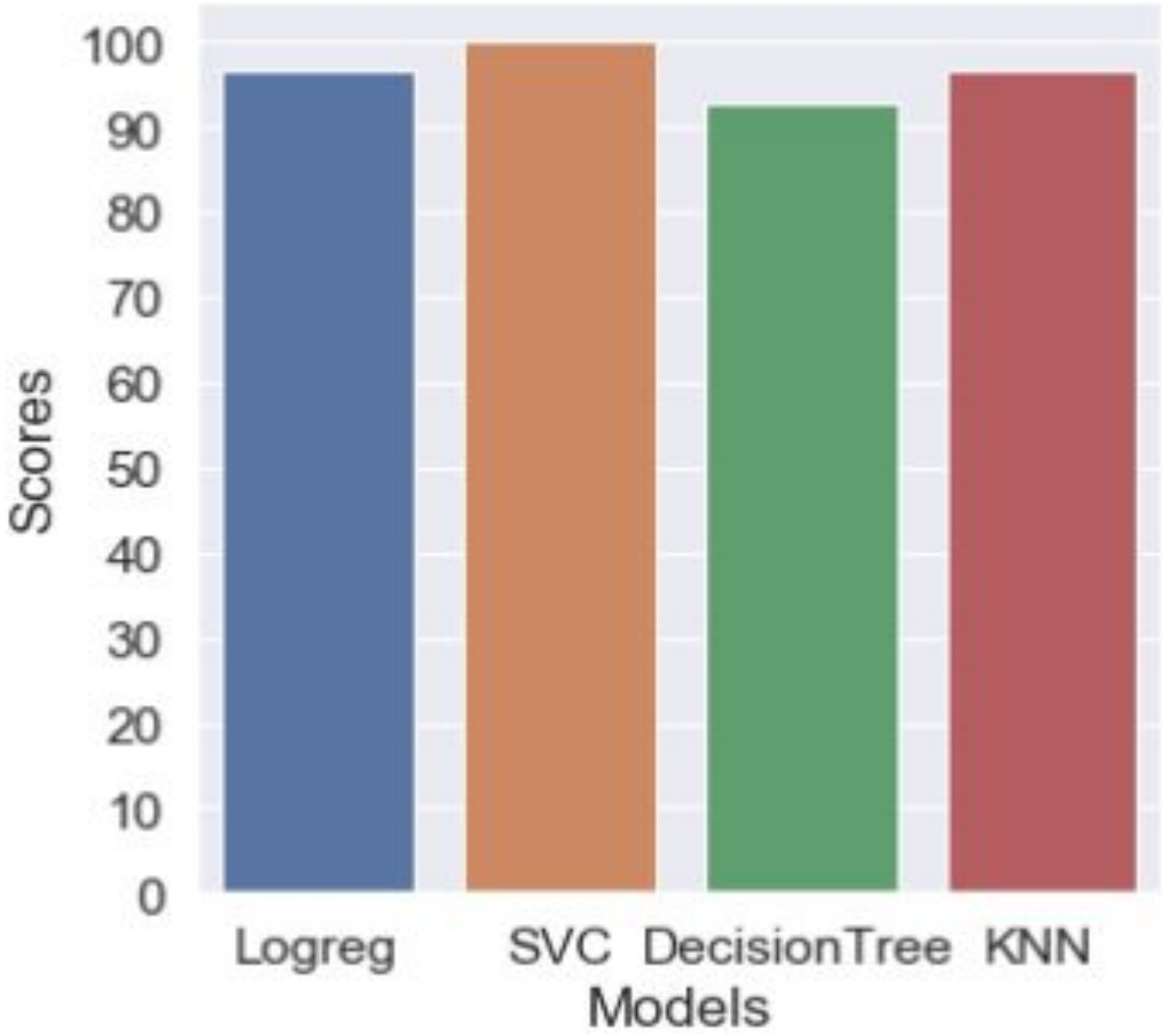

KNN
9
0
1
0
10
0
1
0
9
0
1
2
0
1
2
0
2
4
6
8
10

Comparing accuracy scores

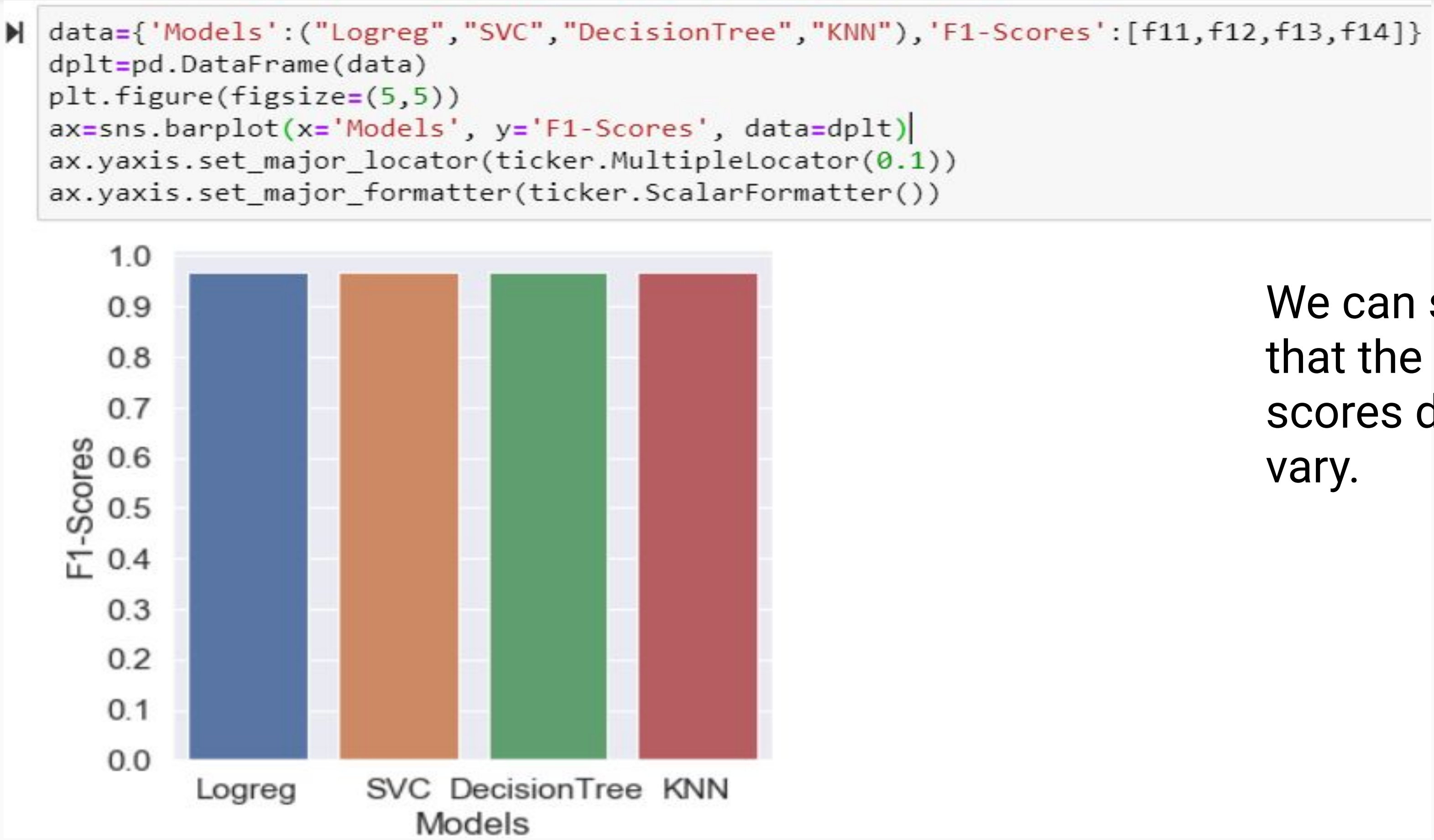


```
import matplotlib.ticker as ticker
data={'Models':("Logreg","SVC","DecisionTree","KNN"),'Scores':[acc1,acc2,acc3,acc4]}
dplt=pd.DataFrame(data)
plt.figure(figsize=(6,6))
ax=sns.barplot(x='Models', y='Scores', data=dplt)
ax.yaxis.set_major_locator(ticker.MultipleLocator(10))
ax.yaxis.set_major_formatter(ticker.ScalarFormatter())
```

From the graph we can see that Support Vector classifier has the highest accuracyscore, Followed by Logistic Regression and KNN models.



F-1 scores

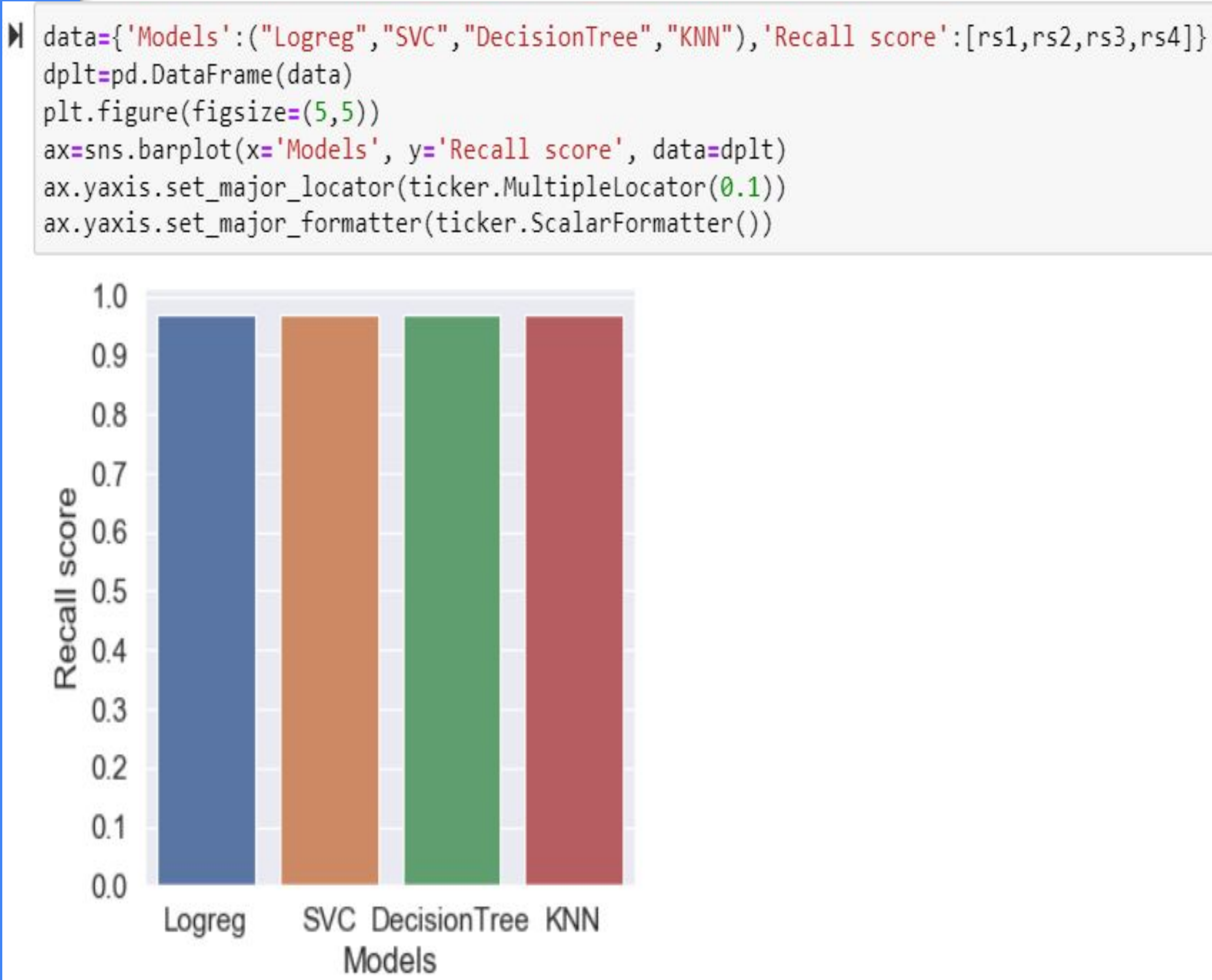


```
data={'Models':("Logreg","SVC","DecisionTree","KNN"),'F1-Scores':[f11,f12,f13,f14]}
dplt=pd.DataFrame(data)
plt.figure(figsize=(5,5))
ax=sns.barplot(x='Models', y='F1-Scores', data=dplt)
ax.yaxis.set_major_locator(ticker.MultipleLocator(0.1))
ax.yaxis.set_major_formatter(ticker.ScalarFormatter())
```

We can see that the f-1 scores do not vary.

Recall Score

The recall score also doesnt vary.



```
data={'Models':("Logreg","SVC","DecisionTree","KNN"),'Recall score':[rs1,rs2,rs3,rs4]}
dplt=pd.DataFrame(data)
plt.figure(figsize=(5,5))
ax=sns.barplot(x='Models', y='Recall score', data=dplt)
ax.yaxis.set_major_locator(ticker.MultipleLocator(0.1))
ax.yaxis.set_major_formatter(ticker.ScalarFormatter())
```