Importing Libraries

In [1]:

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

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [3]:

```
df = pd.read_csv('/content/drive/My Drive/iris_model/Iris.csv', index_col=False)
```

In [4]:

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

In [5]:

```
df.head()
```

Out[5]:

	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

In [6]:

```
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
```

In [7]:

```
df.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
SepalLengthCm	150.0	5.843333	0.828066	4.3	5.1	5.80	6.4	7.9
SepalWidthCm	150.0	3.054000	0.433594	2.0	2.8	3.00	3.3	4.4
PetalLengthCm	150.0	3.758667	1.764420	1.0	1.6	4.35	5.1	6.9
PetalWidthCm	150.0	1.198667	0.763161	0.1	0.3	1.30	1.8	2.5

In [8]:

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

Out[8]:

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

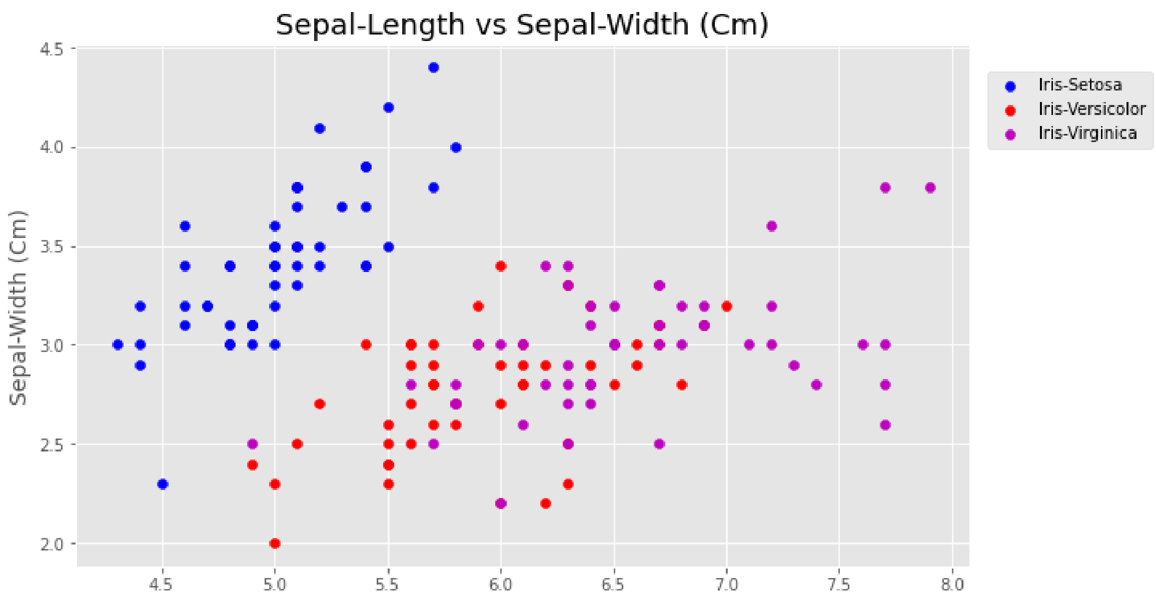
In [9]:

```
setosa = df[df.Species == 'Iris-setosa']
versicolor = df[df.Species == 'Iris-versicolor']
virginica = df[df.Species == 'Iris-virginica']
```

Visual EDA

In [10]:

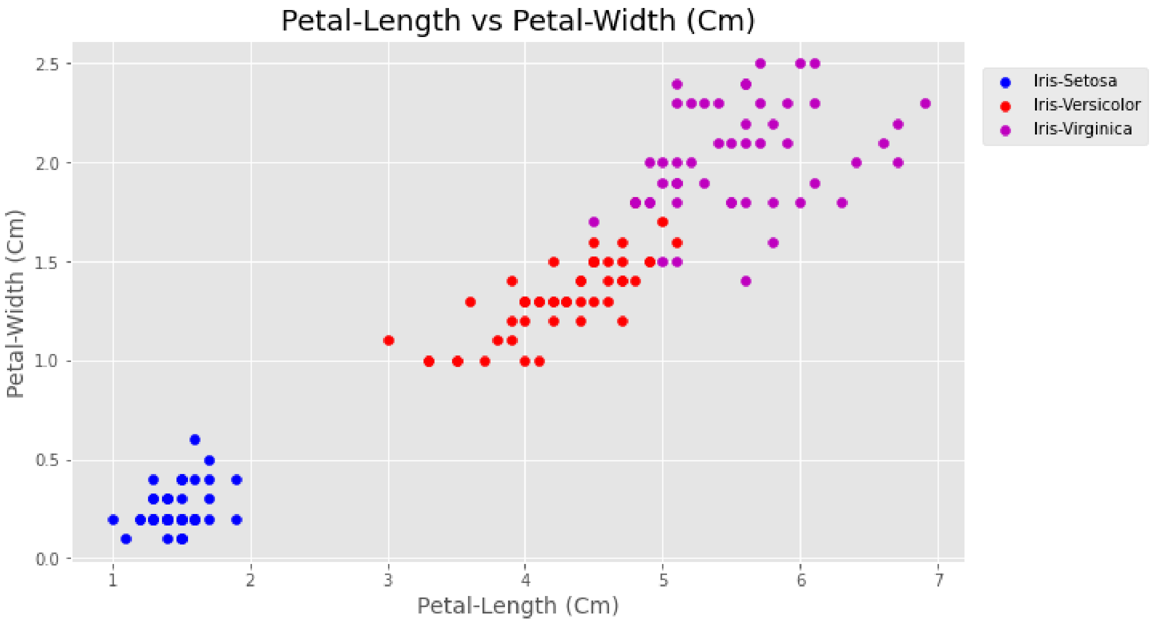
```
plt.figure(figsize=(10,6))
plt.style.use('ggplot')
plt.scatter(setosa['SepalLengthCm'], setosa['SepalWidthCm'], c = 'b', label = 'Iris-Setosa')
plt.scatter(versicolor['SepalLengthCm'], versicolor['SepalWidthCm'], c = 'r', label = 'Iris-Versico
lor')
plt.scatter(virginica['SepalLengthCm'], virginica['SepalWidthCm'], c = 'm', label = 'Iris-Virginica
')
plt.xlabel('Sepal-Length (Cm)', fontsize = 14)
plt.ylabel('Sepal-Width (Cm)', fontsize = 14)
plt.title('Sepal-Length vs Sepal-Width (Cm)', fontsize = 18)
plt.legend(loc = (1.02, 0.8))
plt.show()
```



Sepal-Length (Cm)

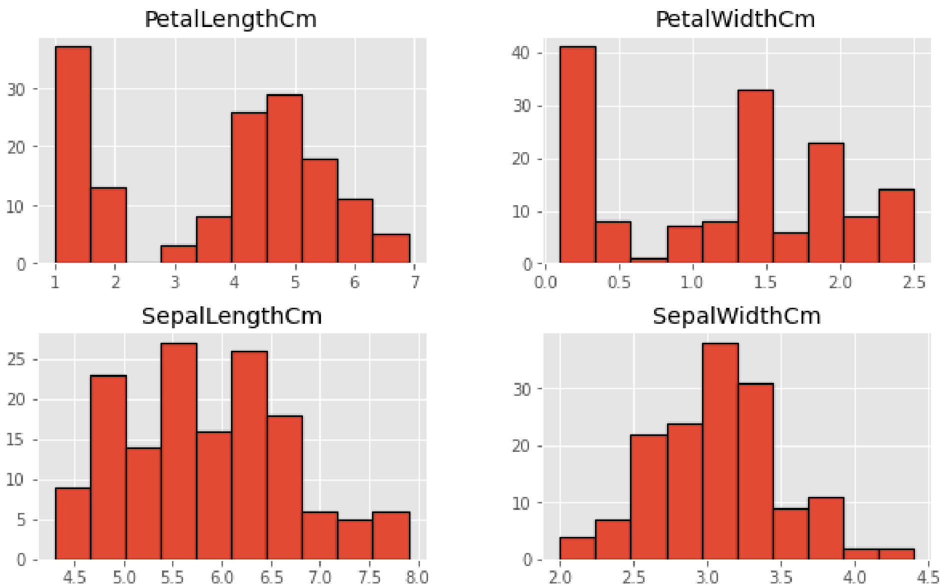
In [11]:

```
plt.figure(figsize=(10,6))
plt.style.use('ggplot')
plt.scatter(setosa['PetalLengthCm'], setosa['PetalWidthCm'], c = 'b', label = 'Iris-Setosa')
plt.scatter(versicolor['PetalLengthCm'], versicolor['PetalWidthCm'], c = 'r', label = 'Iris-Versico
lor')
plt.scatter(virginica['PetalLengthCm'], virginica['PetalWidthCm'], c = 'm', label = 'Iris-Virginica
')
plt.xlabel('Petal-Length (Cm)', fontsize = 14)
plt.ylabel('Petal-Width (Cm)', fontsize = 14)
plt.title('Petal-Length vs Petal-Width (Cm)', fontsize = 18)
plt.legend(loc = (1.02, 0.8))
plt.show()
```



In [12]:

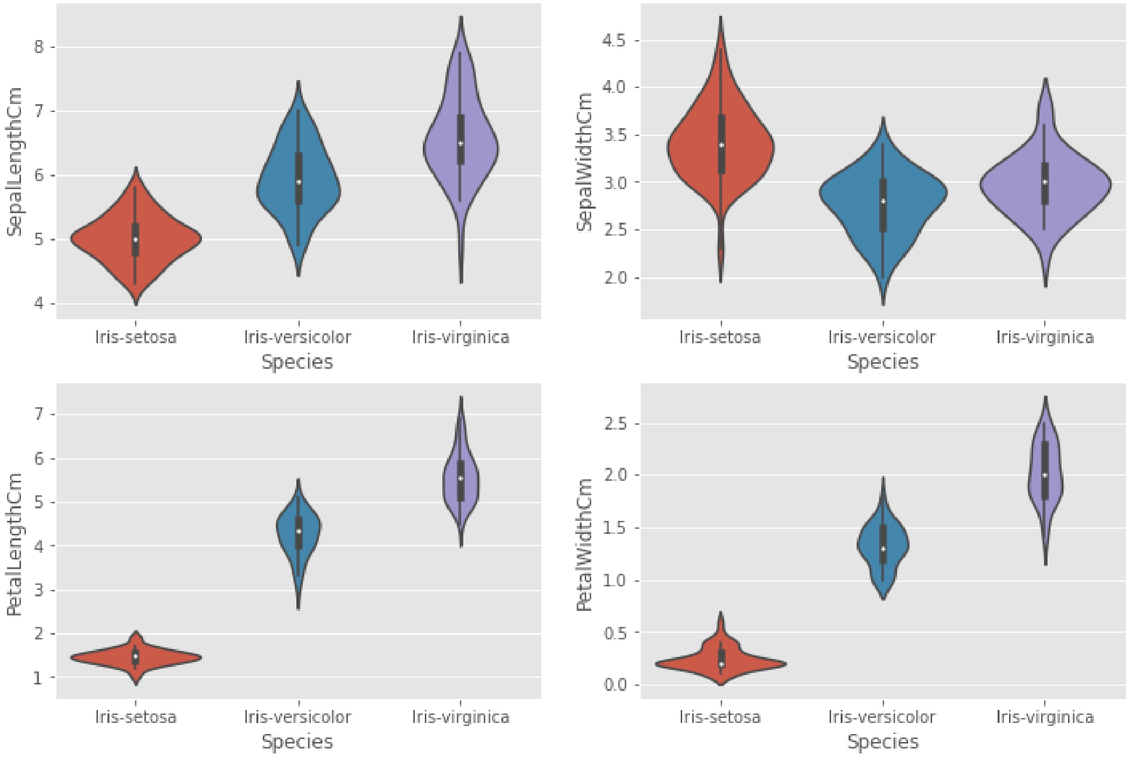
```
df.hist(edgecolor = 'black', linewidth = 1.2)
plt.gcf().set_size_inches(10, 6)
plt.show()
```



In [13]:

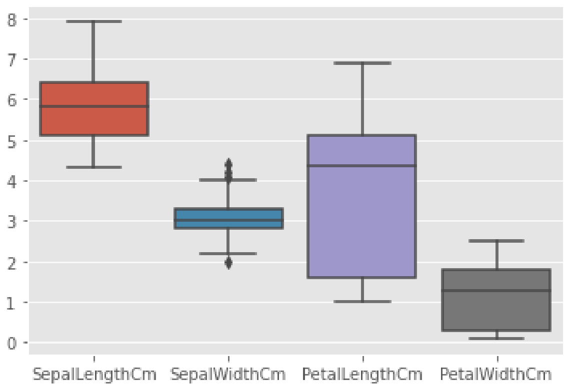
```
plt.figure(figsize=(12,8))
```

```
plt.figure(figsize=(12,8))
plt.subplot(2,2,1)
sns.violinplot(x='Species', y='SepalLengthCm', data=df)
plt.subplot(2,2,2)
sns.violinplot(x='Species', y='SepalWidthCm', data=df)
plt.subplot(2,2,3)
sns.violinplot(x='Species', y='PetalLengthCm', data=df)
plt.subplot(2,2,4)
sns.violinplot(x='Species', y='PetalWidthCm', data=df)
plt.show()
```



In [14]:

```
sns.boxplot(data=df)
plt.show()
```



In [15]:

```
df.shape
```

Out[15]:

```
(150, 5)
```

Building SkLearn Model

In [16]:

```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.metrics import mean_squared_error, accuracy_score, classification_report,
confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

Train Test Split

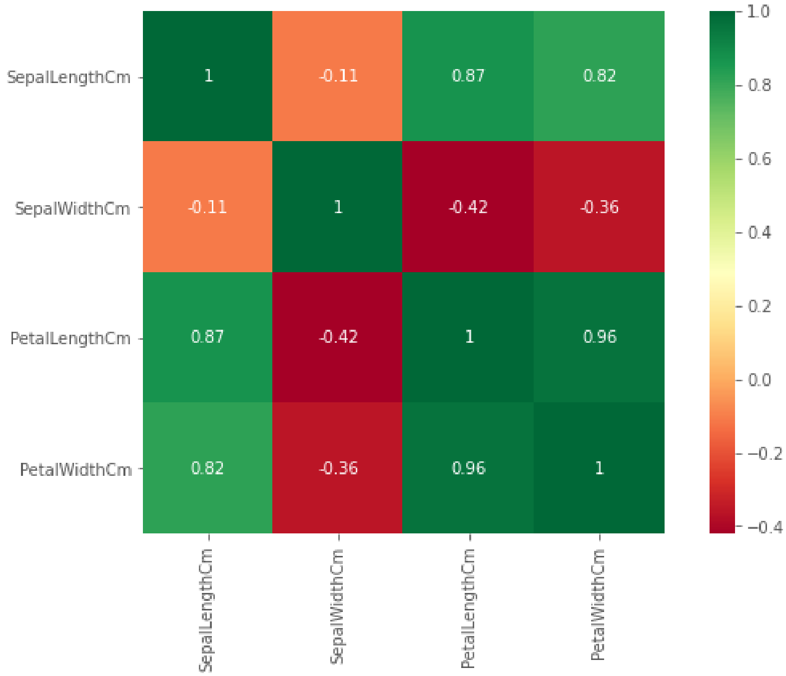
In [17]:

```
X = df.drop('Species', axis = 1)
y = df['Species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42, strat
ify = y)
```

Feature Selection

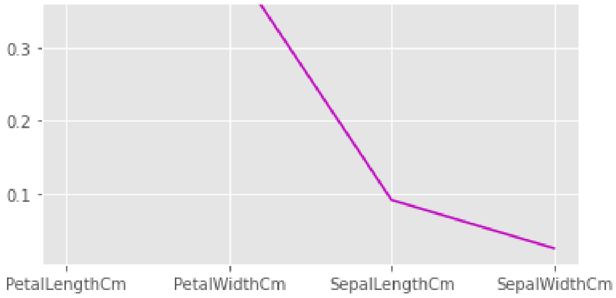
In [18]:

```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot = True, cmap = 'RdYlGn', square=True)
plt.yticks(rotation = 0)
plt.xticks(rotation = 90)
plt.show()
```



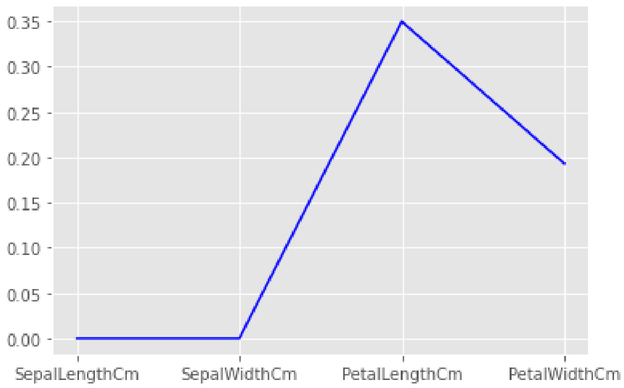
In [19]:

```
model = RandomForestClassifier(n_estimators=100, random_state=0)
model.fit(X, y)
imp_features = pd.Series(model.feature_importances_, index = X.columns).sort_values(ascending=False
)
# print(imp_features)
plt.plot(imp_features, color = 'm')
plt.show()
```



In [20]:

```
from sklearn.preprocessing import LabelEncoder
g = y.copy()
le = LabelEncoder().fit(g)
# print(le.classes_)
encoded_y = le.transform(g)
from sklearn.linear_model import LassoCV
names = X.columns
lasso = LassoCV(cv=3)
features = lasso.fit(X, encoded_y).coef_
plt.plot(features, color = 'blue')
plt.xticks(range(len(names)), names, rotation = 0)
plt.figure(figsize=(7,4))
plt.show()
```




```
<Figure size 504x288 with 0 Axes>
```

Checking Accuracy of Each Model

In [21]:

```
accuracy = []
error = []
classifiers = ['Linear_SVC', 'Radial_SVC', 'KNN', 'Logistic_Regression', 'Decision_Tree', 'Random_F
orest']
models = [SVC(gamma='scale', kernel='linear'), SVC(gamma='auto', kernel='rbf'),
KNeighborsClassifier(n_neighbors=5),
         LogisticRegression(solver='liblinear', multi_class='auto'), DecisionTreeClassifier(), Rand
omForestClassifier(n_estimators=100)]
for i in models:
    model = i
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy.append(accuracy_score(y_pred, y_test))
    y_pred_encoded = LabelEncoder().fit_transform(y_pred)
    y_test_encoded = LabelEncoder().fit_transform(y_test)
    error.append(np.sqrt(mean_squared_error(y_test_encoded, y_pred_encoded)))
d = {'Accuracy' : accuracy, 'RMSE' : error}
score = pd.DataFrame(d, index = classifiers)
score
```

Out[21]:

	Accuracy	RMSE
Linear_SVC	1.000000	0.000000
Radial_SVC	0.977778	0.149071
KNN	0.977778	0.149071
Logistic_Regression	0.911111	0.298142
Decision_Tree	0.977778	0.149071
Random_Forest	0.888889	0.333333

Hyper-Parameter Tuning (KNN)

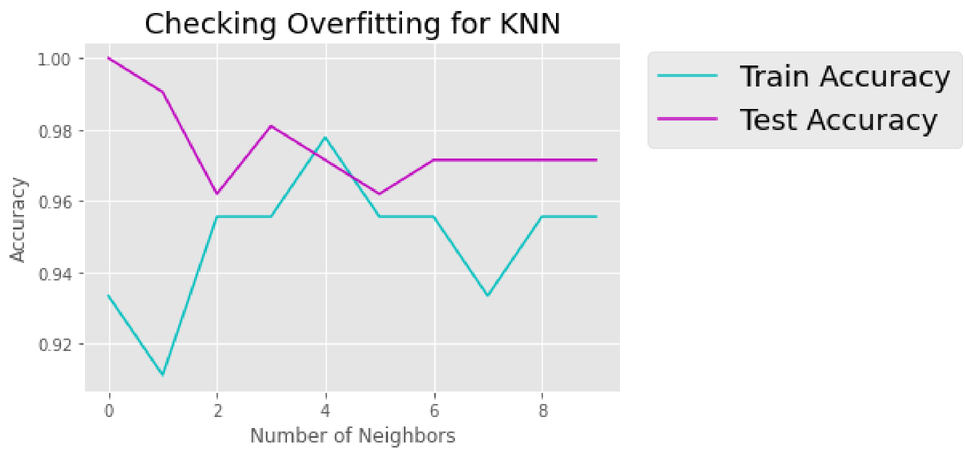
In [22]:

```
param_grid = {'n_neighbors' : np.arange(1, 51)}
knn_model = KNeighborsClassifier()
knn_cv = GridSearchCV(knn_model, param_grid, cv = 5, scoring='accuracy')
knn_cv.fit(X, y)
print(knn_cv.best_params_)
print(knn_cv.best_score_)
```

```
{'n_neighbors': 6}
0.9800000000000001
```

In [23]:

```
neighbors = np.arange(1, 11)
test_accuracy = np.empty(len(neighbors))
train_accuracy = np.empty(len(neighbors))
for i, k in enumerate(neighbors):
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    test_accuracy[i] = model.score(X_train, y_train)
    train_accuracy[i] = model.score(X_test, y_test)
plt.plot(train_accuracy, label='Train Accuracy', color = 'c')
plt.plot(test_accuracy, label='Test Accuracy', color = 'm')
plt.title('Checking Overfitting for KNN', fontsize = 18)
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.legend(fontsize=18, loc = (1.05, 0.7))
plt.figure(figsize=(12,6))
plt.show()
```




```
<Figure size 864x432 with 0 Axes>
```

Choosing Linear_SVC

In [24]:

```
model = SVC(gamma='auto', kernel='linear')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f'Root Mean Square Error: {mean_squared_error(y_test_encoded, y_pred_encoded)}')
print(f'Model Accuracy: {round(accuracy_score(y_test, y_pred), 3) * 100}%')
print('=====================================================')
print(f'Here is the "Classification Report"')
print(classification_report(y_test, y_pred))
print(f'\nHere is the "Confusing Matrix"\n')
print(confusion_matrix(y_test, y_pred))
print('=====================================================')
```

```
Root Mean Square Error: 0.1111111111111111
Model Accuracy: 100.0%
=====================================================
Here is the "Classification Report"
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        15
Iris-versicolor       1.00      1.00      1.00        15
 Iris-virginica       1.00      1.00      1.00        15

       accuracy                           1.00        45
      macro avg       1.00      1.00      1.00        45
   weighted avg       1.00      1.00      1.00        45


Here is the "Confusing Matrix"

[[15  0  0]
 [ 0 15  0]
 [ 0  0 15]]
=====================================================
```

Creating pipeline

In [25]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
steps = [('model', SVC(kernel='linear', gamma = 'auto'))]
pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
print('{}%'.format(accuracy_score(y_test, y_pred)*100))
```

```
100.0%
```

Saving and Loading the Model

In [26]:

```
import joblib
#save the model
joblib.dump(pipeline, 'iris_model.pkl')
new_model = joblib.load('iris_model.pkl')
new_model.predict(X_test)
print('{}%'.format(accuracy_score(y_test, y_pred)*100))
```

```
100.0%
```

Building TensorFlow/Keras Model

In [27]:

```
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

df = pd.read_csv('/content/drive/My Drive/iris_model/Iris.csv', index_col=False)
X = df.drop(['Species', 'Id'], axis = 1).values
y = df['Species'].values
y = LabelEncoder().fit_transform(y)
y = to_categorical(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

In [28]:

```
# Build the model
model = Sequential()

model.add(Dense(25, input_shape=(4,), activation='relu'))
model.add(Dense(25, activation='relu'))
model.add(Dense(3, activation='softmax'))

# Adam optimizer with learning rate of 0.001
optimizer = Adam(lr=0.001)
model.compile(optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

print('Neural Network Model Summary: ')
print(model.summary())
```

```
Neural Network Model Summary:
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense (Dense)                (None, 25)                125       
_________________________________________________________________
dense_1 (Dense)              (None, 25)                650       
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 78        
=================================================================
Total params: 853
Trainable params: 853
Non-trainable params: 0
_________________________________________________________________
None
```

In [29]:

```
model.fit(X_train, y_train, verbose=0, batch_size=5, epochs=200)
```

Out[29]:

```
<tensorflow.python.keras.callbacks.History at 0x7f44d198f668>
```

In [30]:

```
results = model.evaluate(X_test, y_test)

print('Test Loss: {:.2f}'.format(results[0]))
print('Test Accuracy: {:.2f}'.format(results[1]))
```

```
1/1 [==============================] - 0s 2ms/step - loss: 0.0115 - accuracy: 1.0000
Test Loss: 0.01
Test Accuracy: 1.00
```

In [31]:

```
model.save('keras_model.h5')
```

In [32]:

```
from tensorflow.keras.models import load_model
```

In [33]:

```
new_model = load_model('keras_model.h5')
```

In [34]:

```
res = new_model.evaluate(X_test, y_test)
print('Test Loss: {:.2f}'.format(res[0]))
print('Test Accuracy: {:.2f}'.format(res[1]))
```

```
1/1 [==============================] - 0s 1ms/step - loss: 0.0115 - accuracy: 1.0000
Test Loss: 0.01
Test Accuracy: 1.00
```

In [34]:

```

```