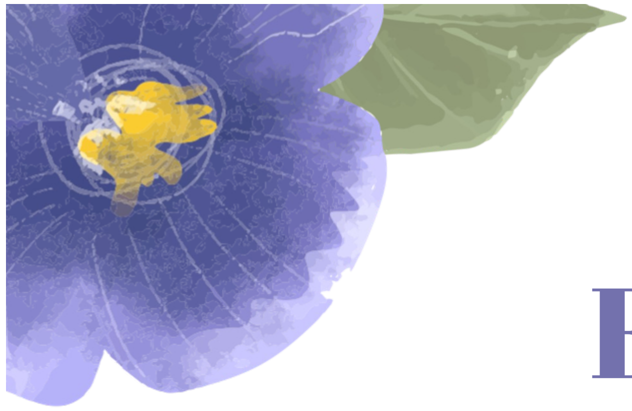
Iris Species

Final Project: Applied Data Science Capstone

Chantal Simó



Executive Summary

Predictive analysis of Iris Species with machine learning

This project utilizes the renowned Iris dataset, introduced by R.A. Fisher in 1936, to address a classic species classification problem.

The primary objective of this project is to explore the unique features of each species and to build a predictive model that can classify new flower samples based on their morphological characteristics. Throughout this analysis, Exploratory Data Analysis (EDA) techniques will be used to identify patterns in the data, interactive visualizations will be implemented to enhance interpretation and a predictive classification model will be developed.

Introduction

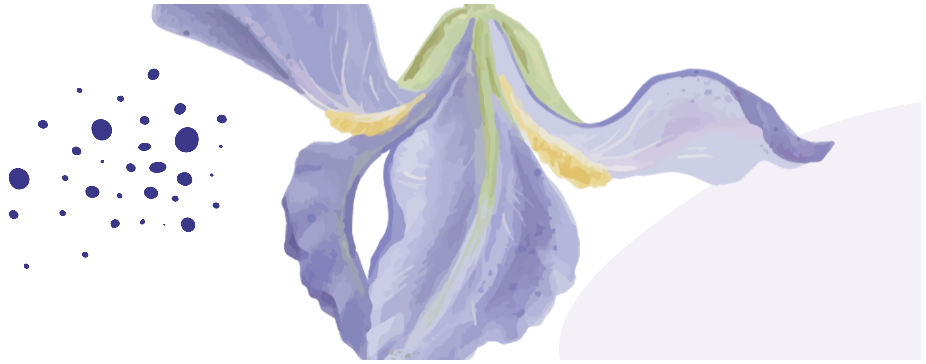
Predictive analysis of Iris Species with machine learning

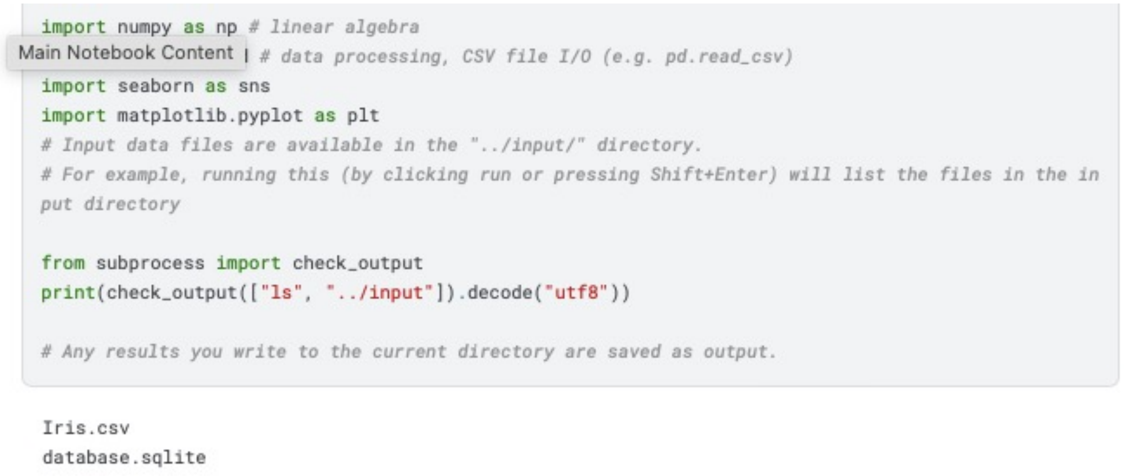
This project focuses on classifying different Iris flower species using the
classic Iris dataset. The goal is to develop a predictive model using machine learning algorithms
to classify flowers based on their physical measurements, such as petal and sepal length
and width.

Project Scope

1. **Exploratory Data Analysis (EDA):** To uncover patterns in the species and their attributes.
2. **Interactive Data Visualization:** To provide clear and dynamic visual interpretations of key features.
3. **Predictive Model:** Use of classification techniques to predict the species of a flower based on its measurements.
4. **Results and Conclusions:** Evaluation of the model's performance, accuracy achieved, and future recommendations.

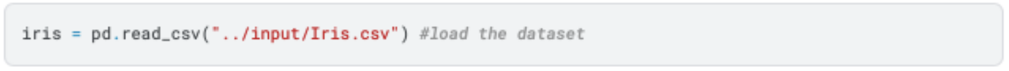
Data collection



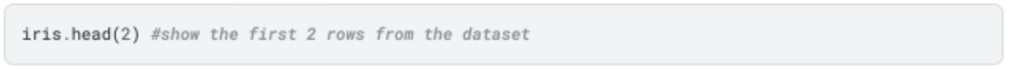


```
import numpy as np # linear algebra
                     # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the in
put directory

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))

# Any results you write to the current directory are saved as output.
```

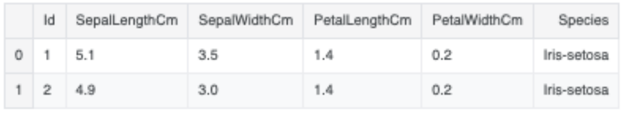
```
Iris.csv
database.sqlite
```



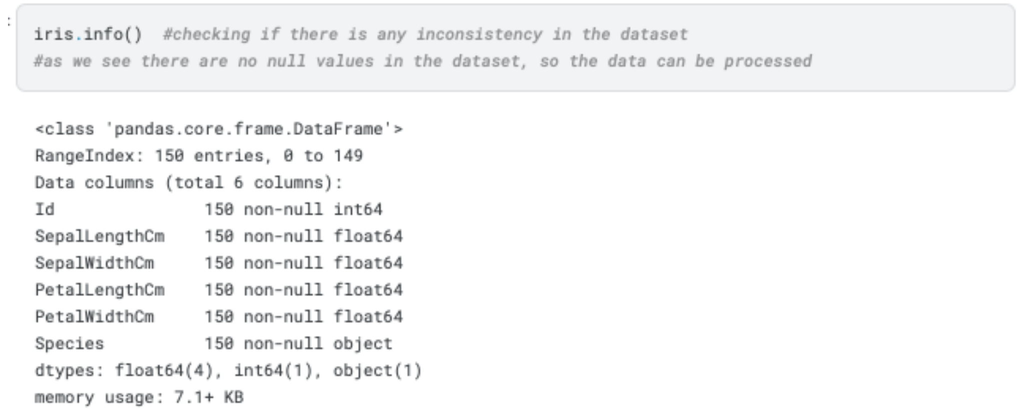
```
iris = pd.read_csv("../input/Iris.csv") #load the dataset
```



```
iris.head(2) #show the first 2 rows from the dataset
```



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	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

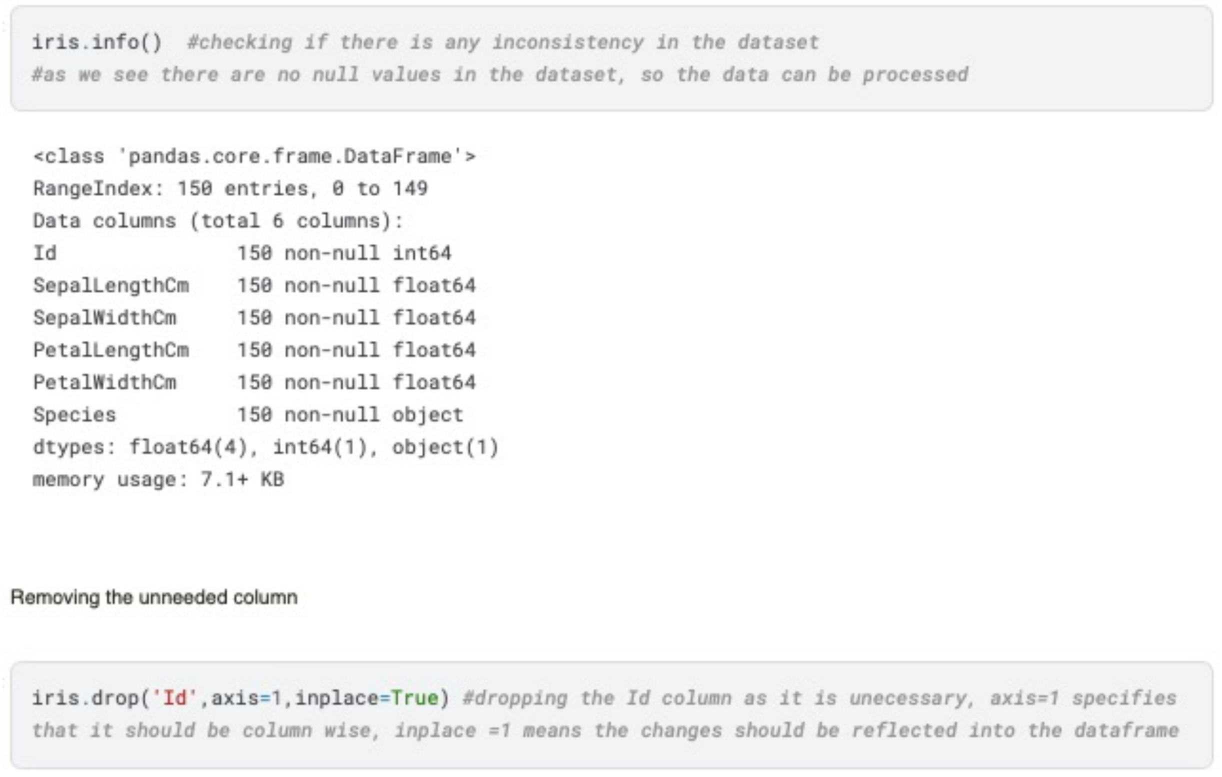


```
iris.info()  #checking if there is any inconsistency in the dataset
#as we see there are no null values in the dataset, so the data can be processed
```

```
<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: 7.1+ KB
```

Here we import the data set and take a quick view of what is inside

Data wrangling



```
iris.info()  #checking if there is any inconsistency in the dataset
#as we see there are no null values in the dataset, so the data can be processed
```

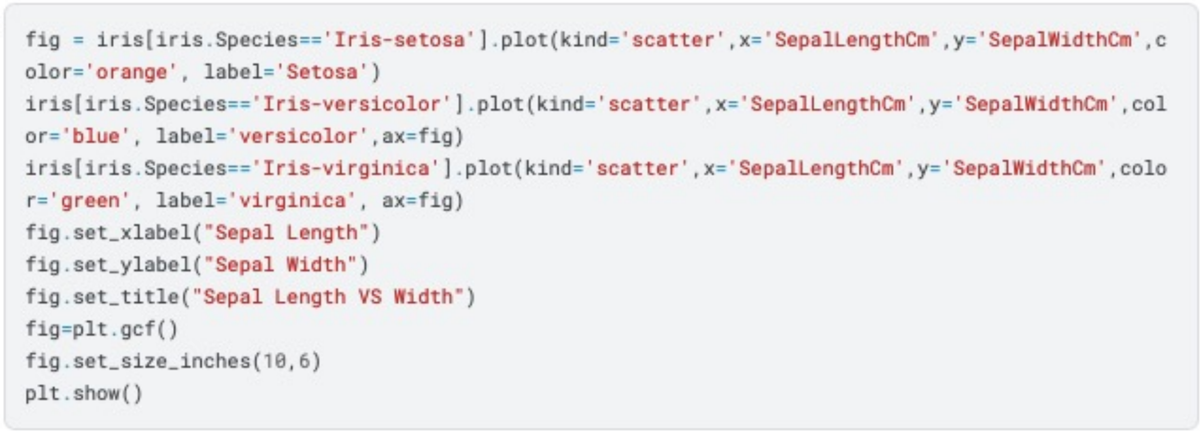
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
Id               150 non-null int64
SepalLengthCm    150 non-null float64
SepalWidthCm     150 non-null float64
PetalLengthCm    150 non-null float64
PetalWidthCm     150 non-null float64
Species          150 non-null object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.1+ KB
```

Removing the unneeded column

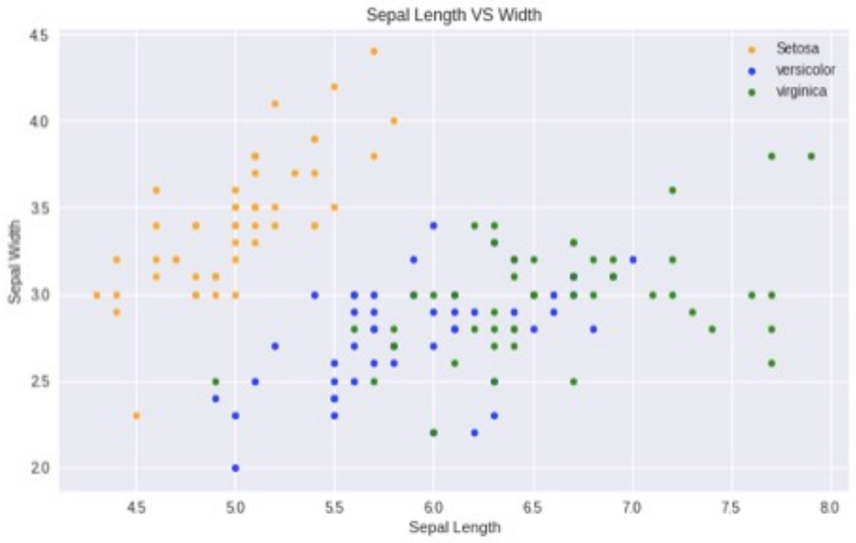
```
iris.drop('Id',axis=1,inplace=True) #dropping the Id column as it is unecessary, axis=1 specifies
that it should be column wise, inplace =1 means the changes should be reflected into the dataframe
```

This database was extracted from Kaggle for the purposes of this project. The Iris dataset is inherently small and is clean enough for visualization. In this case, we will only remove the ID column, as it is not needed for our analysis.

Some Exploratory Data Analysis With Iris



```
fig = iris[iris.Species=='Iris-setosa'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',c
olor='orange', label='Setosa')
iris[iris.Species=='Iris-versicolor'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',col
or='blue', label='versicolor',ax=fig)
iris[iris.Species=='Iris-virginica'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',colo
r='green', label='virginica', ax=fig)
fig.set_xlabel("Sepal Length")
fig.set_ylabel("Sepal Width")
fig.set_title("Sepal Length VS Width")
fig=plt.gcf()
fig.set_size_inches(10,6)
plt.show()
```



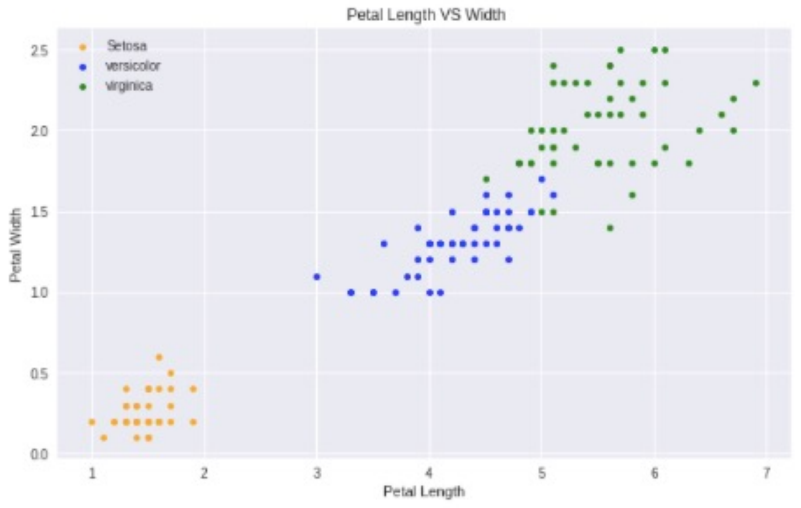
The above graph shows relationship between the sepal length and width

Exploratory Data Analysis

Analysis Methodology

MAP

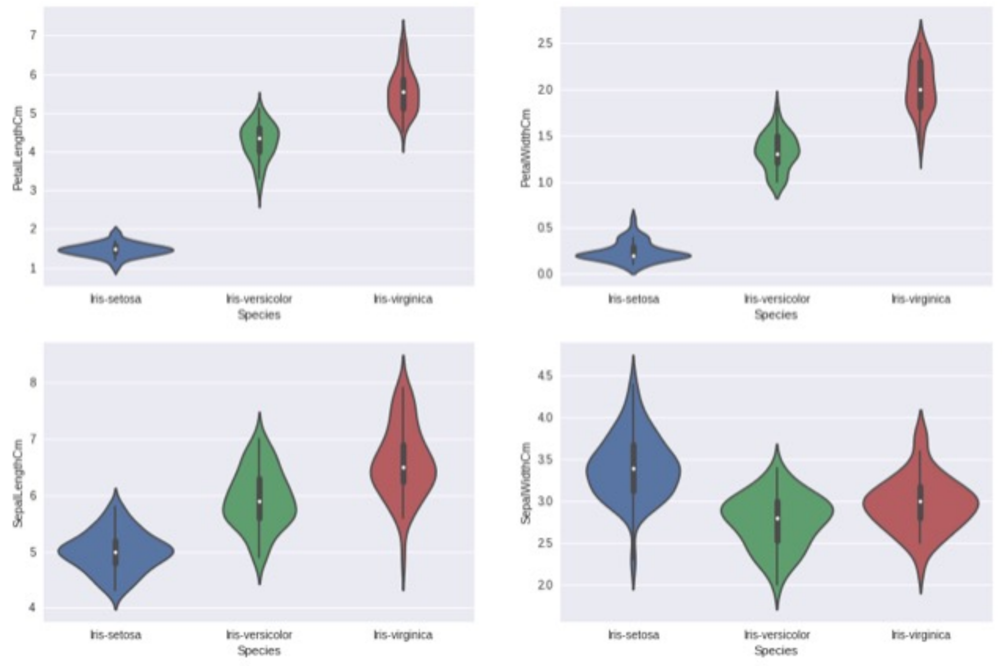
```
fig = iris[iris.Species=='Iris-setosa'].plot.scatter(x='PetalLengthCm',y='PetalWidthCm',color='orange', label='Setosa')
iris[iris.Species=='Iris-versicolor'].plot.scatter(x='PetalLengthCm',y='PetalWidthCm',color='blue', label='versicolor',ax=fig)
iris[iris.Species=='Iris-virginica'].plot.scatter(x='PetalLengthCm',y='PetalWidthCm',color='green', label='virginica', ax=fig)
fig.set_xlabel("Petal Length")
fig.set_ylabel("Petal Width")
fig.set_title(" Petal Length VS Width")
fig=plt.gcf()
fig.set_size_inches(10,6)
plt.show()
```

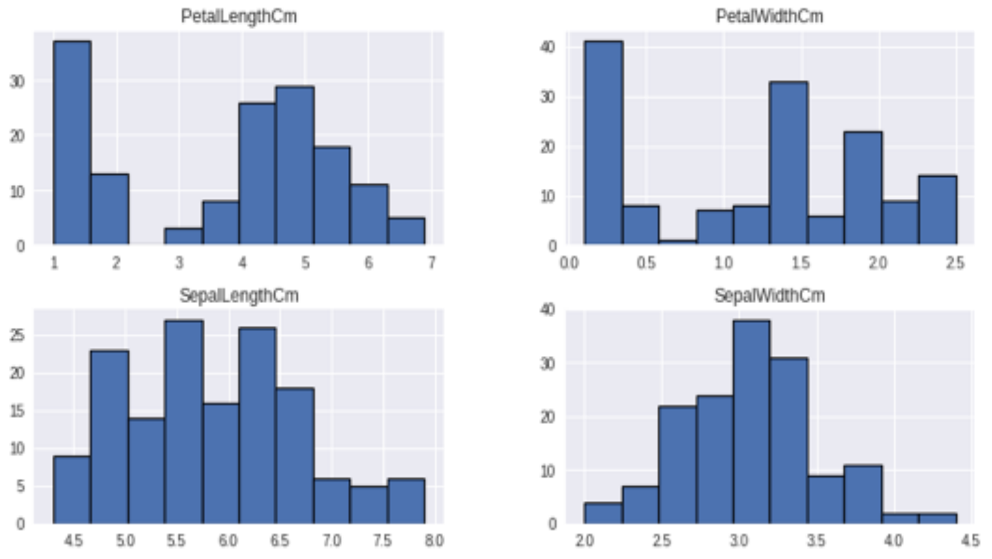


The above graph shows the relationship between the sepal length and width. Now we will check the relationship between the petal length and width.

As we can see the Petal Features are giving a better cluster division compared to the Sepal features. This is an indication that the Petals can help in better and accurate Predictions over the Sepal.

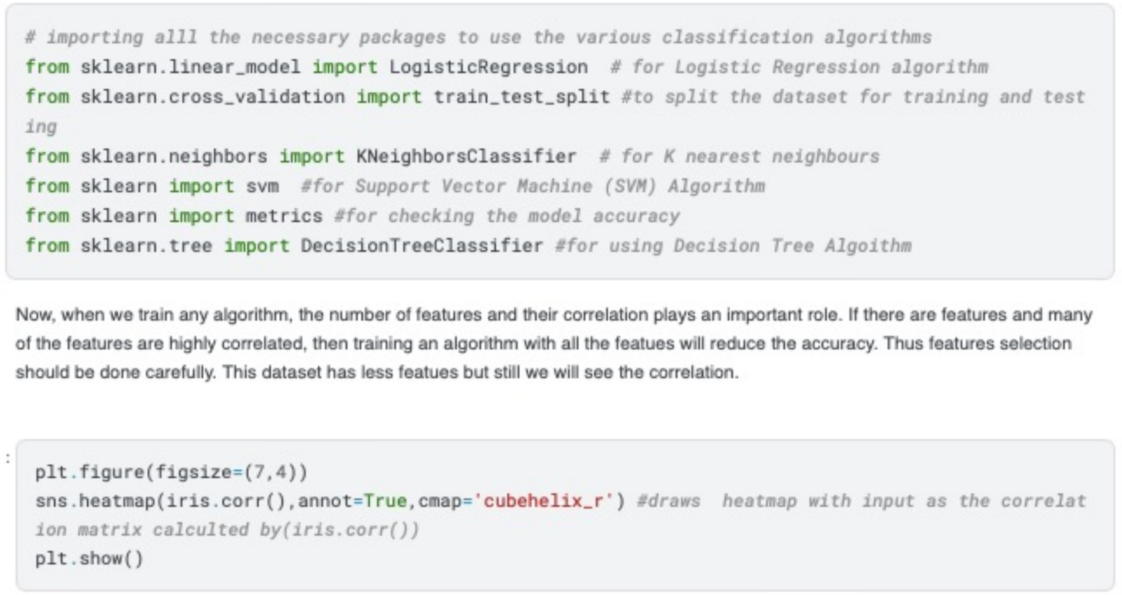
Visualizations





The violinplot shows density of the length and width in the species. The thinner part denotes that there is less density whereas the fatter part conveys higher density

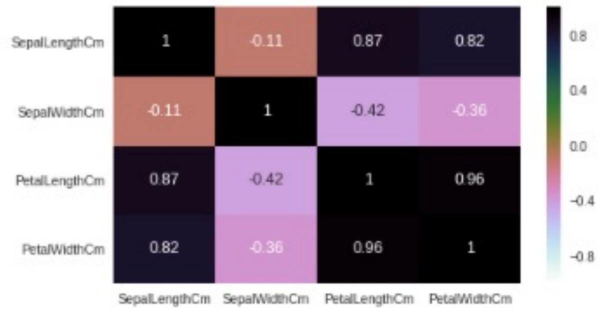
Predictive Analysis



```
# importing alll the necessary packages to use the various classification algorithms
from sklearn.linear_model import LogisticRegression  # for Logistic Regression algorithm
from sklearn.cross_validation import train_test_split #to split the dataset for training and test
ing
from sklearn.neighbors import KNeighborsClassifier  # for K nearest neighbours
from sklearn import svm  #for Support Vector Machine (SVM) Algorithm
from sklearn import metrics #for checking the model accuracy
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
```

Now, when we train any algorithm, the number of features and their correlation plays an important role. If there are features and many of the features are highly correlated, then training an algorithm with all the featues will reduce the accuracy. Thus features selection should be done carefully. This dataset has less featues but still we will see the correlation.

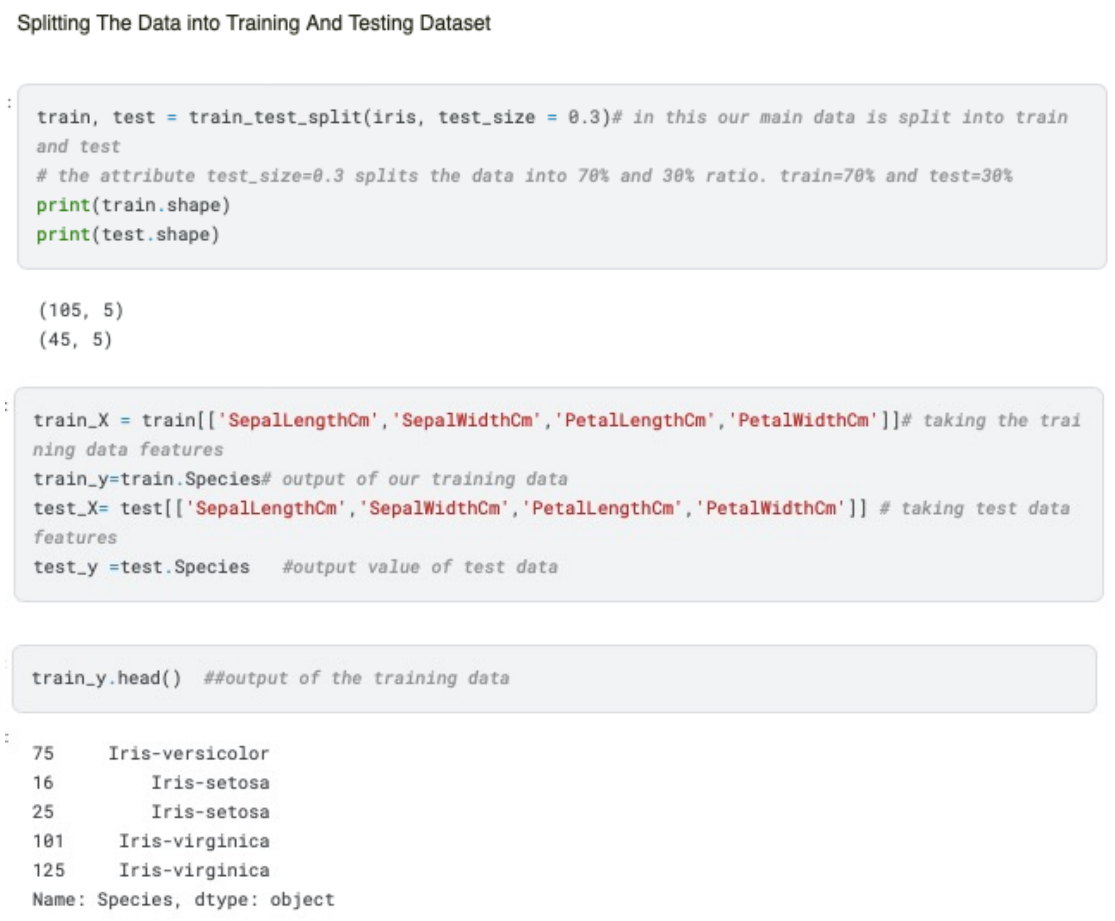
```
plt.figure(figsize=(7,4))
sns.heatmap(iris.corr(),annot=True,cmap='cubehelix_r') #draws  heatmap with input as the correlat
ion matrix calculted by(iris.corr())
plt.show()
```



	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1	-0.11	0.87	0.82
SepalWidthCm	-0.11	1	-0.42	-0.36
PetalLengthCm	0.87	-0.42	1	0.96
PetalWidthCm	0.82	-0.36	0.96	1

- The Sepal Width and Length are not correlated
- The Petal Width and Length are highly correlated
- We will use all the features for training the algorithm and check the accuracy.
- Then we will use 1 Petal Feature and 1 Sepal Feature to check the accuracy of the algorithm as we are using only 2 features that are not correlated

Predictive Analysis



Splitting The Data into Training And Testing Dataset

```
train, test = train_test_split(iris, test_size = 0.3)# in this our main data is split into train
and test
# the attribute test_size=0.3 splits the data into 70% and 30% ratio. train=70% and test=30%
print(train.shape)
print(test.shape)
```

```
(105, 5)
(45, 5)
```

```
train_X = train[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]# taking the trai
ning data features
train_y=train.Species# output of our training data
test_X= test[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']] # taking test data
features
test_y =test.Species   #output value of test data
```

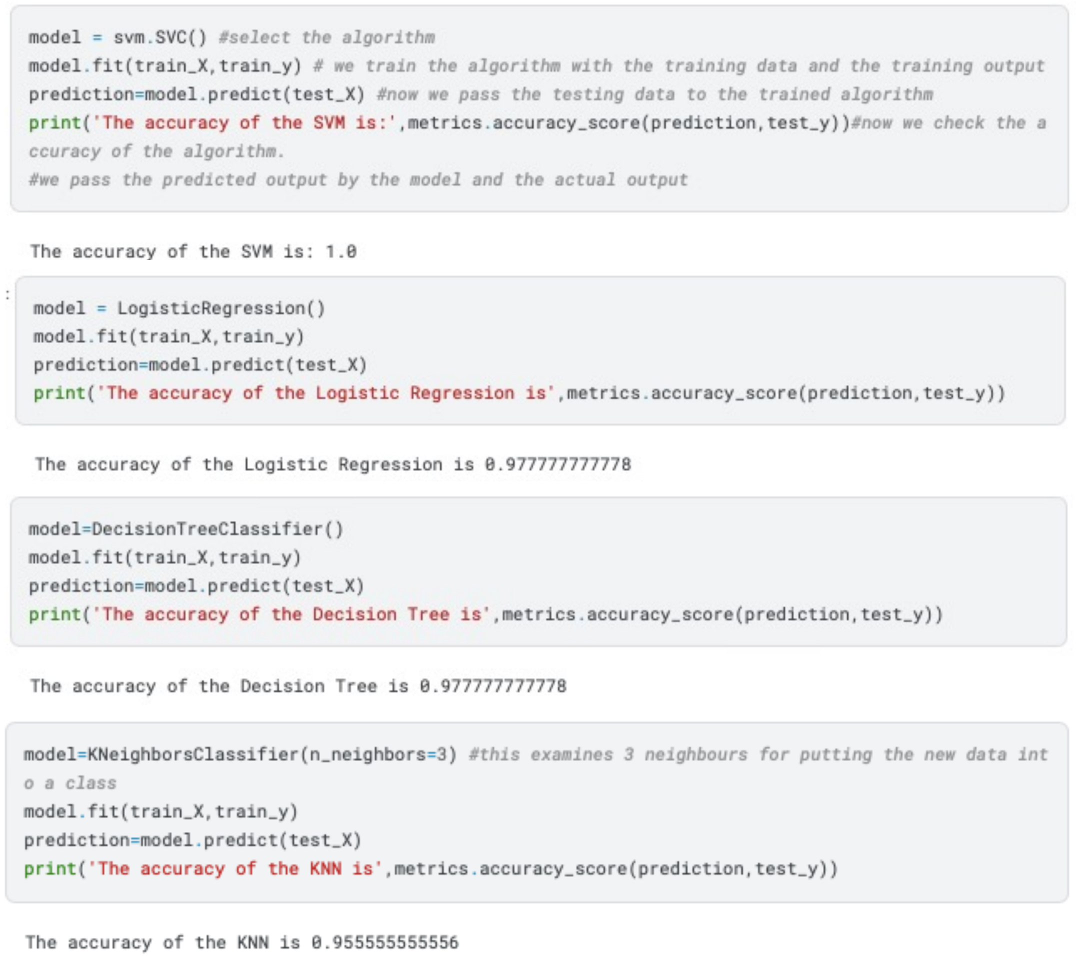
```
train_y.head()  ##output of the training data
```

```
75     Iris-versicolor
16         Iris-setosa
25         Iris-setosa
101     Iris-virginica
125     Iris-virginica
Name: Species, dtype: object
```

Steps To Be Followed When Applying an Algorithm

1. Split the dataset into training and testing
2. Select any algorithm based on the problem (classification)
3. Then pass the training dataset to the algorithm to train it with **.fit()** method
4. Then pass the testing data to the trained algorithm to predict the outcome **with predict()** method.
5. We then check the accuracy by **passing the predicted outcome and the actual output** to the model.

Machine Learning



```
model = svm.SVC() #select the algorithm
model.fit(train_X,train_y) # we train the algorithm with the training data and the training output
prediction=model.predict(test_X) #now we pass the testing data to the trained algorithm
print('The accuracy of the SVM is:',metrics.accuracy_score(prediction,test_y))#now we check the a
ccuracy of the algorithm.
#we pass the predicted output by the model and the actual output
```

```
The accuracy of the SVM is: 1.0
```

```
model = LogisticRegression()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
print('The accuracy of the Logistic Regression is',metrics.accuracy_score(prediction,test_y))
```

```
The accuracy of the Logistic Regression is 0.977777777778
```

```
model=DecisionTreeClassifier()
model.fit(train_X,train_y)
prediction=model.predict(test_X)
print('The accuracy of the Decision Tree is',metrics.accuracy_score(prediction,test_y))
```

```
The accuracy of the Decision Tree is 0.977777777778
```

```
model=KNeighborsClassifier(n_neighbors=3) #this examines 3 neighbours for putting the new data int
o a class
model.fit(train_X,train_y)
prediction=model.predict(test_X)
print('The accuracy of the KNN is',metrics.accuracy_score(prediction,test_y))
```

```
The accuracy of the KNN is 0.955555555556
```

Training set

SVM is giving very good accuracy with the training data. We will continue to check the accuracy of different models.

Machine Learning

Creating Petals And Sepals Training Data

```
petal=iris[['PetalLengthCm','PetalWidthCm','Species']]
sepal=iris[['SepalLengthCm','SepalWidthCm','Species']]
```

```
train_p,test_p=train_test_split(petal,test_size=0.3,random_state=0)  #petals
train_x_p=train_p[['PetalWidthCm','PetalLengthCm']]
train_y_p=train_p.Species
test_x_p=test_p[['PetalWidthCm','PetalLengthCm']]
test_y_p=test_p.Species


train_s,test_s=train_test_split(sepal,test_size=0.3,random_state=0)  #Sepal
train_x_s=train_s[['SepalWidthCm','SepalLengthCm']]
train_y_s=train_s.Species
test_x_s=test_s[['SepalWidthCm','SepalLengthCm']]
test_y_s=test_s.Species
```

K-Nearest Neighbours

```
model=KNeighborsClassifier(n_neighbors=3)
model.fit(train_x_p,train_y_p)
prediction=model.predict(test_x_p)
print('The accuracy of the KNN using Petals is:',metrics.accuracy_score(prediction,test_y_p))

model.fit(train_x_s,train_y_s)
prediction=model.predict(test_x_s)
print('The accuracy of the KNN using Sepals is:',metrics.accuracy_score(prediction,test_y_s))
```

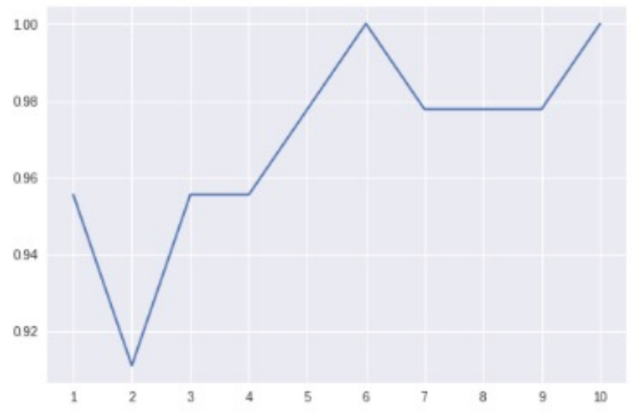
```
The accuracy of the KNN using Petals is: 0.9777777777778
The accuracy of the KNN using Sepals is: 0.733333333333
```

Training and testing set

Even the **SVM** was giving the best accuracy with the training data. Now we will use Petals and Sepals separately. When trying all the models we used in the training data, the results show that the **KNN** was giving now the best accuracy given the factors applied.

Predictive Analysis

```
a_index=list(range(1,11))
a=pd.Series()
x=[1,2,3,4,5,6,7,8,9,10]
for i in list(range(1,11)):
    model=KNeighborsClassifier(n_neighbors=i)
    model.fit(train_X,train_y)
    prediction=model.predict(test_X)
    a=a.append(pd.Series(metrics.accuracy_score(prediction,test_y)))
plt.plot(a_index, a)
plt.xticks(x)
```



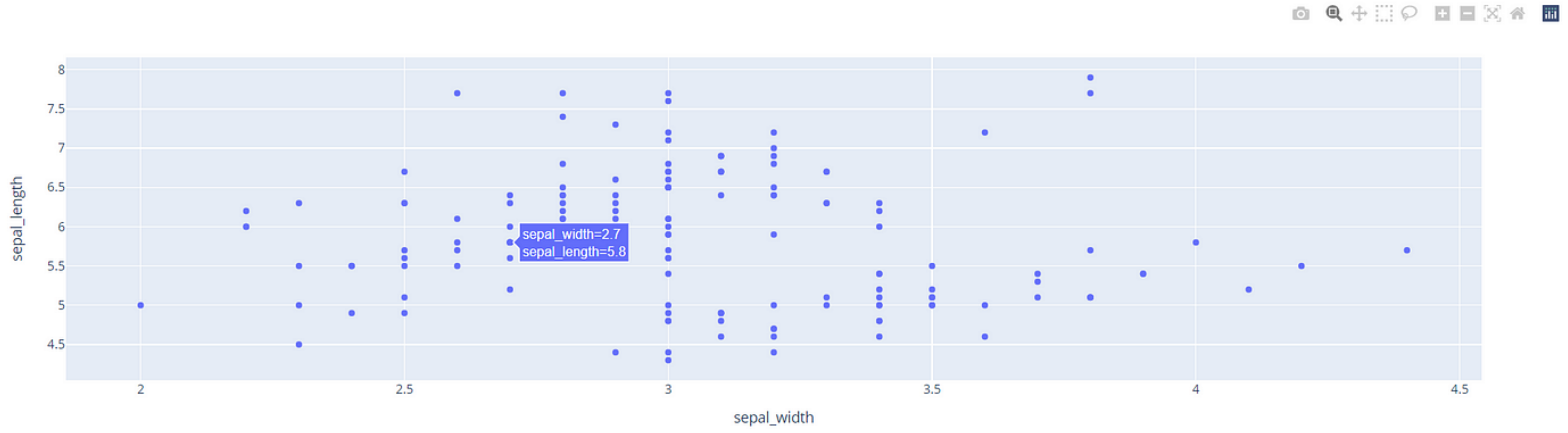
Results for accuracy for various values of n for K-Nearest nerighbours

Above is the graph showing the accuracy for the KNN models using different values of n.

Plotly Dashboard

Dashboard of Iris dataset

Dash: A web application framework for Python.



Conclusion

Predictive analysis of Iris Species with machine learning main inisghts

The analysis demonstrates that using Petal dimensions (Width and Length) for training the model results in significantly better accuracy compared to using Sepal dimensions. This outcome aligns with our expectations, as indicated by the heatmap, which revealed a low correlation between Sepal Width and Length, while Petal Width and Length exhibited a strong correlation. Therefore, Petal measurements are more informative and effective for improving model performance in this dataset.