```
#Import some libraries to perform some calculations, visualization, plotting, remove warnings and other usage of functions

import pandas as pd
import matplotlib.pyplot as plt
import seaborn  as sns
from scipy import stats
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

```
#Load the dataset of Iris Species and stored in variable called iris:

iris = pd.read_csv("/content/Iris.csv")
iris
```

	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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

Basic Pandas

```
#This command gives the information of given dataset:

iris.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
```

```
#This command gives the static information of given dataset:

iris.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

```
#This command shows the columns of given dataset:

iris.columns
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
#This command shows the order pair of given dataset:

iris.shape
```

```
(150, 6)
```

```
#This command shows the first 5 rows of given dataset:

iris.head()
```

	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

```
#This command shows the last 5 rows of given dataset:

iris.tail()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

```
#This command gives the duplicated values  of given dataset:

iris.duplicated().sum()
```

```
0
```

```
#This command gives the missing values of given dataset:

iris.isnull().sum()
```

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

```
#This command counts the value of every columns individually:

iris.count()
```

```
Id               150
SepalLengthCm    150
SepalWidthCm     150
PetalLengthCm    150
```

```
PetalWidthCm     150
Species          150
dtype: int64
```

```
#This command counts the value of target column:

iris.Species.value_counts()
```

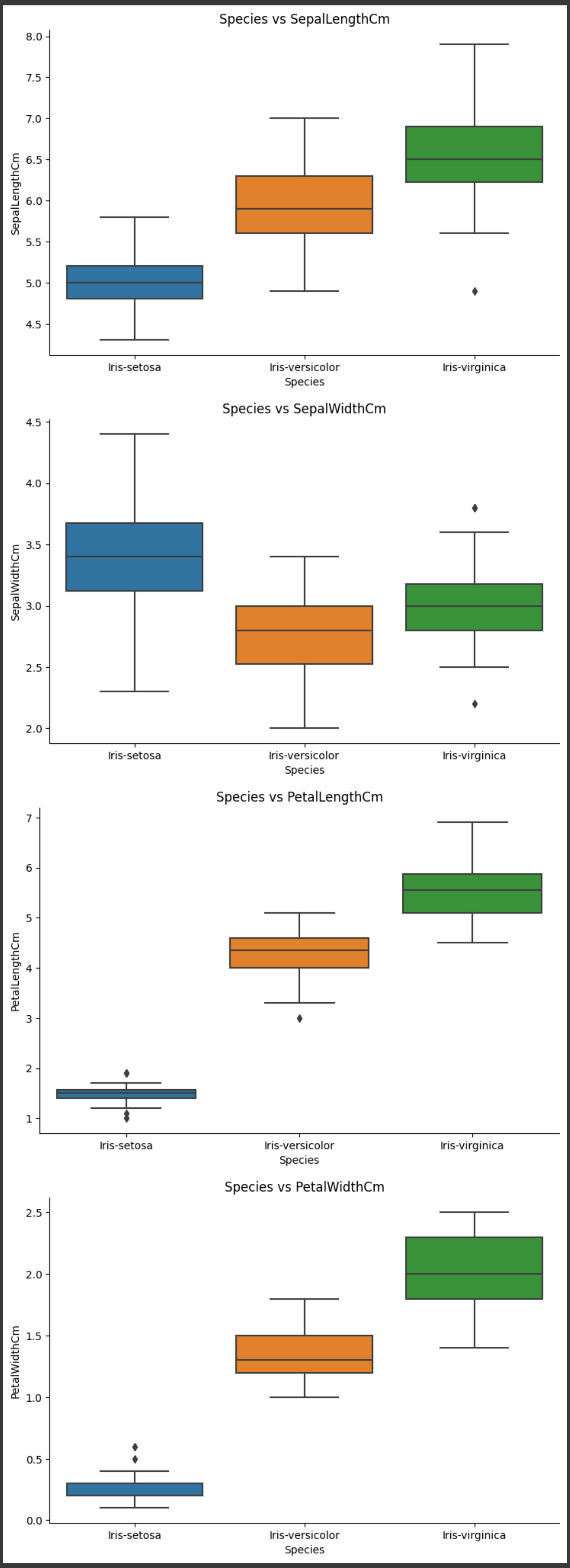
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

```
#This command groups the target column ("is_claim") into all other columns.
#Following cammand shows the relationship between target and other column:

gp = iris.groupby('Species').count()
gp
```

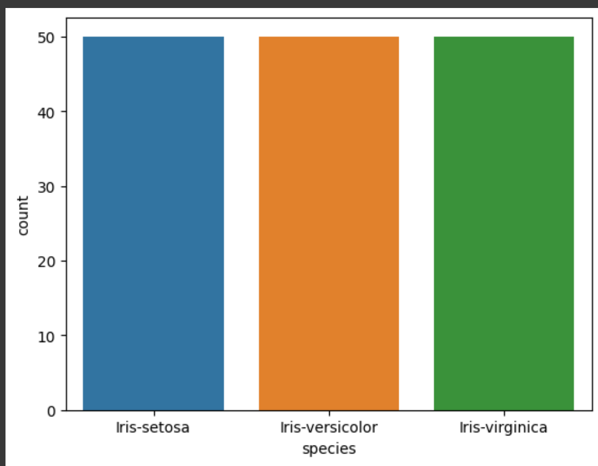
Species	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Iris-setosa	50	50	50	50	50
Iris-versicolor	50	50	50	50	50
Iris-virginica	50	50	50	50	50

Visualization:

```
# This command shows the relationship between target and other columns in the form of catplot:

columns = ['SepalLengthCm' ,    'SepalWidthCm' ,    'PetalLengthCm',    'PetalWidthCm']
for i in range(0,4,1):
  sns.catplot(x= "Species", y= columns[i],  data = iris, kind = "box", aspect = 1.5 )
  plt.title(f"Species vs %s" %columns[i] )
  plt.show()
```



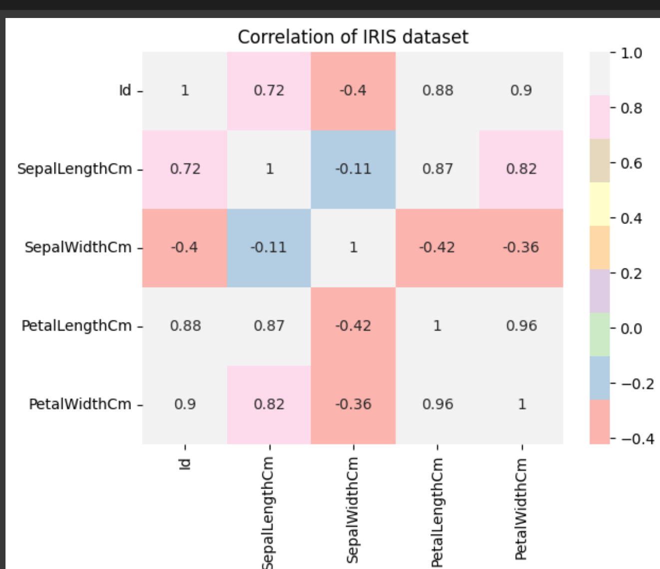
```
# This command shows the value of target column in the form of graph:

sns.countplot(x = 'Species', data = iris)
plt.xlabel('species')
plt.show()
```



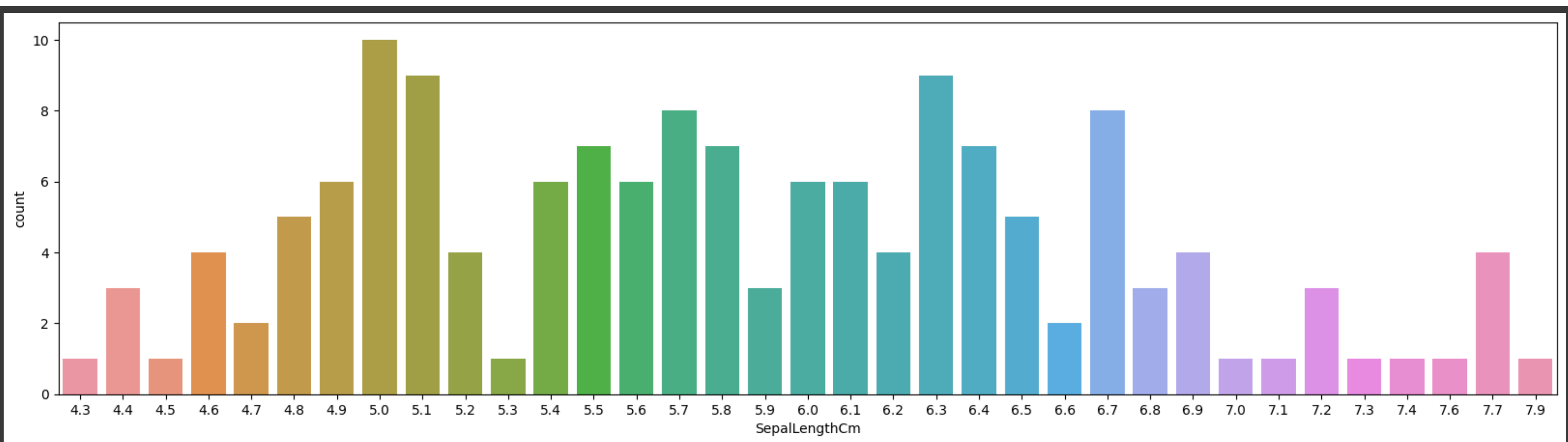

```
#This command is used to identify the co-relation of entire dataset:

iris.corr()
```

	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

```
#This command convert the co-relation into heatmap:
#Graphical view of co-relation from heatmap:

correlation = iris.corr()
sns.heatmap(correlation , annot = True , cmap = 'Pastel1')
plt.title('Correlation of IRIS dataset')
plt.show()
```

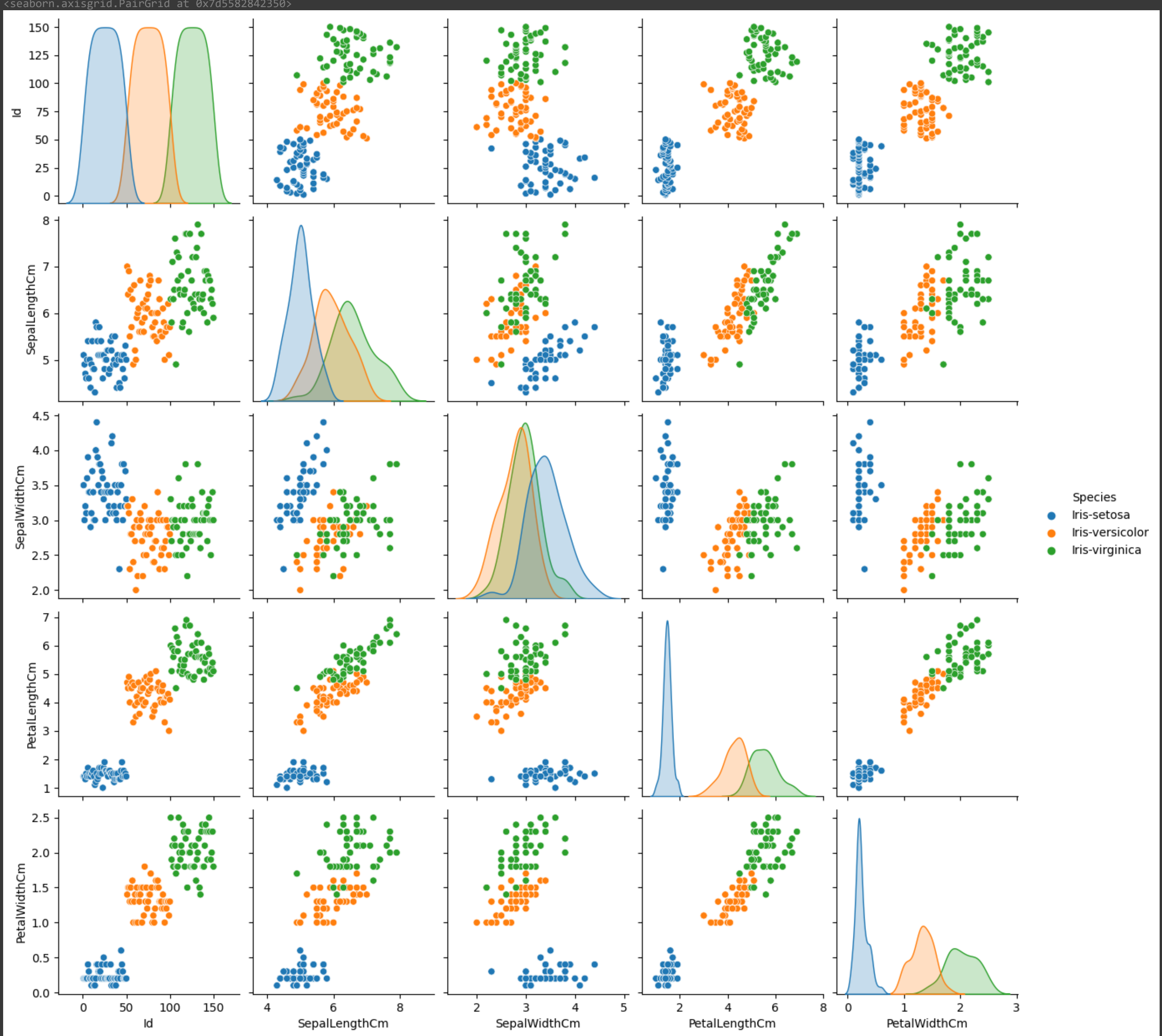



```
#This command draws the countplot in some columns:

arr =  ['SepalLengthCm' , 'SepalWidthCm' , 'PetalLengthCm' , 'PetalWidthCm']
for i in range (0, 4, 1):
  plt.figure ( figsize = (20,5))
  sns.countplot( x = arr[i]  , data = iris)
  plt.show()
```




```
#This command draw the pair plot of entire dataset:

sns.pairplot(iris, hue = 'Species', size = 2.5)
```

<seaborn.axisgrid.PairGrid at 0x7d5582842350>



Label Encoding at Species column, Split data into train and test model

```
#Import some libraries for label encoding, accurarcy and other:

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
```

```
#This command is used to convert categorical column into numeric column:

le = LabelEncoder()
iris['Species']= le.fit_transform(iris['Species'])
iris['Species']
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

```
# This command is used to Split Features and Target Variable:
X = iris.drop(columns = ['Species', 'Id'])
Y = iris['Species']

#This command split given dataset into test and train:
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25)
```

```
#This command shows the order pair of test and train

print("shape of X_train: ", X_train.shape)
print("shape of Y_train: ", Y_train.shape)
print("shape of X_test: ", X_test.shape)
print("shape of Y_test: ", Y_test.shape)
```

```
shape of X_train:  (112, 4)
shape of Y_train:  (112,)
shape of X_test:  (38, 4)
shape of Y_test:  (38,)
```

Classification Model --> "Logistic Regression"

```
#Import Logistic Regression libraries and develop model:

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

```
#Fit the model:

model.fit(X_train, Y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

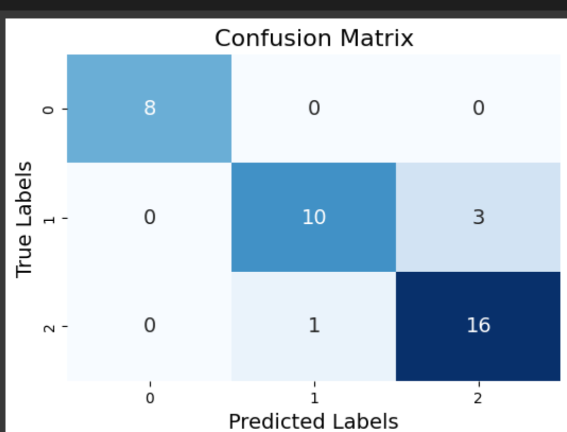
```
#Calculate accuracy score:

y_pred = model.predict(X_test)
score = accuracy_score(Y_test, y_pred)
accuracy = score*100
print(accuracy)
```

```
89.47368421052632
```

```
# This command is used to draw confussion matrix:

from sklearn.metrics import confusion_matrix
y_pred = model.predict(X_test)
print("Confusion Matrix")
cm = confusion_matrix(Y_test, y_pred)
print(cm)
```

```
Confusion Matrix
[[ 8  0  0]
 [ 0 10  3]
 [ 0  1 16]]
```

```
# This command is used to draw a heatmap of confussion matrix:

plt.figure(figsize = (6, 4))
sns.heatmap(cm, annot = True, fmt = 'd', cmap = 'Blues', cbar = False, annot_kws = {'size' : 14})
plt.xlabel('Predicted Labels', fontsize = 14)
plt.ylabel('True Labels', fontsize = 14)
plt.title('Confusion Matrix', fontsize = 16)
plt.show()
```



Classification Model --> "Decision Tree"

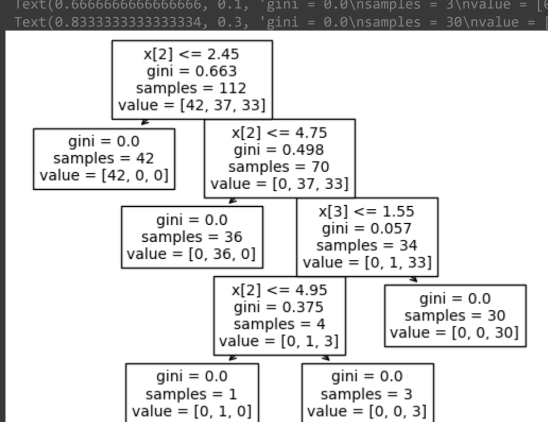
```
#Import decesion tree classifier library and develop the model:

from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
```

```
#Fit the model:

dtc.fit(X_train, Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
#Import tree libarry for plotting the tree:
from sklearn import tree

#This command is used to draw the tree:
tree.plot_tree(dtc)
```

```
[Text(0.3333333333333333, 0.9, 'x[2] <= 2.45\ngini = 0.663\nsamples = 112\nvalue = [42, 37, 33]'),
 Text(0.16666666666666666, 0.7, 'gini = 0.0\nsamples = 42\nvalue = [42, 0, 0]'),
 Text(0.5, 0.7, 'x[2] <= 4.75\ngini = 0.498\nsamples = 70\nvalue = [0, 37, 33]'),
 Text(0.3333333333333333, 0.5, 'gini = 0.0\nsamples = 36\nvalue = [0, 36, 0]'),
 Text(0.6666666666666666, 0.5, 'x[3] <= 1.55\ngini = 0.057\nsamples = 34\nvalue = [0, 1, 33]'),
 Text(0.5, 0.3, 'x[2] <= 4.95\ngini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(0.3333333333333333, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.6666666666666666, 0.1, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.8333333333333334, 0.3, 'gini = 0.0\nsamples = 30\nvalue = [0, 0, 30]')]
```



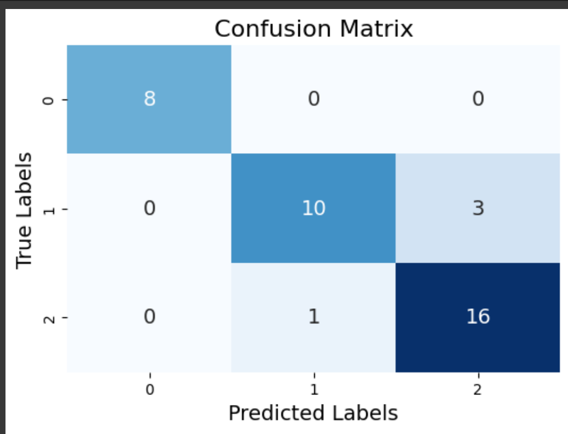

```
#This command is used to calculate accuracy of the model:

y_pred = dtc.predict(X_test)
score = accuracy_score(Y_test, y_pred)
accuracy = score*100
print(accuracy)
```

```
89.47368421052632
```

```
# This command is used to draw confussion matrix:

from sklearn.metrics import confusion_matrix
y_pred = dtc.predict(X_test)
print("Confusion Matrix")
cm_dtc = confusion_matrix(Y_test, y_pred)
print(cm_dtc)
```

```
Confusion Matrix
[[ 8  0  0]
 [ 0 10  3]
 [ 0  1 16]]
```

```
# This command is used to draw a heatmap of confussion matrix:

plt.figure(figsize = (6, 4))
sns.heatmap(cm_dtc, annot = True, fmt = 'd', cmap = 'Blues', cbar = False, annot_kws = {'size' : 14})
plt.xlabel('Predicted Labels', fontsize = 14)
plt.ylabel('True Labels', fontsize = 14)
plt.title('Confusion Matrix', fontsize = 16)
plt.show()
```



Classification Model --> "Support Vector Machine (SVM)"

```
#Import support vector machine library and develop the model:

from sklearn.svm import SVC
sss = SVC()
```

```
#Fit the model:

sss.fit(X_train, Y_train)
```

```
SVC
SVC()
```

```
#This command is used to calculate accuracy of the model:

y_pred = sss.predict(X_test)
score = accuracy_score(Y_test, y_pred)
accuracy = score*100
print(accuracy)
```

```
92.10526315789474
```

```
#This command is used to calculate accuracy of the model:

from sklearn.metrics import confusion_matrix
y_pred = sss.predict(X_test)
print("Confusion Matrix")
cm_sss = confusion_matrix(Y_test, y_pred)
print(cm_sss)
```

```
Confusion Matrix
[[ 8  0  0]
 [ 0 11  2]
 [ 0  1 16]]
```

```
# This command is used to draw a heatmap of confussion matrix:

plt.figure(figsize = (6, 4))
sns.heatmap(cm_sss, annot = True, fmt = 'd', cmap = 'Blues', cbar = False, annot_kws = {'size' : 14})
plt.xlabel('Predicted Labels', fontsize = 14)
plt.ylabel('True Labels', fontsize = 14)
plt.title('Confusion Matrix', fontsize = 16)
plt.show()
```