Assignment – 1

AML 1413 Group - 1 Introduction to Artificial Intelligence

Group Members:

Sai Varun Kollipara – C0828403

Bhanu Prakash Mahadevuni C0850515

Deeksha Naikap C0835440

Pramod Reddy Gurrala C0850493

Problem Statement:

- Demonstrate Supervised and unsupervised ML models for classification and clustering using UCI dataset.

Solution and Results:

- Data Considered from UCI Data Repository – IRIS Data. Click here for data. This data consists of information a specific kind of flower which is known as IRIS, and there are multiple categories in it. The information consists of sepal length, sepal width, petal length, petal width and the kind of category.

- Supervised Learning – It is a machine learning process where labelled data is used to train along with the output, and the trained model is expected to predict information based on the trained data.
    - For this assignment, we are using the algorithms under binary classification like Decision Trees, Support Vector Machines, Naïve Bayes, and Logistic Regression.
    - First, we are implementing the data and explore the data to find any correlation between the features.

```
In [8]: #information about the data/

        df.info()
```

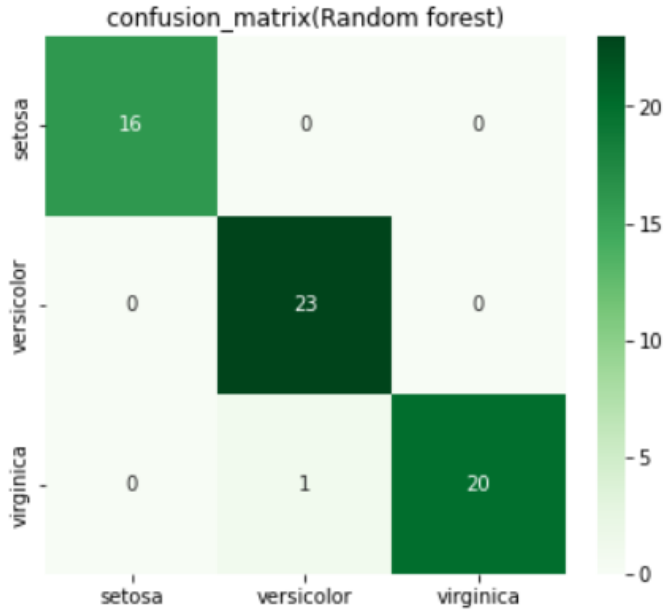
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150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

- Next, we are splitting the data into testing and training data, which will be used for training the models and provide the accuracy for the comparison.
- Now all the results are displayed in here, from the images we can understand the accuracy of the models, working and performance of each one. From the results we can see decision tree is giving best accuracy.

Out[52]:

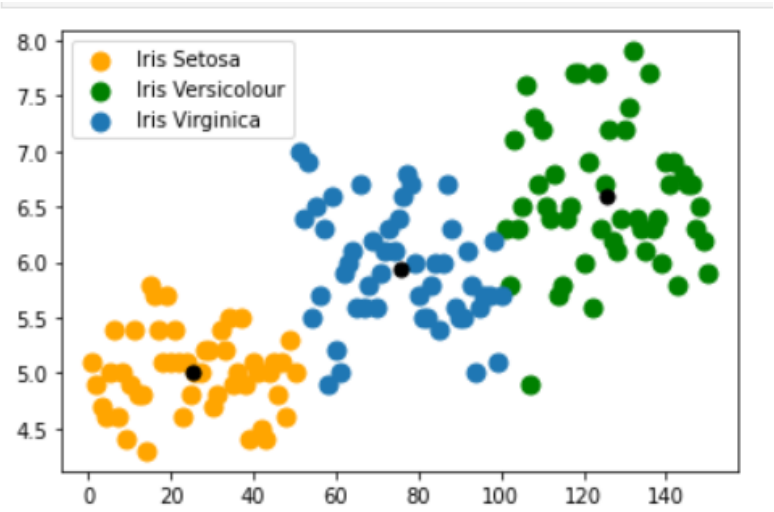
	Model	Score
3	Decision Tree	100
0	Logistic Regression	98
1	Support Vector Machines	98
2	Naive Bayes	98
4	Random Forest	98

- Here we are plotting a confusion matrix to presenting the results. This can be used to identify the intensity of relation between the features.



- Unsupervised Learning – It is also a machine leaning process where the model is expected to identify various features in data and categorize it accordingly.
  - For the assignment, we are using K-Means Clustering which is implemented for providing insights of the data.
  - K-Means clustering algorithm will provide the results in the form of clusters and the number is defined by ‘k’. And the clusters will be labelled based on the observations and the value limits provided in the function variables.

- Since the model are not required for any pre-training, we customized it to provide the results with 3 clusters, since the categories are 3 types of IRIS flower. And using the visualization tools to represent the response of the algorithm.



Source Code:

In [4]:
```
# Importing pakages for data processing and visualization

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

In [5]:
```
df = pd.read_csv('./Iris.csv')
```

In [77]:
```
df.head(10)
```

Out[77]:

	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
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
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

In [8]:
```
#information about the data/

df.info()
```

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

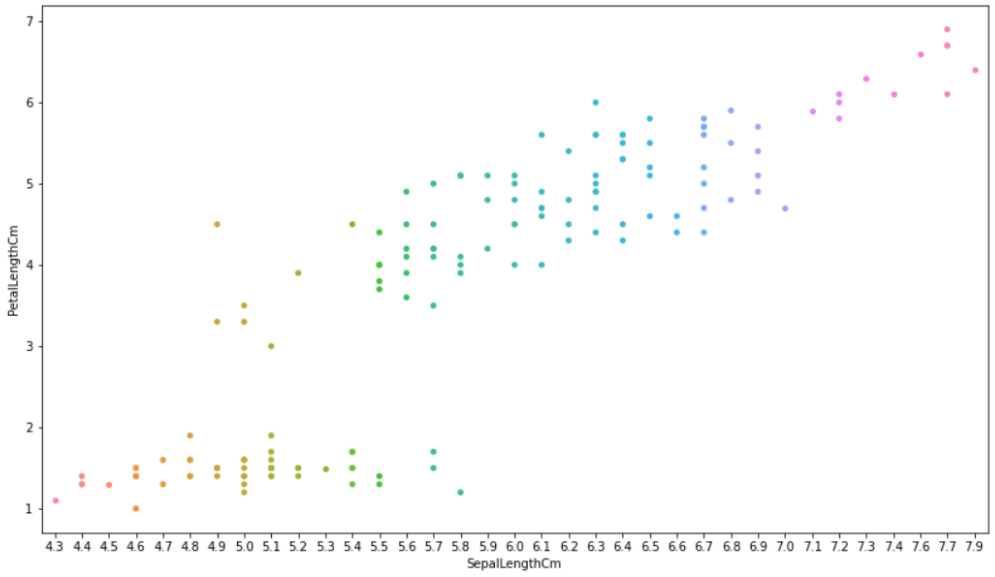
In [11]:
```
#checking for null values

df.isnull().sum()
```

Out[11]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

Oberving Length of Sepal and Petal

[13]:
```
plt.figure(figsize=(14,8))
plt.xlabel('Sepal Length in cm')
plt.ylabel('Petal Length in cm')
sns.stripplot(x = 'SepalLengthCm', y = 'PetalLengthCm', data = df,jitter = False)
```

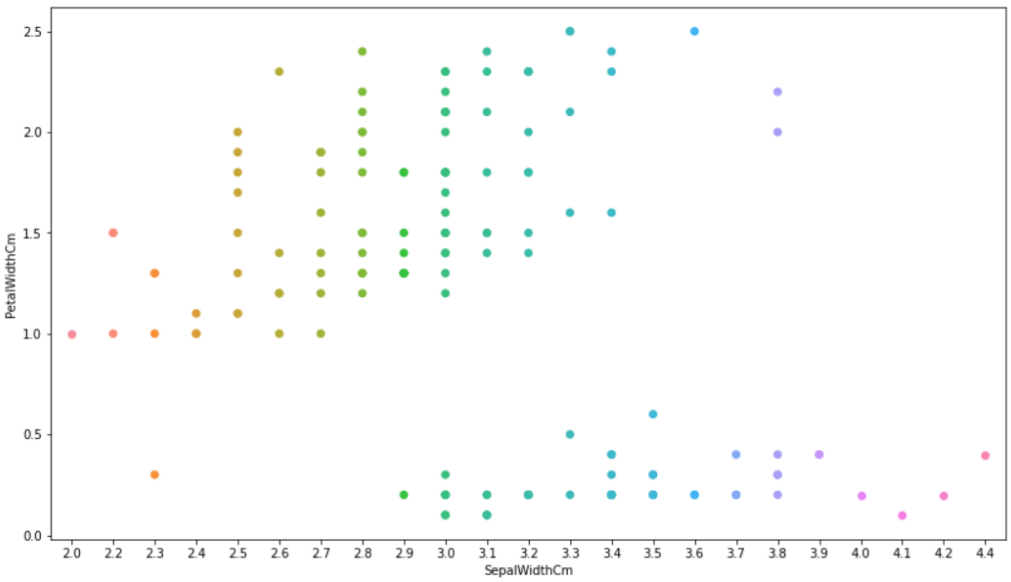
[13]:
```
<AxesSubplot:xlabel='SepalLengthCm', ylabel='PetalLengthCm'>
```



Observing Width of Sepal and Petal

[20]:
```
plt.figure(figsize=(14,8))
plt.xlabel('Sepal Width in Cm')
plt.ylabel('Petal Width in Cm')
sns.stripplot(x = 'SepalWidthCm', y = 'PetalWidthCm', data = df,size = 7,jitter = False)
```

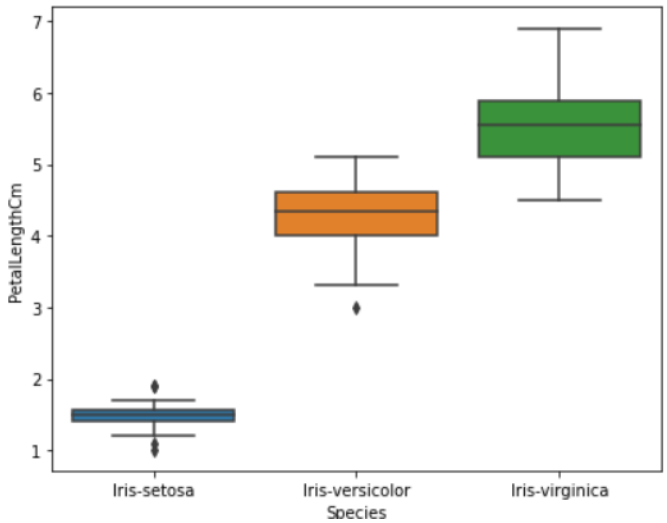
:[20]: <AxesSubplot:xlabel='SepalWidthCm', ylabel='PetalWidthCm'>

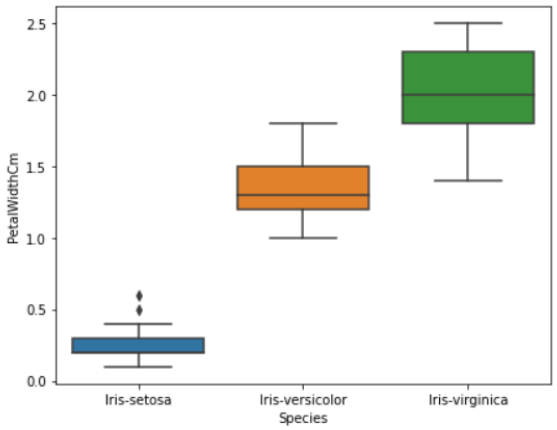


Observing Petal Length and Width

In [17]:
```
plt.figure(figsize=(15,12))
plt.subplots_adjust(hspace = .25)

plt.subplot(2,2,1)
sns.boxplot(x="Species", y="PetalLengthCm", data=df)

plt.subplot(2,2,3)
sns.boxplot(x="Species", y="PetalWidthCm", data=df)
```

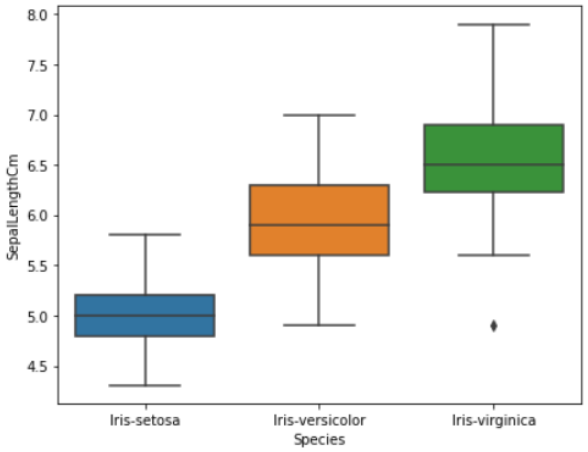
Out[17]: <AxesSubplot:xlabel='Species', ylabel='PetalWidthCm'>

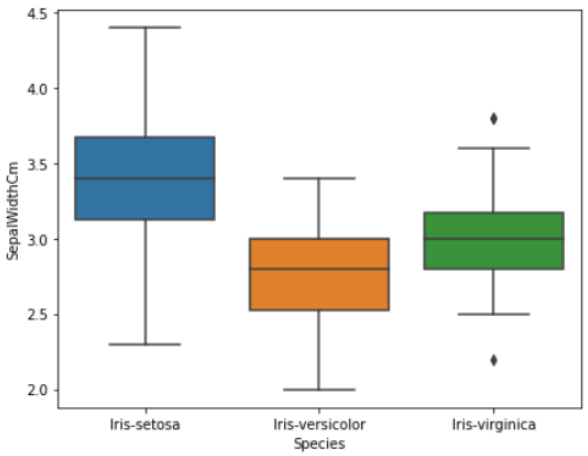




Observing Sepal length and Width

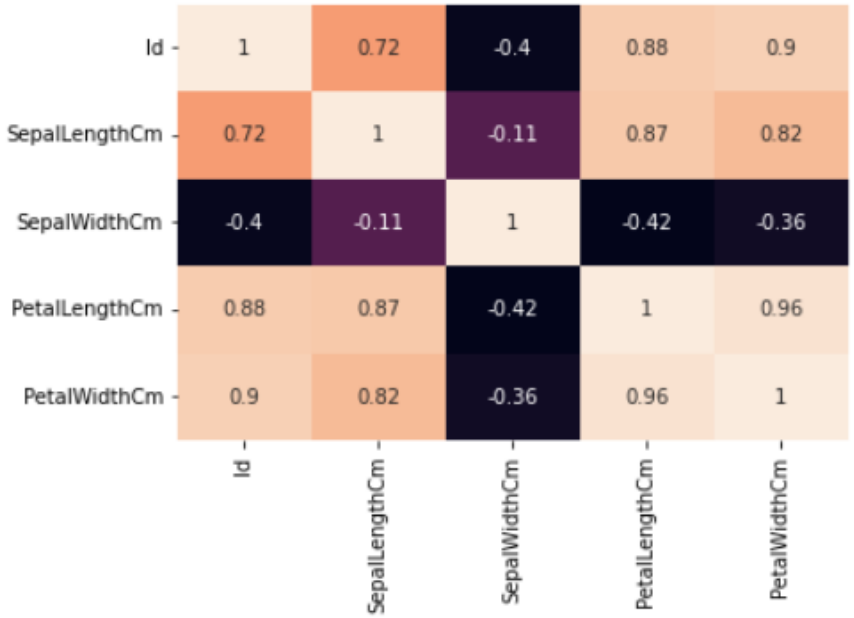
[19]:
```
plt.figure(figsize=(15,12))
plt.subplots_adjust(hspace = .25)

plt.subplot(2,2,1)
sns.boxplot(x="Species", y="SepalLengthCm", data=df)

plt.subplot(2,2,3)
sns.boxplot(x="Species", y="SepalWidthCm", data=df)
```





In [22]:
```
sns.heatmap(cbar=False,annot=True,data=df.corr())
```

Out[22]: <AxesSubplot:>



Data Preprocessing

Instruction 1 - Split of data as test and training set using Python

In [23]:
```
from sklearn.model_selection import train_test_split
x = df.iloc[:,0:4].values
y = df.iloc[:,5].values
```

In [24]:
```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
y = labelencoder.fit_transform(y)
```

In [25]:
```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.4, random_state = 0)
```

In [26]:
```
from sklearn.preprocessing import StandardScaler
scx=StandardScaler()
xtrain=scx.fit_transform(xtrain)
xtest=scx.transform(xtest)
```

Modeling

Instruction 2 - Usage of Classification Model

using Logistic Regression :

```
from sklearn.linear_model import LogisticRegression
log_regressor = LogisticRegression(random_state=0)
log_regressor.fit(xtrain,ytrain)
```

```
LogisticRegression(random_state=0)
```

```
log_predictions = log_regressor.predict(xtest)
log_predictions
```

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,
       0, 1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 2, 1, 2])
```

```
log_accuracy = log_regressor.score(xtest,ytest)
log_accuracy
```

```
0.9833333333333333
```

using Support Vector Machines :

```
from sklearn.svm import SVC
svc = SVC()
svc.fit(xtrain,ytrain)
```

```
SVC()
```

```
svc_predictions = svc.predict(xtest)
svc_predictions
```

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,
       0, 1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 2, 1, 2])
```

```
svc_accuracy = svc.score(xtest,ytest)
svc_accuracy
```

```
0.9833333333333333
```

using Naive Bayes :

```
from sklearn.naive_bayes import GaussianNB
nbs = GaussianNB()
nbs.fit(xtrain, ytrain)
```

```
GaussianNB()
```

```
nbs_predictions = nbs.predict(xtest)
nbs_predictions
```

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,
       0, 1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 2, 1, 2])
```

```
nbs_accuracy = nbs.score(xtest,ytest)
nbs_accuracy
```

```
0.9833333333333333
```

using Decision Tree :

```
from sklearn.tree import DecisionTreeClassifier
dc_tree_classifier = DecisionTreeClassifier()
dc_tree_classifier.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier()
```

```
dc_tree_predictions = dc_tree_classifier.predict(xtest)
dc_tree_predictions
```

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,
       0, 1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2])
```

```
dc_tree_accuracy = dc_tree_classifier.score(xtest,ytest)
dc_tree_accuracy
```

```
1.0
```

using Random Forest :

```
from sklearn.ensemble import RandomForestClassifier
rn_forest_classifier = RandomForestClassifier()
rn_forest_classifier.fit(xtrain, ytrain)
```

```
RandomForestClassifier()
```

```
rn_forest_predictions = rn_forest_classifier.predict(xtest)
rn_forest_predictions
```

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,
       0, 1, 2, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 2, 1, 2])
```

```
rn_forest_accuracy = rn_forest_classifier.score(xtest,ytest)
rn_forest_accuracy
```

```
0.9833333333333333
```

Instruction 3 - Observing the Models and the results

```
#models used
Models = ['Logistic Regression','Support Vector Machines','Naive Bayes','Decision Tree', 'Random Forest']
```

```
Accuracy = []

score = [log_accuracy,svc_accuracy, nbs_accuracy, dc_tree_accuracy, rn_forest_accuracy]

for i in score :
    Accuracy.append(round(i*100))
```

```
Performance_of_Models = pd.DataFrame({'Model' : Models , 'Score' : Accuracy}).sort_values(by='Score', ascending=False)
```

```
Performance_of_Models
```

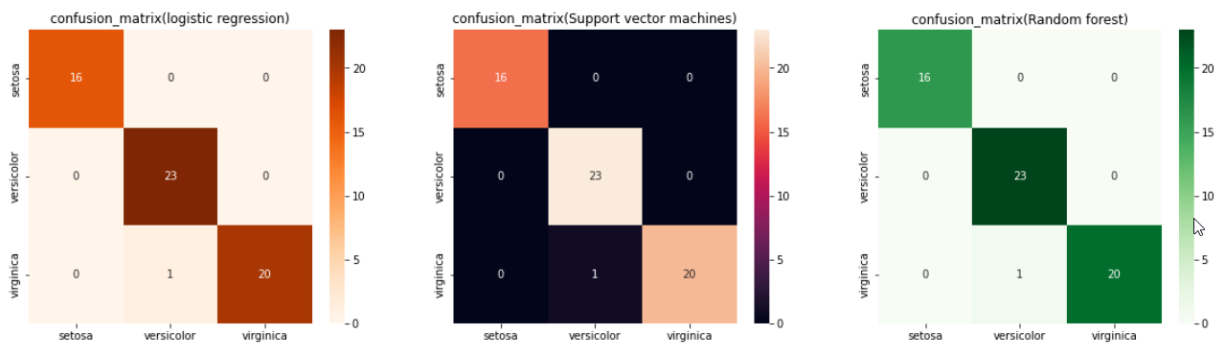
	Model	Score
3	Decision Tree	100
0	Logistic Regression	98
1	Support Vector Machines	98
2	Naive Bayes	98
4	Random Forest	98

Instruction 4 - Creating and viewing the reuslts in Confusion Matrix

```
from sklearn.metrics import accuracy_score, confusion_matrix
matrix1 = confusion_matrix(ytest, log_predictions)
matrix2 = confusion_matrix(ytest, svc_predictions)
matrix3 = confusion_matrix(ytest, rn_forest_predictions)
```

```
df1 = pd.DataFrame(matrix1,
                   index = ['setosa','versicolor','virginica'],
                   columns = ['setosa','versicolor','virginica'])

df2 = pd.DataFrame(matrix2,
                   index = ['setosa','versicolor','virginica'],
                   columns = ['setosa','versicolor','virginica'])

df3 = pd.DataFrame(matrix3,
                   index = ['setosa','versicolor','virginica'],
                   columns = ['setosa','versicolor','virginica'])
```

```
plt.figure(figsize=(20,5))
plt.subplots_adjust(hspace = .25)

plt.subplot(1,3,1)
plt.title('confusion_matrix(logistic regression)')
sns.heatmap(df1, annot=True,cmap='Oranges')

plt.subplot(1,3,2)
plt.title('confusion_matrix(Support vector machines)')
sns.heatmap(df2, annot=True)

plt.subplot(1,3,3)
plt.title('confusion_matrix(Random forest)')
sns.heatmap(df3, annot=True,cmap='Greens')

plt.show()
```



Task 2 - Analyzing the data using K-means Clustring

```
from sklearn.cluster import KMeans
```

```
iris_data=load_iris()
```

```
iris_df = pd.DataFrame(iris_data.data, columns = iris_data.feature_names)

kmeans = KMeans(n_clusters=3,init = 'k-means++',   max_iter = 100, n_init = 10, random_state = 0)

y_kmeans = kmeans.fit_predict(x)
```

```
print(kmeans.cluster_centers_)
```

```
[[ 25.5     5.006   3.418   1.464]
 [125.5     6.588   2.974   5.552]
 [ 75.5     5.936   2.77    4.26 ]]
```

```
plt.scatter(x[y_kmeans   == 0, 0], x[y_kmeans == 0, 1],s = 80, c = 'orange', label = 'Iris Setosa')
plt.scatter(x[y_kmeans   == 1, 0], x[y_kmeans == 1, 1],s = 80, c = 'green', label = 'Iris Versicolour')
plt.scatter(x[y_kmeans   == 2, 0], x[y_kmeans == 2, 1],s = 80, label = 'Iris Virginica')

plt.scatter(kmeans.cluster_centers_[:,   0], kmeans.cluster_centers_[:,1],s = 50, c = 'black')

plt.legend()
plt.show()
```