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

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

	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

```
df.shape
```

```
(150, 6)
```

```
print(f"Length of dataset --> {len(df)}")
```

```
Length of dataset --> 150
```

```
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
```

```
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

```
df.dtypes
```

```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
```

```
PetalWidthCm     float64
Species           object
dtype: object
```

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

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

```
df.isna().sum()
```

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

```
print(f"Minimum value:\n{df.min()}")
```

```
Minimum value:
Id                         1
SepalLengthCm            4.3
SepalWidthCm             2.0
PetalLengthCm            1.0
PetalWidthCm             0.1
Species          Iris-setosa
dtype: object
```

```
print(f"Maximum value:\n{df.max()}")
```

```
Maximum value:
Id                            150
SepalLengthCm                 7.9
SepalWidthCm                  4.4
PetalLengthCm                 6.9
PetalWidthCm                  2.5
Species          Iris-virginica
dtype: object
```

```
print(f"Average value:\n{df.mean()}")
```

```
Average value:
Id               75.500000
SepalLengthCm     5.843333
SepalWidthCm      3.054000
PetalLengthCm     3.758667
PetalWidthCm      1.198667
dtype: float64
<ipython-input-12-acf45c2ab497>:1: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future ve
  print(f"Average value:\n{df.mean()}")
```

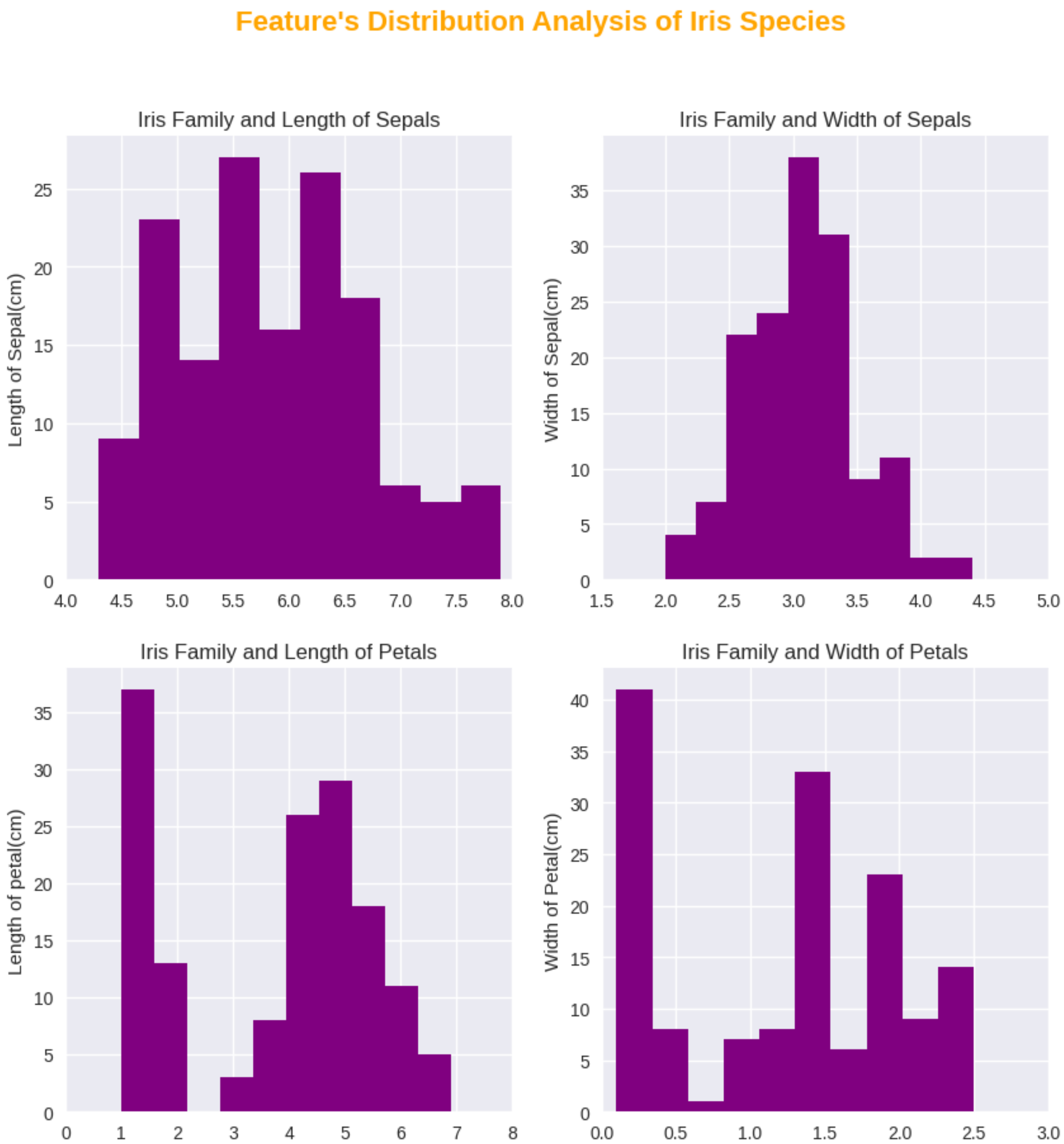
```
df.drop(['Id'],inplace = True, axis=1)
df.head
```

```
<bound method NDFrame.head of      SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm         Species
0              5.1           3.5            1.4           0.2     Iris-setosa
1              4.9           3.0            1.4           0.2     Iris-setosa
2              4.7           3.2            1.3           0.2     Iris-setosa
3              4.6           3.1            1.5           0.2     Iris-setosa
4              5.0           3.6            1.4           0.2     Iris-setosa
..             ...           ...            ...           ...             ...
145            6.7           3.0            5.2           2.3  Iris-virginica
146            6.3           2.5            5.0           1.9  Iris-virginica
147            6.5           3.0            5.2           2.0  Iris-virginica
148            6.2           3.4            5.4           2.3  Iris-virginica
149            5.9           3.0            5.1           1.8  Iris-virginica

[150 rows x 5 columns]>
```

```
plt.style.use('seaborn')
fig , ((ax0, ax1),(ax2, ax3)) = plt.subplots(nrows=2,
                                             ncols=2,
                                             figsize=(10,10))
ax0.hist(df["SepalLengthCm"],
         color="purple");
ax0.set_xlim(4,8)
ax0.set(title="Iris Family and Length of Sepals",
```

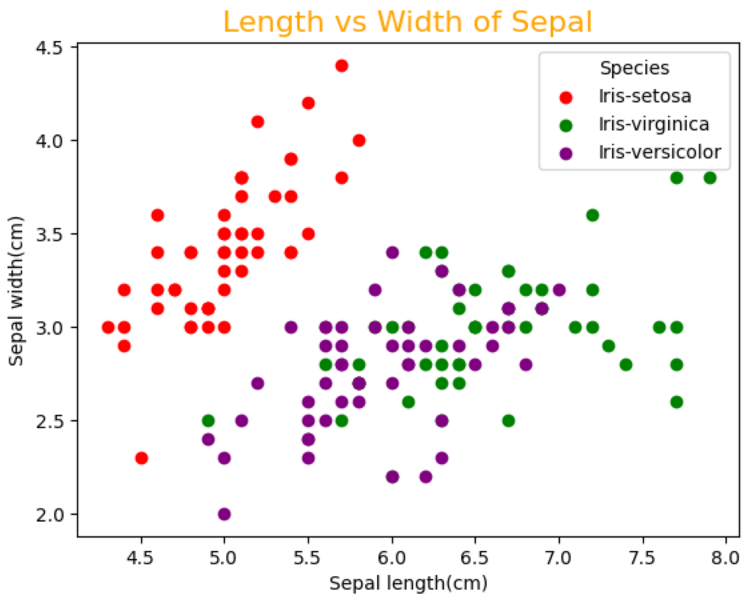
```
       ylabel="Length of Sepal(cm)");

ax1.hist(df["SepalWidthCm"],
         color="purple");
ax1.set_xlim(1.5,5)
ax1.set(title="Iris Family and Width of Sepals",
       ylabel="Width of Sepal(cm)");

ax2.hist(df["PetalLengthCm"],
         color="purple");
ax2.set_xlim(0,8)
ax2.set(title="Iris Family and Length of Petals",
       ylabel="Length of petal(cm)");

ax3.hist(df["PetalWidthCm"],
         color="purple");
ax3.set_xlim(0,3)
ax3.set(title="Iris Family and Width of Petals",
       ylabel="Width of Petal(cm)");

# Add a title to figure

fig.suptitle("Feature's Distribution Analysis of Iris Species",
            fontsize =16,
            fontweight="bold",
            color="orange");
```

```
<ipython-input-14-934c1bf6e55c>:1: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6,
  plt.style.use('seaborn')
```

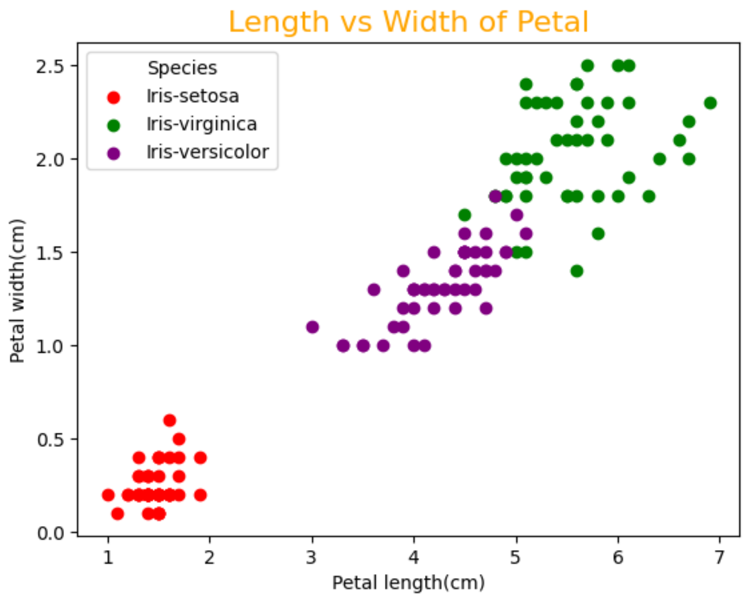



```
plt.style.use('default')
colors = ['red', 'green', 'purple']
species = ['Iris-setosa', 'Iris-virginica', 'Iris-versicolor']
```

```
for i in range(3):
    filter_species  = df[df.Species == species[i]]
    plt.scatter(x=filter_species.SepalLengthCm,
                y=filter_species.SepalWidthCm,
                c=colors[i],
                label=species[i])
plt.xlabel("Sepal length(cm)")
plt.ylabel("Sepal width(cm)")
plt.legend(title="Species");
plt.title("Length vs Width of Sepal",
          color="orange",
          fontsize=16);
```

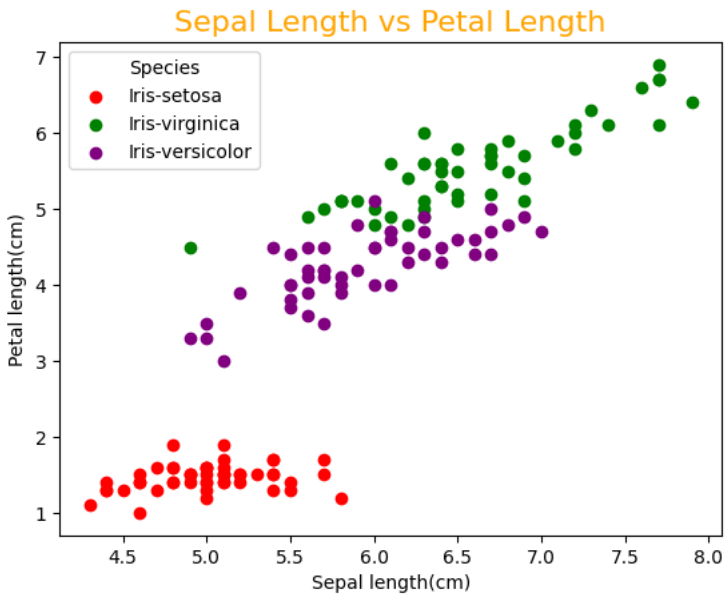



```
for i in range(3):
    filter_species  = df[df.Species == species[i]]
    plt.scatter(x=filter_species.PetalLengthCm,
                y=filter_species.PetalWidthCm,
                c=colors[i],
                label=species[i])
plt.xlabel("Petal length(cm)")
plt.ylabel("Petal width(cm)")
plt.legend(title="Species");
plt.title("Length vs Width of Petal",
          color="orange",
          fontsize=16);
```

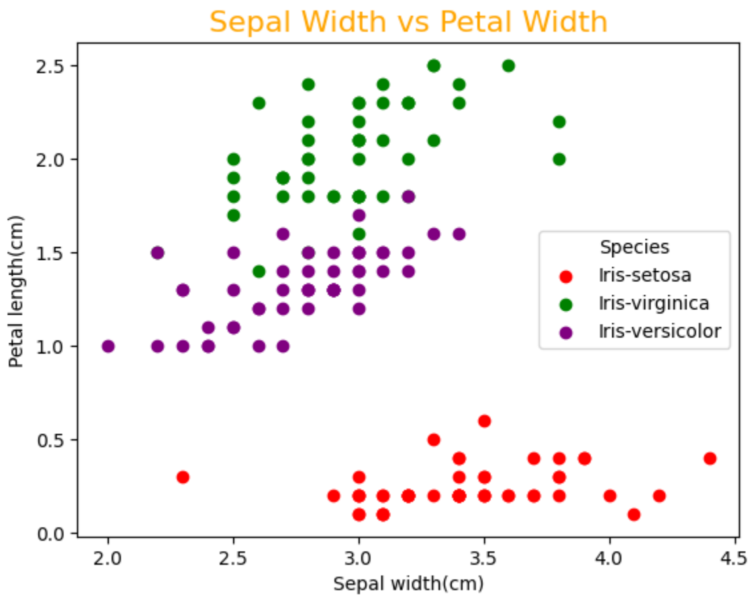



```
for i in range(3):
    filter_species  = df[df.Species == species[i]]
    plt.scatter(x=filter_species.SepalLengthCm,
                y=filter_species.PetalLengthCm,
```

```
                c=colors[i],
                label=species[i])
plt.xlabel("Sepal length(cm)")
plt.ylabel("Petal length(cm)")
plt.legend(title="Species");
plt.title("Sepal Length vs Petal Length",
          color="orange",
          fontsize=16);
```




```
for i in range(3):
    filter_species  = df[df.Species == species[i]]
    plt.scatter(x=filter_species.SepalWidthCm,
                y=filter_species.PetalWidthCm,
                c=colors[i],
                label=species[i])
plt.xlabel("Sepal width(cm)")
plt.ylabel("Petal length(cm)")
plt.legend(title="Species");
plt.title("Sepal Width vs Petal Width",
          color="orange",
          fontsize=16);
```

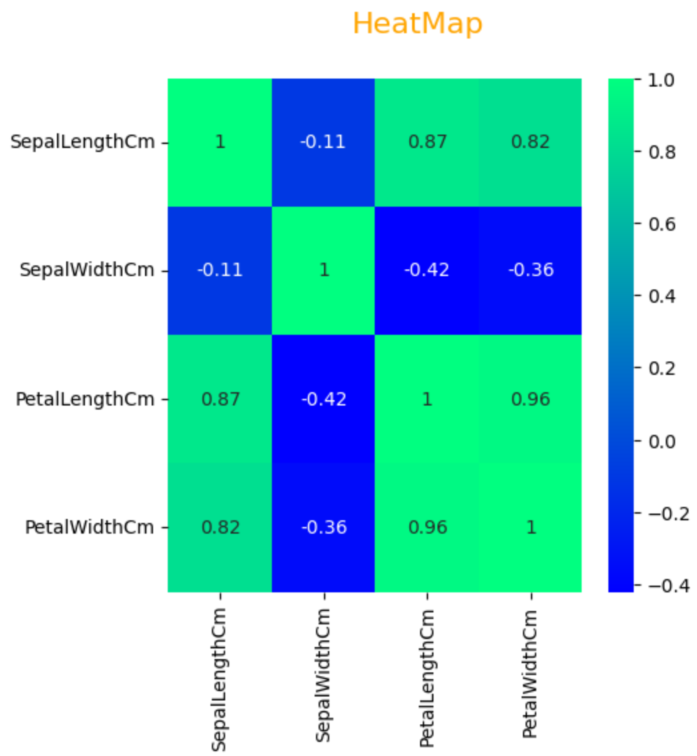



```
corr = df.corr()
corr
```

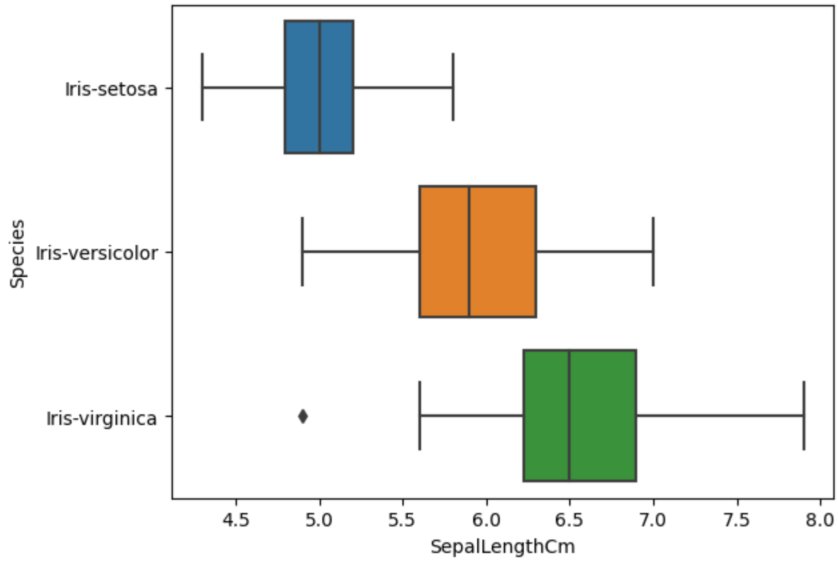
<ipython-input-19-4381f08f6434>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  corr = df.corr()

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954

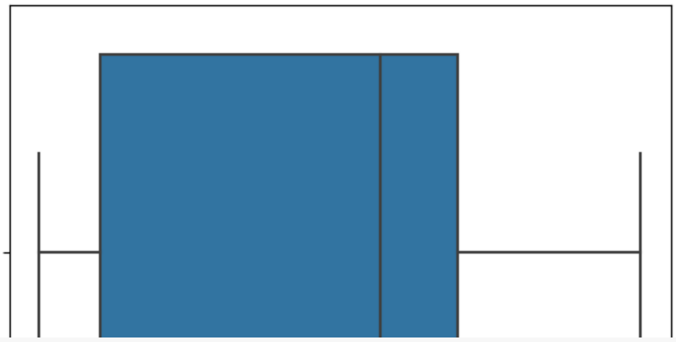
```
fig, ax = plt.subplots(figsize=(5,5))
sns.heatmap(corr, annot=True, ax=ax, cmap='winter');
fig.suptitle(t="HeatMap",
             color="orange",
             fontsize=16);
```



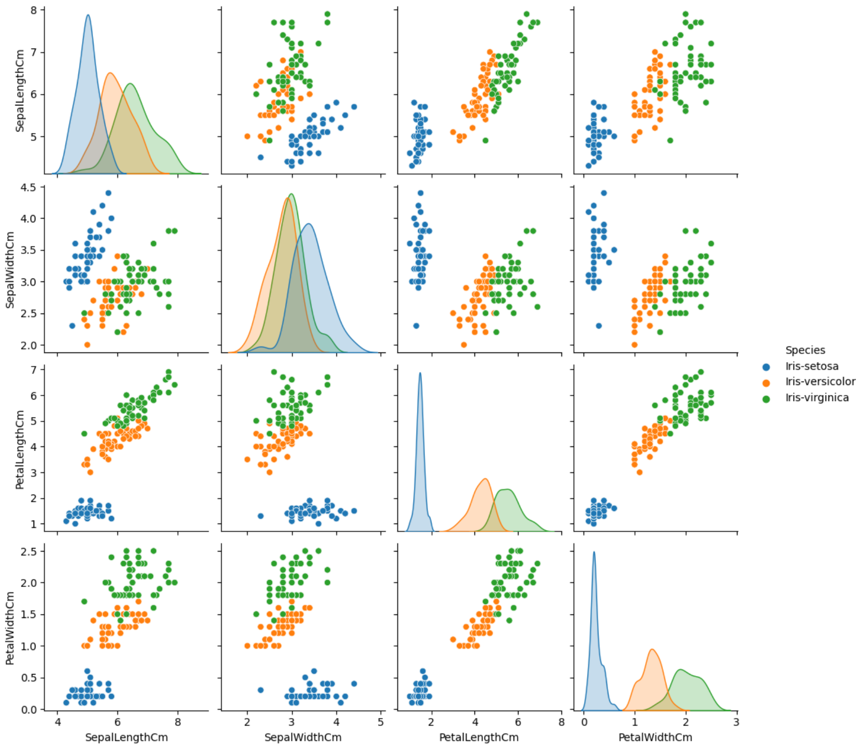

```
sns.boxplot(x="SepalLengthCm", y="Species", data=df);
```




```
sns.boxplot(x="PetalLengthCm", data=df);
```



```
sns.pairplot(data = df, hue = "Species");
```




```
df.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
#Selecting one tuple of class="iris-setosa" for prediction purpose
preds_data1 = df.iloc[32]
preds_data1
```

```
SepalLengthCm            5.2
SepalWidthCm             4.1
PetalLengthCm            1.5
PetalWidthCm             0.1
Species          Iris-setosa
Name: 32, dtype: object
```

```
#Selecting one tuple of class="iris-versicolor" for prediction purpose
preds_data2 = df.iloc[76]
preds_data2
```

```
SepalLengthCm                6.8
SepalWidthCm                 2.8
PetalLengthCm                4.8
PetalWidthCm                 1.4
Species          Iris-versicolor
Name: 76, dtype: object
```

```
#Selecting one tuple of class="iris-virginica" for prediction purpose
preds_data3 = df.iloc[132]
preds_data3
```

```
SepalLengthCm               6.4
SepalWidthCm                2.8
PetalLengthCm               5.6
PetalWidthCm                2.2
Species          Iris-virginica
Name: 132, dtype: object
```

```
df.shape
```

```
(150, 5)
```

```
# Removing tuples with index 32, 76, 132 from our dataset
df.drop([32, 76, 132], inplace=True)
df.shape
```

```
(147, 5)
```

```
# Making X and y features

X = df.drop("Species", axis=1)
y = df["Species"]
```

```
X.head
```

```
<bound method NDFrame.head of      SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0              5.1           3.5            1.4           0.2
1              4.9           3.0            1.4           0.2
2              4.7           3.2            1.3           0.2
3              4.6           3.1            1.5           0.2
4              5.0           3.6            1.4           0.2
..             ...           ...            ...           ...
145            6.7           3.0            5.2           2.3
146            6.3           2.5            5.0           1.9
147            6.5           3.0            5.2           2.0
148            6.2           3.4            5.4           2.3
149            5.9           3.0            5.1           1.8

[147 rows x 4 columns]>
```

```
y.head
```

```
<bound method NDFrame.head of 0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
            ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 147, dtype: object>
```

```
X.dtypes
```

```
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
dtype: object
```

```
y.dtypes
```

```
dtype('O')
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["Encoded species"] = le.fit_transform(y.ravel()) # Appending to our dataset
y = le.fit_transform(y.ravel())
y
```

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

```
df[df["Encoded species"] == 1].head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	Encoded species
50	7.0	3.2	4.7	1.4	Iris-versicolor	1
51	6.4	3.2	4.5	1.5	Iris-versicolor	1
52	6.9	3.1	4.9	1.5	Iris-versicolor	1
53	5.5	2.3	4.0	1.3	Iris-versicolor	1
54	6.5	2.8	4.6	1.5	Iris-versicolor	1

```
Encoded_class = pd.DataFrame({'Species': ["Iris-setosa", "Iris-versicolor", "Iris-virginica"],
                              'Encoded': [0, 1, 2]})
Encoded_class
```

	Species	Encoded
0	Iris-setosa	0
1	Iris-versicolor	1
2	Iris-virginica	2

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

np.random.seed(35)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2)
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
score = clf.score(X_test, y_test)
print(f"Model Accuracy on Test Data = {score*100:2f}%")
```

```
Model Accuracy on Test Data = 93.333333%
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

y_preds = clf.predict(X_test)
print(f"Classfifcation Report:\n\n{classification_report(y_test, y_preds)}")
```

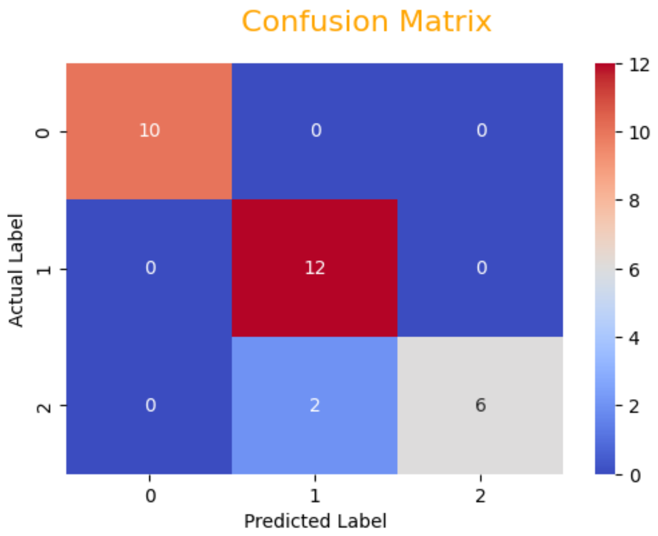
```
Classfifcation Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.86      1.00      0.92        12
           2       1.00      0.75      0.86         8

    accuracy                           0.93        30
   macro avg       0.95      0.92      0.93        30
```

```
weighted avg       0.94      0.93      0.93        30
```

```
cf_matrix = confusion_matrix(y_test, y_preds)
print(f"Confusion Matrix:\n\n{cf_matrix}")
```

```
Confusion Matrix:

[[10  0  0]
 [ 0 12  0]
 [ 0  2  6]]
```

```
fig, ax = plt.subplots(figsize=(6,4))
sns.heatmap(cf_matrix, annot=True, cmap='coolwarm')
fig.suptitle(t="Confusion Matrix",
             color="orange",
             fontsize=16);
ax.set(xlabel="Predicted Label",
       ylabel="Actual Label");
```

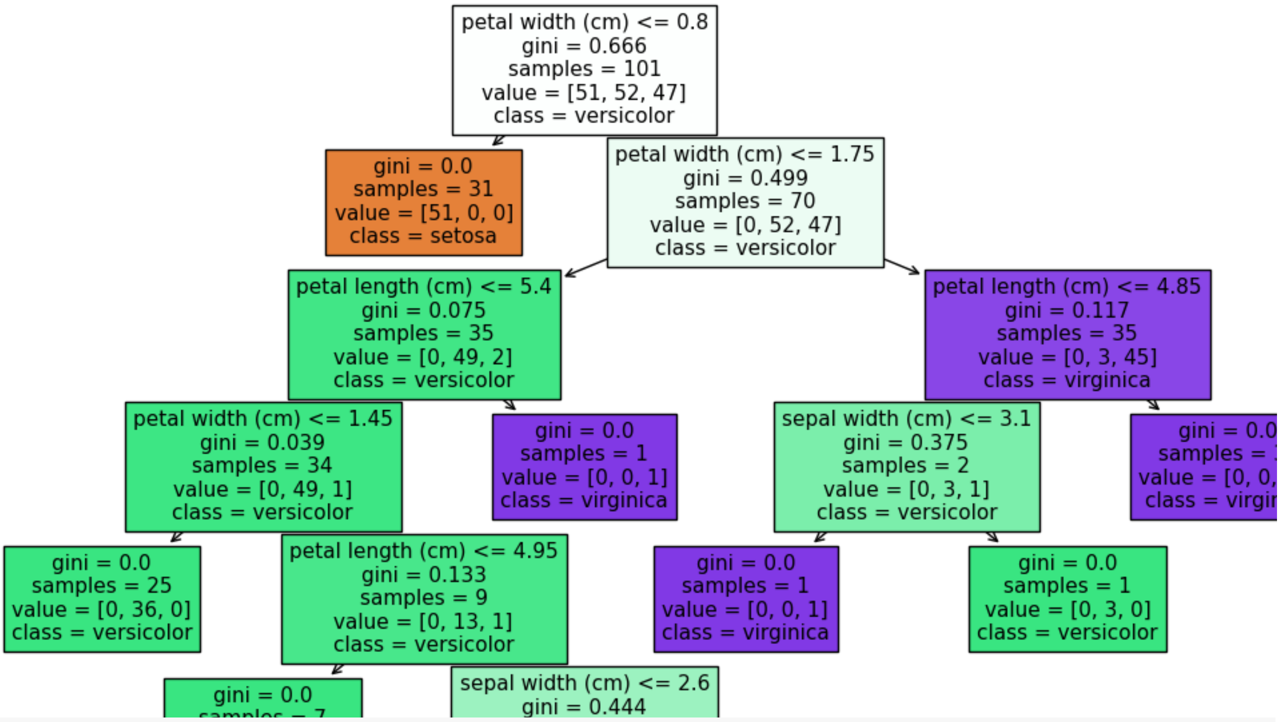



```
print(f"Accuracy Score:\n\n{accuracy_score(y_test, y_preds)*100:2f}%")
```

```
Accuracy Score:

93.333333%
```

```
from sklearn import tree
```

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Load iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Create a random forest classifier
clf = RandomForestClassifier(n_estimators=10, random_state=42)
clf.fit(X, y)

# Visualize one of the trees (e.g., the first tree)
plt.figure(figsize=(14, 9))
plot_tree(clf.estimators_[0], filled=True, feature_names=iris.feature_names, class_names=iris.target_names)
plt.show()
```




```
preds_data1
```

```
SepalLengthCm             5.2
SepalWidthCm              4.1
PetalLengthCm             1.5
PetalWidthCm              0.1
Species           Iris-setosa
Name: 32, dtype: object
```

```
# Making prediction on data 2 with Species =  Iris-virginica

pred_x3 = pd.DataFrame(np.array([6.4,2.8, 5.6, 2.2]).reshape(1,-1) ,
                    columns=['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'])
pred_y3 = clf.predict(pred_x3)
pred_class3 = Encoded_class[Encoded_class["Encoded"] == pred_y3[0]]["Species"].item()
print(f"Predicted class by model on preds_data1 : {pred_class3}")
```

```
Predicted class by model on preds_data1 : Iris-virginica
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fitte
  warnings.warn(
```

```
np.random.seed(42)
for i in range(10, 100, 10):
    print(f"Trying model with {i} estimators...")
    clf = RandomForestClassifier(n_estimators=i).fit(X_train, y_train)
    print(f"Model Accuracy on test set: {clf.score(X_test, y_test)*100:2f}%")
    print(" ")
```

```
Trying model with 10 estimators...
Model Accuracy on test set: 93.333333%

Trying model with 20 estimators...
Model Accuracy on test set: 93.333333%

Trying model with 30 estimators...
Model Accuracy on test set: 93.333333%

Trying model with 40 estimators...
Model Accuracy on test set: 93.333333%

Trying model with 50 estimators...
Model Accuracy on test set: 93.333333%

Trying model with 60 estimators...
Model Accuracy on test set: 93.333333%

Trying model with 70 estimators...
Model Accuracy on test set: 93.333333%

Trying model with 80 estimators...
Model Accuracy on test set: 93.333333%

Trying model with 90 estimators...
```

```
Model Accuracy on test set: 93.333333%
```

```
for i in range(0, 20):
  np.random.seed(i)
  X_train, X_test, y_train, y_test = train_test_split(X,
                                                      y,
                                                 test_size=0.2)
  clf = RandomForestClassifier()
  clf.fit(X_train, y_train)
  score = clf.score(X_test, y_test)
  print(f"Score with {i} seed number: {score}")
```

```
Score with 0 seed number: 0.9666666666666667
Score with 1 seed number: 0.9666666666666667
Score with 2 seed number: 0.9666666666666667
Score with 3 seed number: 0.9666666666666667
Score with 4 seed number: 0.9666666666666667
Score with 5 seed number: 0.9
Score with 6 seed number: 0.9333333333333333
Score with 7 seed number: 0.8666666666666667
Score with 8 seed number: 0.9
Score with 9 seed number: 1.0
Score with 10 seed number: 0.9666666666666667
Score with 11 seed number: 0.9333333333333333
Score with 12 seed number: 0.9666666666666667
Score with 13 seed number: 0.9666666666666667
Score with 14 seed number: 0.9666666666666667
Score with 15 seed number: 1.0
Score with 16 seed number: 0.8666666666666667
Score with 17 seed number: 0.9666666666666667
Score with 18 seed number: 1.0
Score with 19 seed number: 0.9666666666666667
```

```
import pickle

pickle.dump(clf, open("random_forest_model.pkl", "wb"))
```

```
loaded_model = pickle.load(open("random_forest_model.pkl", "rb"))
loaded_model.score(X_test, y_test)
```

```
0.9666666666666667
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

```
# Putting model into dictionary

models = {"Logistic Regression" : LogisticRegression(),
          "KNeighborsClassifier" : KNeighborsClassifier(),
          "Support Vector Classifier" : SVC(),
          "DecisionTree Classifier" : DecisionTreeClassifier(),
          "RandomForest Classifier" : RandomForestClassifier(),
          "Naive-Bayesian Classifier" : GaussianNB()}
models
```

```
{'Logistic Regression': LogisticRegression(),
 'KNeighborsClassifier': KNeighborsClassifier(),
 'Support Vector Classifier': SVC(),
 'DecisionTree Classifier': DecisionTreeClassifier(),
 'RandomForest Classifier': RandomForestClassifier(),
 'Naive-Bayesian Classifier': GaussianNB()}
```

```
from sklearn.model_selection import train_test_split

np.random.seed(65)
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2)
```

```
def fit_score_predict(models, X_train, X_test, y_train, y_test):
    #set random seed
    model_scores = {}
    model_prediction1 = {}
```

```
    model_prediction2 = {}
    model_prediction3 = {}
    for name , model in models.items():
        model.fit(X_train, y_train)
        model_scores[name] = model.score(X_test, y_test)

        # Making prediction on data 1 with Species = Iris-setosa

        pred_x1 = pd.DataFrame(np.array([5.2, 4.1, 1.5, 0.1]).reshape(1,-1) ,
                    columns=['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'])
        pred_y1 = model.predict(pred_x1)
        pred_class1 = Encoded_class[Encoded_class["Encoded"] == pred_y1[0]]["Species"].item()

        # Making prediction on data 2 with Species = Iris-versicolor

        pred_x2 = pd.DataFrame(np.array([6.8, 2.8,4.8, 1.4]).reshape(1,-1) ,
                    columns=['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'])
        pred_y2 = model.predict(pred_x2)
        pred_class2 = Encoded_class[Encoded_class["Encoded"] == pred_y2[0]]["Species"].item()

        # Making prediction on data 2 with Species =  Iris-virginica

        pred_x3 = pd.DataFrame(np.array([6.4,2.8, 5.6, 2.2]).reshape(1,-1) ,
                    columns=['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'])
        pred_y3 = clf.predict(pred_x3)
        pred_class3 = Encoded_class[Encoded_class["Encoded"] == pred_y3[0]]["Species"].item()


        model_prediction1[name] = pred_class1
        model_prediction2[name] = pred_class2
        model_prediction3[name] = pred_class3
    return {'score' : model_scores, 'predict' : [model_prediction1, model_prediction2, model_prediction3]}
```

```
model_scores_predict = fit_score_predict(models=models,
                             X_train=X_train,
                             X_test=X_test,
                             y_train=y_train,
                             y_test=y_test)
model_scores_predict
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but KNeighborsClassifier was fitte
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but KNeighborsClassifier was fitte
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SVC was fitted without feature
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SVC was fitted without feature
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but DecisionTreeClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but DecisionTreeClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fit
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but GaussianNB was fitted without
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but GaussianNB was fitted without
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but RandomForestClassifier was fit
  warnings.warn(
{'score': {'Logistic Regression': 0.9666666666666667,
```

```
 {'Logistic Regression': 'Iris-versicolor',
  'KNeighborsClassifier': 'Iris-versicolor',
  'Support Vector Classifier': 'Iris-versicolor',
  'DecisionTree Classifier': 'Iris-versicolor',
  'RandomForest Classifier': 'Iris-versicolor',
  'Naive-Bayesian Classifier': 'Iris-versicolor'},
 {'Logistic Regression': 'Iris-virginica',
  'KNeighborsClassifier': 'Iris-virginica',
  'Support Vector Classifier': 'Iris-virginica',
  'DecisionTree Classifier': 'Iris-virginica',
  'RandomForest Classifier': 'Iris-virginica',
  'Naive-Bayesian Classifier': 'Iris-virginica'}]}
```

```
model_scores = model_scores_predict["score"]
model_scores
```

```
{'Logistic Regression': 0.9666666666666667,
 'KNeighborsClassifier': 0.9333333333333333,
 'Support Vector Classifier': 0.9666666666666667,
 'DecisionTree Classifier': 0.9333333333333333,
 'RandomForest Classifier': 0.9333333333333333,
 'Naive-Bayesian Classifier': 0.9333333333333333}
```

```
score_list = []
model_list = []
for name, score in model_scores.items():
    score_list.append(score)
    model_list.append(name)
score_list
```

```
[0.9666666666666667,
 0.9333333333333333,
 0.9666666666666667,
 0.9333333333333333,
 0.9333333333333333,
 0.9333333333333333]
```

```
predict_list1 = []
for name, value in model_scores_predict["predict"][0].items():
    predict_list1.append(value)
predict_list1

predict_list2 = []
for name, value in  model_scores_predict["predict"][1].items():
    predict_list2.append(value)
predict_list2

predict_list3 = []
for name, value in  model_scores_predict["predict"][2].items():
    predict_list3.append(value)
predict_list3
```

```
['Iris-virginica',
 'Iris-virginica',
 'Iris-virginica',
 'Iris-virginica',
 'Iris-virginica',
 'Iris-virginica']
```

```
model_score_df = pd.DataFrame({'Model': model_list,
                               'Score': score_list,
                               'predict on sample1': predict_list1,
                               'predict on sample2': predict_list2,
                               'predict on sample3': predict_list3})
model_score_df
```

	Model	Score	predict on sample1	predict on sample2	predict on sample3
0	Logistic Regression	0.966667	Iris-setosa	Iris-versicolor	Iris-virginica
1	KNeighborsClassifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
2	Support Vector Classifier	0.966667	Iris-setosa	Iris-versicolor	Iris-virginica
3	DecisionTree Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
4	RandomForest Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
5	Naive-Bayesian Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica

```
result_df = model_score_df.sort_values(by='Score', ascending=False)
result_df
```

	Model	Score	predict on sample1	predict on sample2	predict on sample3
0	Logistic Regression	0.966667	Iris-setosa	Iris-versicolor	Iris-virginica
2	Support Vector Classifier	0.966667	Iris-setosa	Iris-versicolor	Iris-virginica
1	KNeighborsClassifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
3	DecisionTree Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
4	RandomForest Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
5	Naive-Bayesian Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica

```
result = result_df.set_index('Model')
result
```

	Score	predict on sample1	predict on sample2	predict on sample3
Model				
Logistic Regression	0.966667	Iris-setosa	Iris-versicolor	Iris-virginica
Support Vector Classifier	0.966667	Iris-setosa	Iris-versicolor	Iris-virginica
KNeighborsClassifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
DecisionTree Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
RandomForest Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica
Naive-Bayesian Classifier	0.933333	Iris-setosa	Iris-versicolor	Iris-virginica