Assignment no. 6

-Shivam Tawari A-58

- Iris Dataset :

Iris dataset is the multivariate dataset which was introduced by British statistician and biologist Ronald Fisher.

The attributes of Iris data are :

1. sepal_length
2. sepal_width
3. petal_length
4. petal_width
5. setosa
6. versicolour
7. Virginica

- Naive Bayes Classifier : Bayesian Theorem

NB Classifier is used as a knowledge accumulator during training and testing data. This helps to classify and sense unseen data. NB classifier calculate the most possible output based on input. Naive Bayes is a better probabilistic classifier it consider the presence of a particular features of a class.

Relationships using Bayesian Network concept:

① Let D be the training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, $X = (x_1, x_2, x_3, \ldots, x_n)$ depicting n measurements made on the tuple from n attributes, respectively, $A_1, A_2, \ldots A_n$.

② If there are m classes, $C_1, C_2, C_3, \ldots C_m$, given tuple x, the classifier will predict that x belongs to the class having the highest posterior probability, conditioned on x. That is, the Naive Bayes classifier predicts that tuple x belongs to the class C_i if and only if:

$$P(C_i|x) > P(C_j|x) \text{ for } 1 \leq j \leq m, j \neq i.$$

Complexities:

① The main complexity of Naive Bayes is the assumption of independent predictor features. Naive Bayes implicitly assumes that all the iris dataset attributes are mutually independent.

② If a categorical variable has a category in the dataset, which was not observed in training dataset, then the model will assign a zero probability and will be unable to make a prediction. This is often known as zero frequency.

```
print("Performesd by Shivam Tawari_AI dept.,Assignment-6_MLA")
```

```
Performesd by Shivam Tawari_AI dept.,Assignment-6_MLA
```

```
from google.colab import drive
drive.mount("/gdrive")
%cd /gdrive
```

```
Mounted at /gdrive
/gdrive
```

```
ls
```

```
MyDrive/  Shareddrives/
```

```
cd/gdrive/My Drive/Iris flower
```

```
/gdrive/My Drive/Iris flower
```

```
ls
```

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

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import os
from sklearn.preprocessing import LabelEncoder
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

```
input_path='/gdrive/My Drive/Iris flower/Iris.csv'
df=pd.read_csv(input_path)
df.head()
```

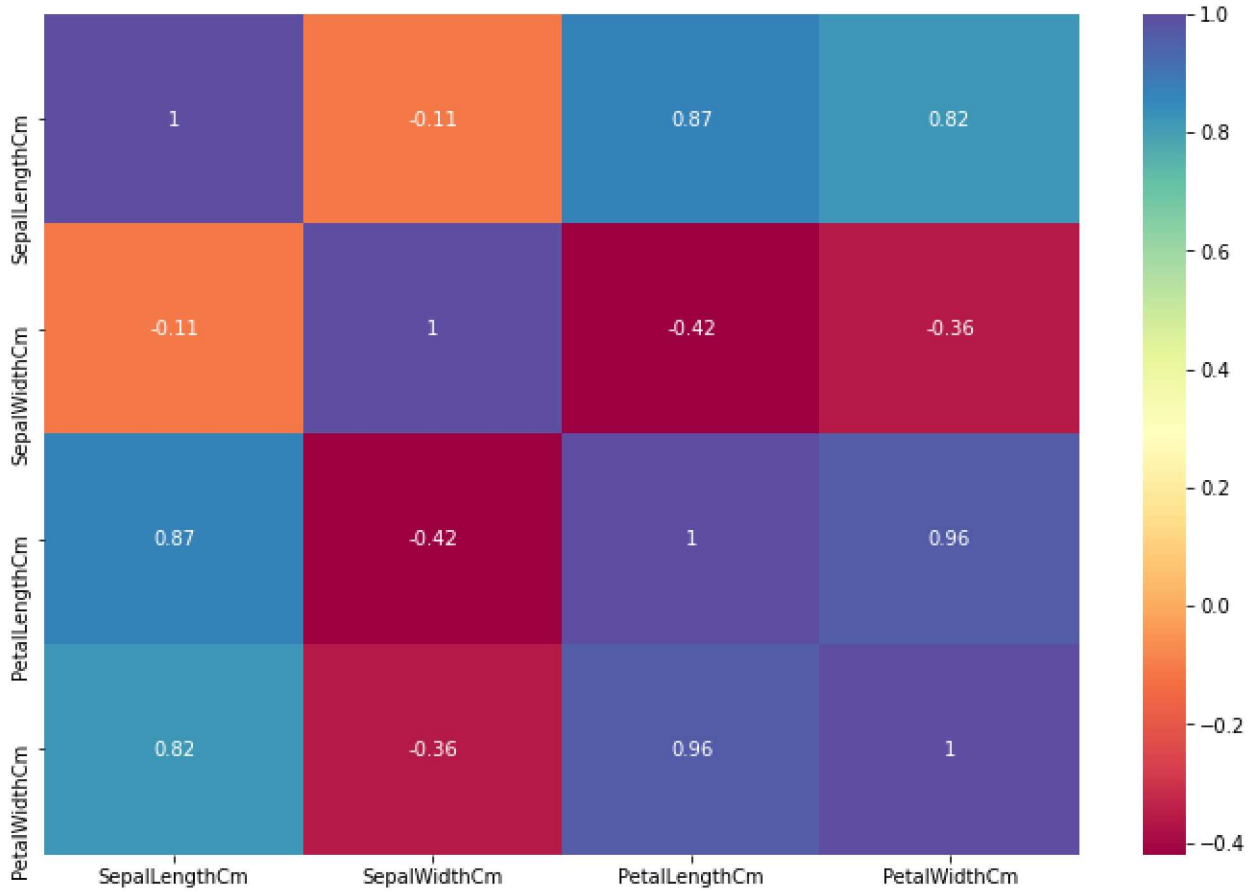
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa

```
df.drop('Id',axis=1,inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

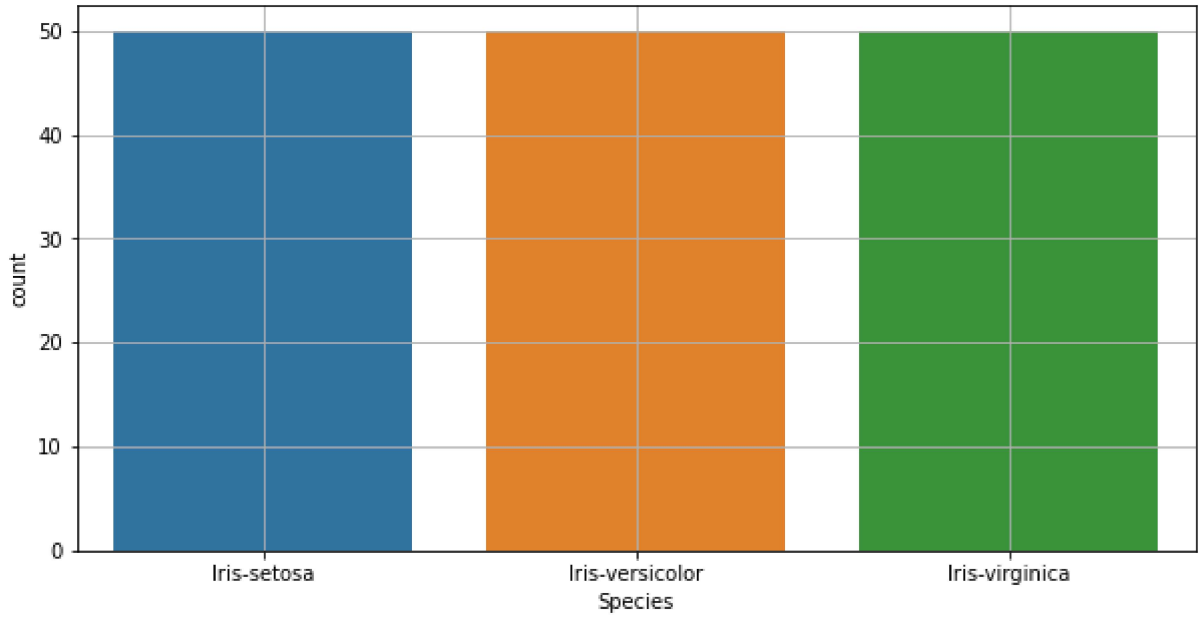
```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=True,cmap=plt.cm.Spectral)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5abf47a390>
```

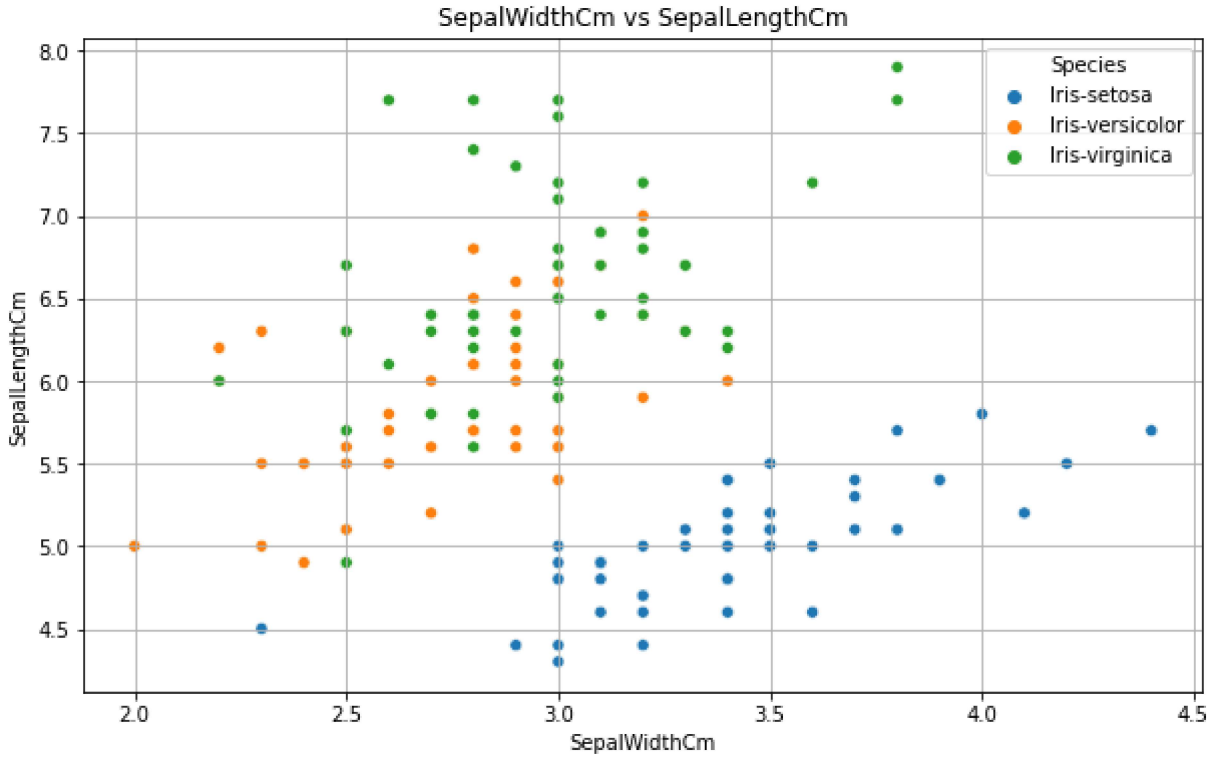


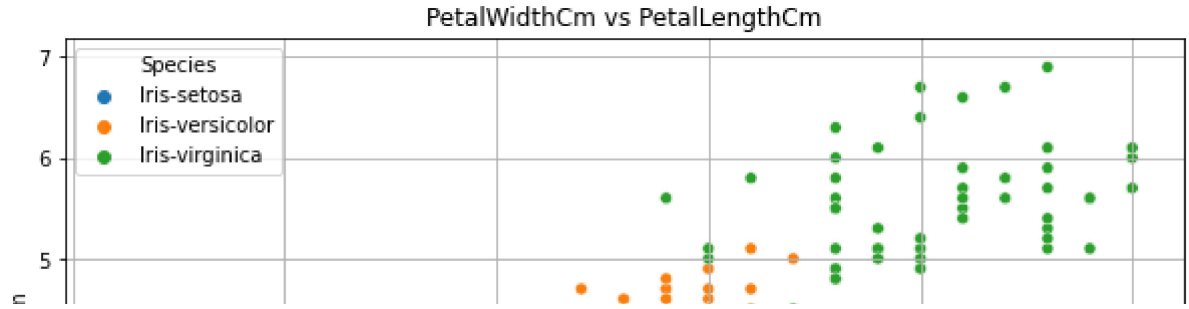

```
plt.figure(figsize=(10,5))
```

```
sns.countplot(x='Species',data=df)
plt.grid()
```



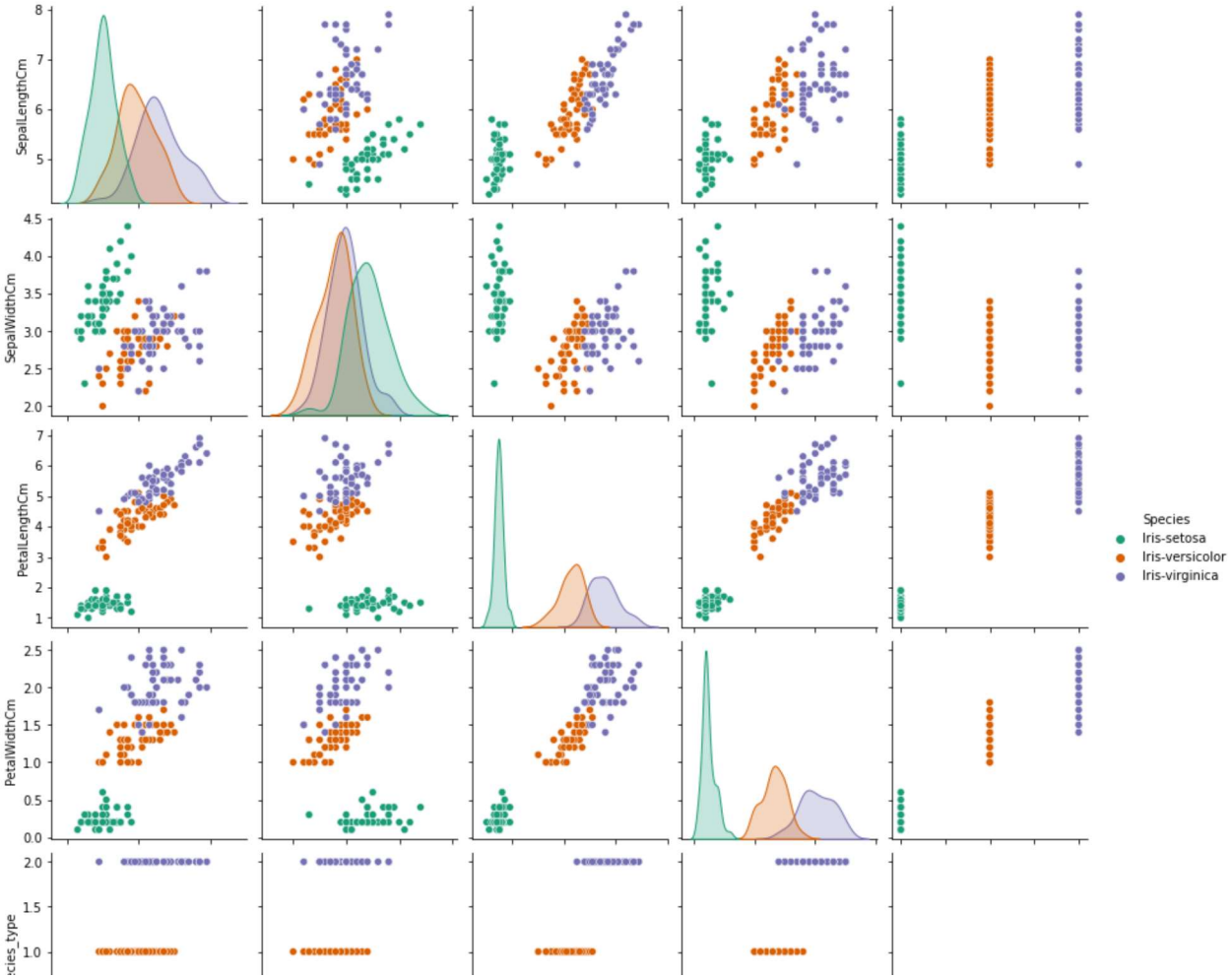

```
le=LabelEncoder()
df['Species_type']=le.fit_transform(df['Species'])


def scatterplot(X,Y):
    sns.scatterplot(data=df,x=X,y=Y,hue='Species')
    plt.title(X+" vs "+Y)
    plt.grid()
    plt.show()


plt.figure(figsize=(10,6))
scatterplot('SepalWidthCm','SepalLengthCm')
plt.figure(figsize=(10,6))
scatterplot('PetalWidthCm','PetalLengthCm')
plt.tight_layout()
```





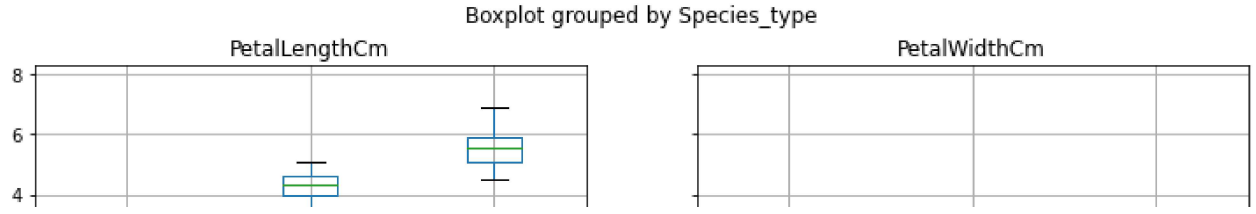

```
sns.pairplot(df,hue='Species',palette='Dark2')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:306: UserWarning: Datase
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:306: UserWarning: Datase
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:306: UserWarning: Datase
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.PairGrid at 0x7f5aa185ca90>
```

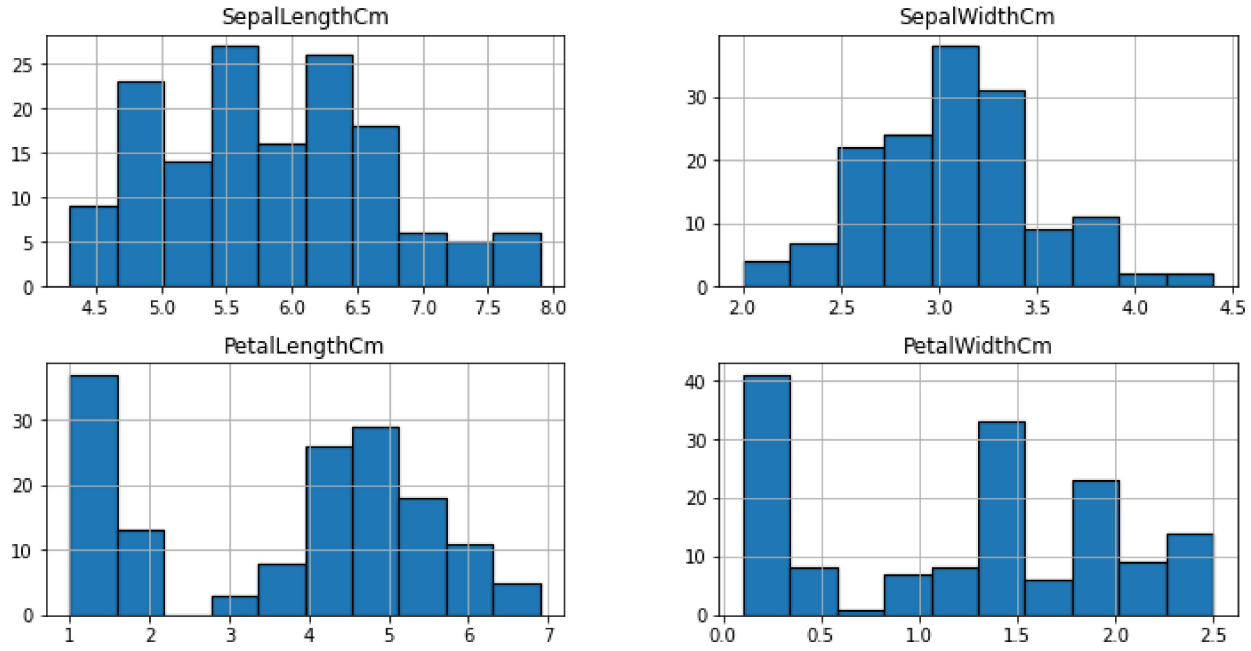



```
df.boxplot(by="Species_type", figsize=(12, 6))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5a9ed0f190>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5a9ed22710>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5a9ecc7a50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5a9ec80b50>]],
      dtype=object)
```

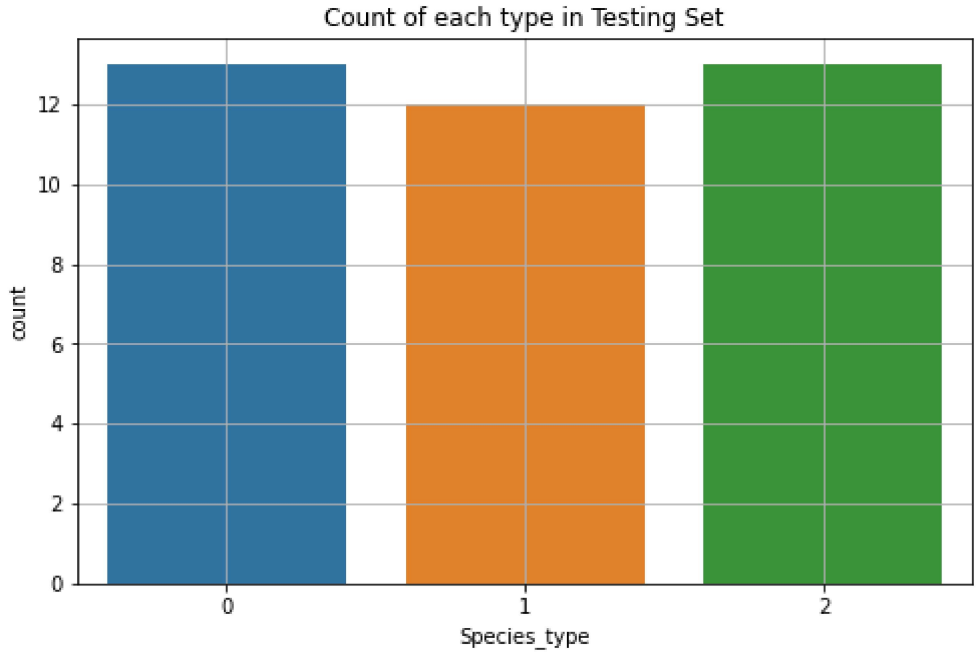


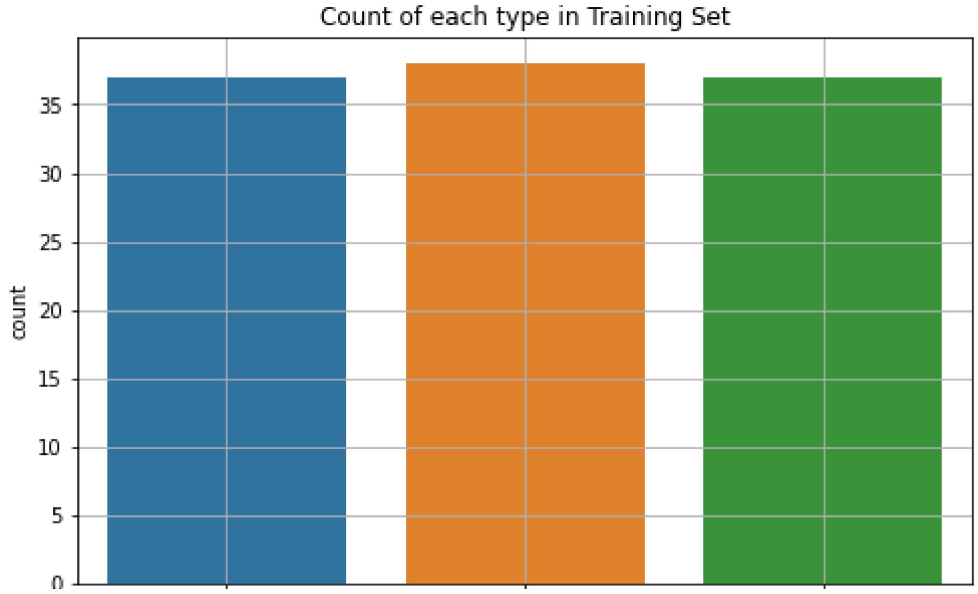

```
df.drop('Species_type',axis=1).hist(edgecolor='black', linewidth=1.2)
fig=plt.gcf()
fig.set_size_inches(12,6)
plt.show()
```




```
X_train,X_test,Y_train,Y_test = train_test_split(df.drop(['Species_type','Species'],axis=1),d
```

```
plt.figure(figsize=(8,5))
sns.countplot(x=Y_test)
plt.title("Count of each type in Testing Set")
plt.grid()
plt.figure(figsize=(8,5))
sns.countplot(x=Y_train)
plt.title("Count of each type in Training Set")
plt.grid()
```






```
models = ['random_forest','svm','decision_tree','logistic_regression','knn']
model_train_acc=[]
model_test_acc=[]


svm= SVC()
svm.fit(X_train,Y_train)
y_hat_train = svm.predict(X_train)
y_hat_test = svm.predict(X_test)
train_acc = np.round(accuracy_score(y_hat_train,Y_train),3)
test_acc =  np.round(accuracy_score(y_hat_test,Y_test),3)

model_train_acc.append(train_acc)
model_test_acc.append(test_acc)

print("Accuracy Score on train data using SVM: ",train_acc)
print("Accuracy Score on test data using SVM: ",test_acc)
```

```
     Accuracy Score on train data using SVM:  0.955
     Accuracy Score on test data using SVM:  0.947
```

```
rfc = RandomForestClassifier()
rfc.fit(X_train,Y_train)
y_hat_train = rfc.predict(X_train)
y_hat_test = rfc.predict(X_test)
train_acc = np.round(accuracy_score(y_hat_train,Y_train),3)
test_acc =  np.round(accuracy_score(y_hat_test,Y_test),3)

model_train_acc.append(train_acc)
model_test_acc.append(test_acc)

print("Accuracy Score on train data using RandomForest: ",train_acc)
print("Accuracy Score on test data using RandomForest: ",test_acc)
```

```
     Accuracy Score on train data using RandomForest:  1.0
     Accuracy Score on test data using RandomForest:  0.895
```

```
tree = DecisionTreeClassifier()
tree.fit(X_train,Y_train)
y_hat_train = tree.predict(X_train)
y_hat_test = tree.predict(X_test)
train_acc = np.round(accuracy_score(y_hat_train,Y_train),3)
test_acc =  np.round(accuracy_score(y_hat_test,Y_test),3)

model_train_acc.append(train_acc)
model_test_acc.append(test_acc)

print("Accuracy Score on train data using Decision tree: ",train_acc)
print("Accuracy Score on test data using Decision tree: ",test_acc)
```

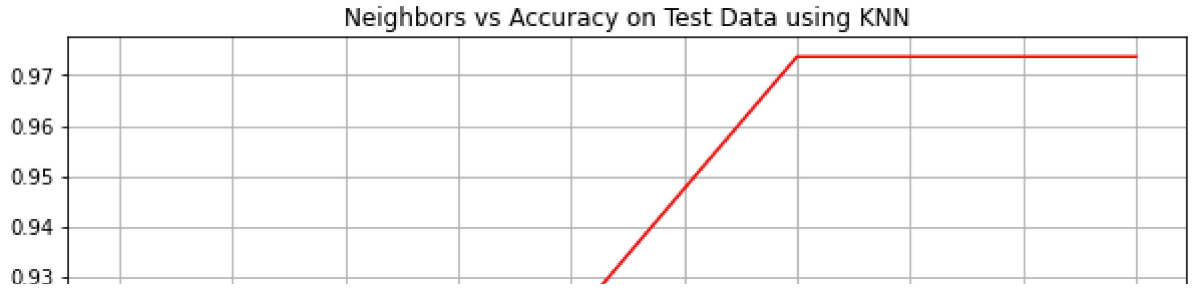
```
     Accuracy Score on train data using Decision tree:  1.0
     Accuracy Score on test data using Decision tree:  0.895
```

```
lr = LogisticRegression(max_iter=200)
lr.fit(X_train,Y_train)
y_hat_train = lr.predict(X_train)
y_hat_test = lr.predict(X_test)
train_acc = np.round(accuracy_score(y_hat_train,Y_train),3)
test_acc =  np.round(accuracy_score(y_hat_test,Y_test),3)

model_train_acc.append(train_acc)
model_test_acc.append(test_acc)

print("Accuracy Score on train data using Logistic Regression: ",train_acc)
print("Accuracy Score on test data using Logistic Regression: ",test_acc)
```

```
     Accuracy Score on train data using Logistic Regression:  0.973
```

```
 Accuracy Score on test data using Logistic Regression:  0.921
```

```
def select_neighbors():
    knn_train_acc=[]
    knn_test_acc=[]
    for i in range(1,11):
        knn = KNeighborsClassifier(n_neighbors=i)
        knn.fit(X_train,Y_train)
        y_hat_train = knn.predict(X_train)
        y_hat_test = knn.predict(X_test)
        knn_train_acc.append(accuracy_score(y_hat_train,Y_train))
        knn_test_acc.append(accuracy_score(y_hat_test,Y_test))

    return knn_train_acc,knn_test_acc


knn_train,knn_test = select_neighbors()
x = np.linspace(1,10,10)

plt.figure(figsize=(10,4))
plt.plot(x,knn_test,color='red')
plt.title("Neighbors vs Accuracy on Test Data using KNN")
plt.xticks(x)
plt.grid()

plt.figure(figsize=(10,4))
plt.plot(x,knn_train,color='red')
plt.title("Neighbors vs Accuracy on Train Data using KNN")
plt.xticks(x)
plt.grid()
```




```
train_acc = np.round(knn_train[6],3)
test_acc = np.round(knn_test[6],3)

model_train_acc.append(train_acc)
model_test_acc.append(test_acc)

print("Accuracy Score on train data using KNN: ",train_acc)
print("Accuracy Score on test data using KNN: ",test_acc)
```

```
Accuracy Score on train data using KNN:  0.982
Accuracy Score on test data using KNN:  0.974
```