Species Prediction From Iris Data

Importing libraries

In [1]:
```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

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

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor


%matplotlib inline
```

Reading Exploring Dataset

In [2]:
```
data = pd.read_csv('clean.csv')
```

In [3]:
```
# shape
print(data.shape)
```

```
(150, 6)
```

In [4]:
```
data.dtypes
```

Out[4]:
```
Unnamed: 0        int64
Species          object
Sepal_Length    float64
Petal_Length    float64
Sepal_Width     float64
Petal_Width     float64
dtype: object
```

In [5]:
```
# descriptions
print(data.describe())
```

```
       Unnamed: 0  Sepal_Length  Petal_Length  Sepal_Width  Petal_Width
count  150.000000    150.000000    150.000000   150.000000   150.000000
mean    74.500000      5.843333      3.758667     3.054000     1.198667
std     43.445368      0.828066      1.764420     0.433594     0.763161
min      0.000000      4.300000      1.000000     2.000000     0.100000
25%     37.250000      5.100000      1.600000     2.800000     0.300000
50%     74.500000      5.800000      4.350000     3.000000     1.300000
75%    111.750000      6.400000      5.100000     3.300000     1.800000
max    149.000000      7.900000      6.900000     4.400000     2.500000
```

In [6]:
```
# Checking for missing values
data.isna().sum()
```

Out[6]:
```
Unnamed: 0      0
Species         0
Sepal_Length    0
Petal_Length    0
Sepal_Width     0
Petal_Width     0
dtype: int64
```

In [7]:
```
# Checking for missing values another method
pd.isnull(data).any()
```

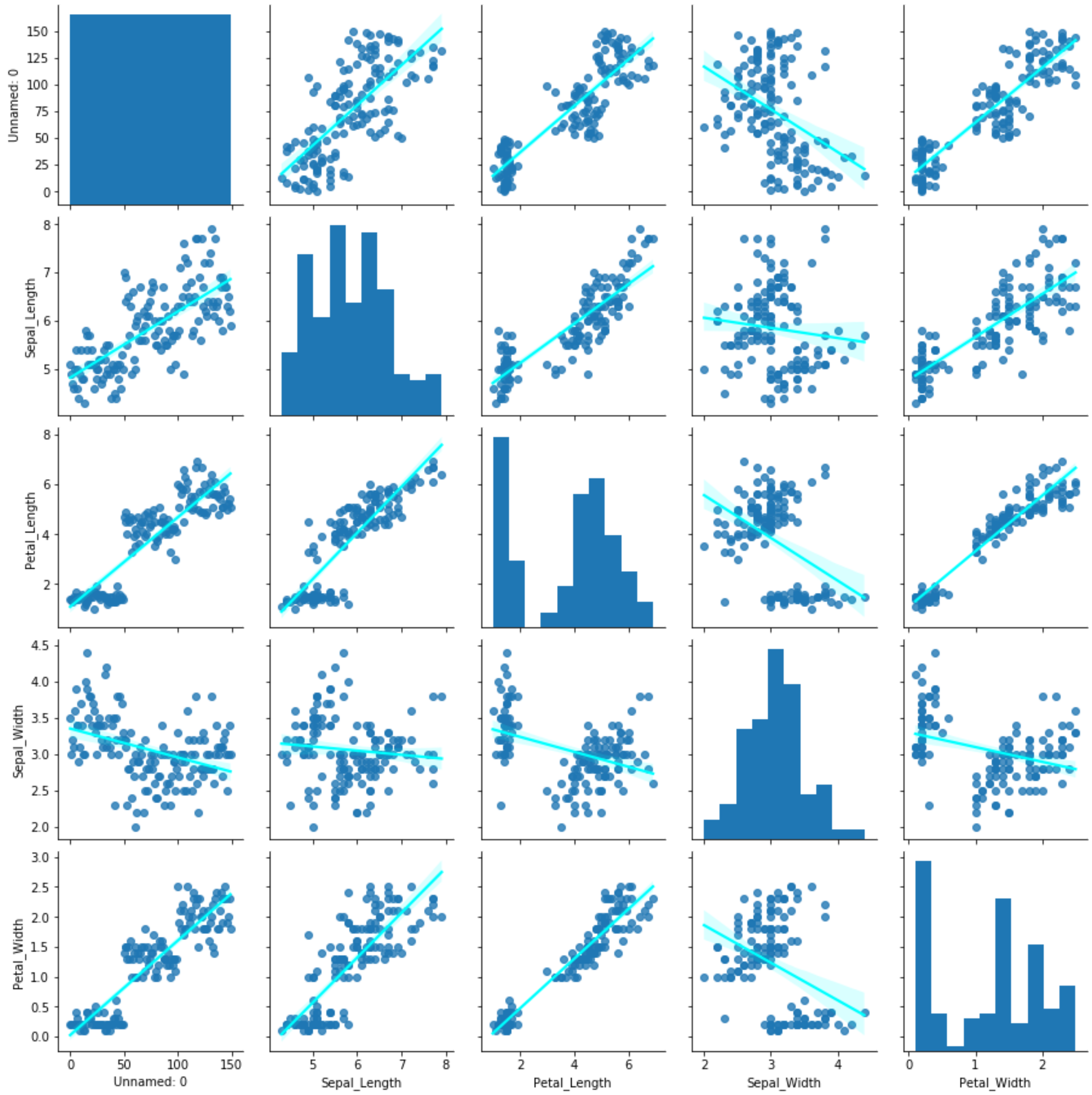
Out[7]:
```
Unnamed: 0      False
Species         False
Sepal_Length    False
Petal_Length    False
Sepal_Width     False
Petal_Width     False
dtype: bool
```

In [8]:
```
#counting species
data['Species'].value_counts()
```

Out[8]:
```
Iris-versicolor    50
Iris-setosa        50
Iris-virginica     50
Name: Species, dtype: int64
```

Initial visualization

In [9]:
```
%%time
sns.pairplot(data, kind='reg', plot_kws={'line_kws':{'color': 'cyan'}})
plt.show()
```




```
Wall time: 8.13 s
```

In [10]:
```
data.columns
```

Out[10]:
```
Index(['Unnamed: 0', 'Species', 'Sepal_Length', 'Petal_Length', 'Sepal_Widt
h',
       'Petal_Width'],
      dtype='object')
```

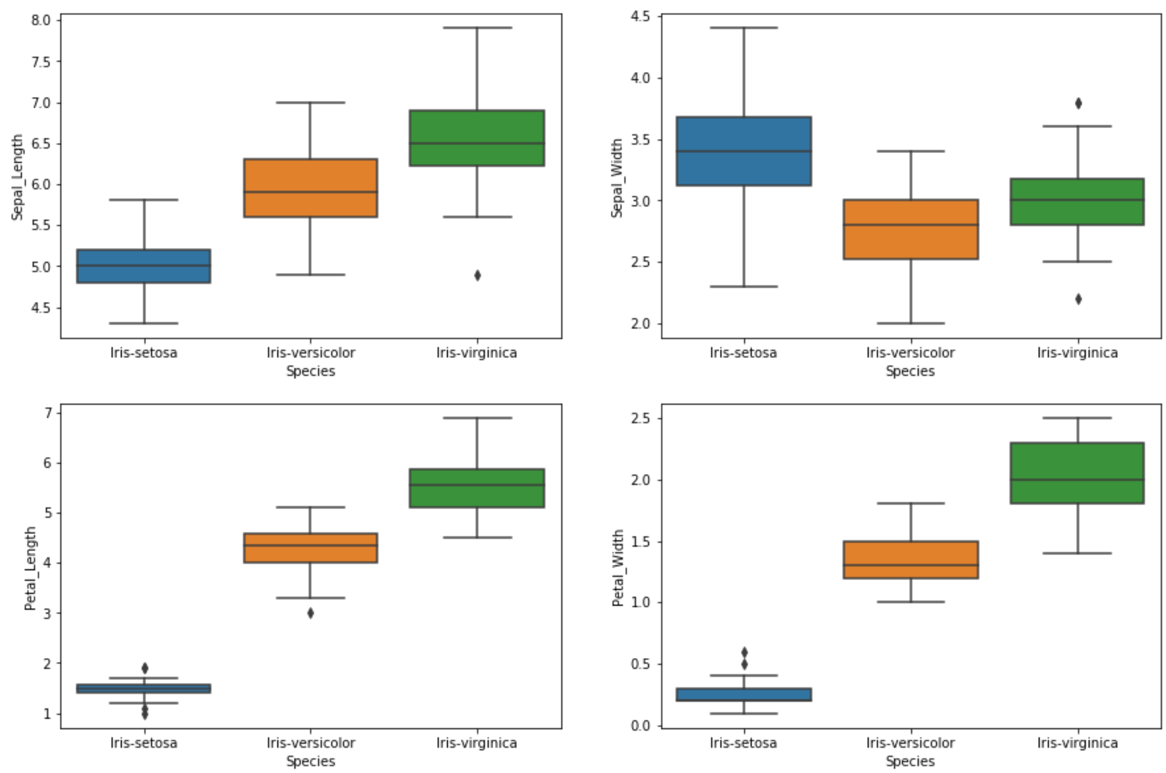
In [13]:
```
#Removing unnamed columns
data = data.loc[:, ~data.columns.str.contains('^Unnamed')]
```

In [14]:
```
data.columns
```

Out[14]:
```
Index(['Species', 'Sepal_Length', 'Petal_Length', 'Sepal_Width',
       'Petal_Width'],
      dtype='object')
```

In [16]:
```
#Boxplot
import seaborn as sns
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.boxplot(x='Species',y='Sepal_Length',data=data)
plt.subplot(2,2,2)
sns.boxplot(x='Species',y='Sepal_Width',data=data)
plt.subplot(2,2,3)
sns.boxplot(x='Species',y='Petal_Length',data=data)
plt.subplot(2,2,4)
sns.boxplot(x='Species',y='Petal_Width',data=data)
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x248d081e088>
```

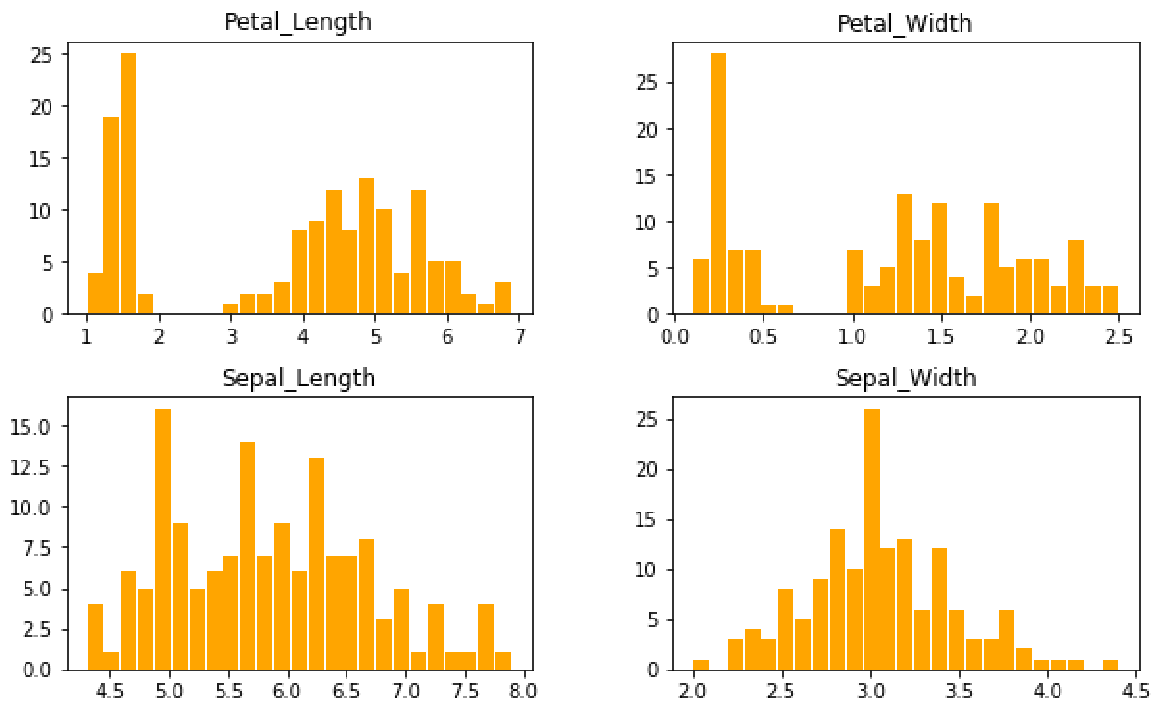


In [17]:

```
# histograms
data.hist(bins=25, grid=False, figsize=(10,6), color='orange', zorder=2, rwic
```

Out[17]:

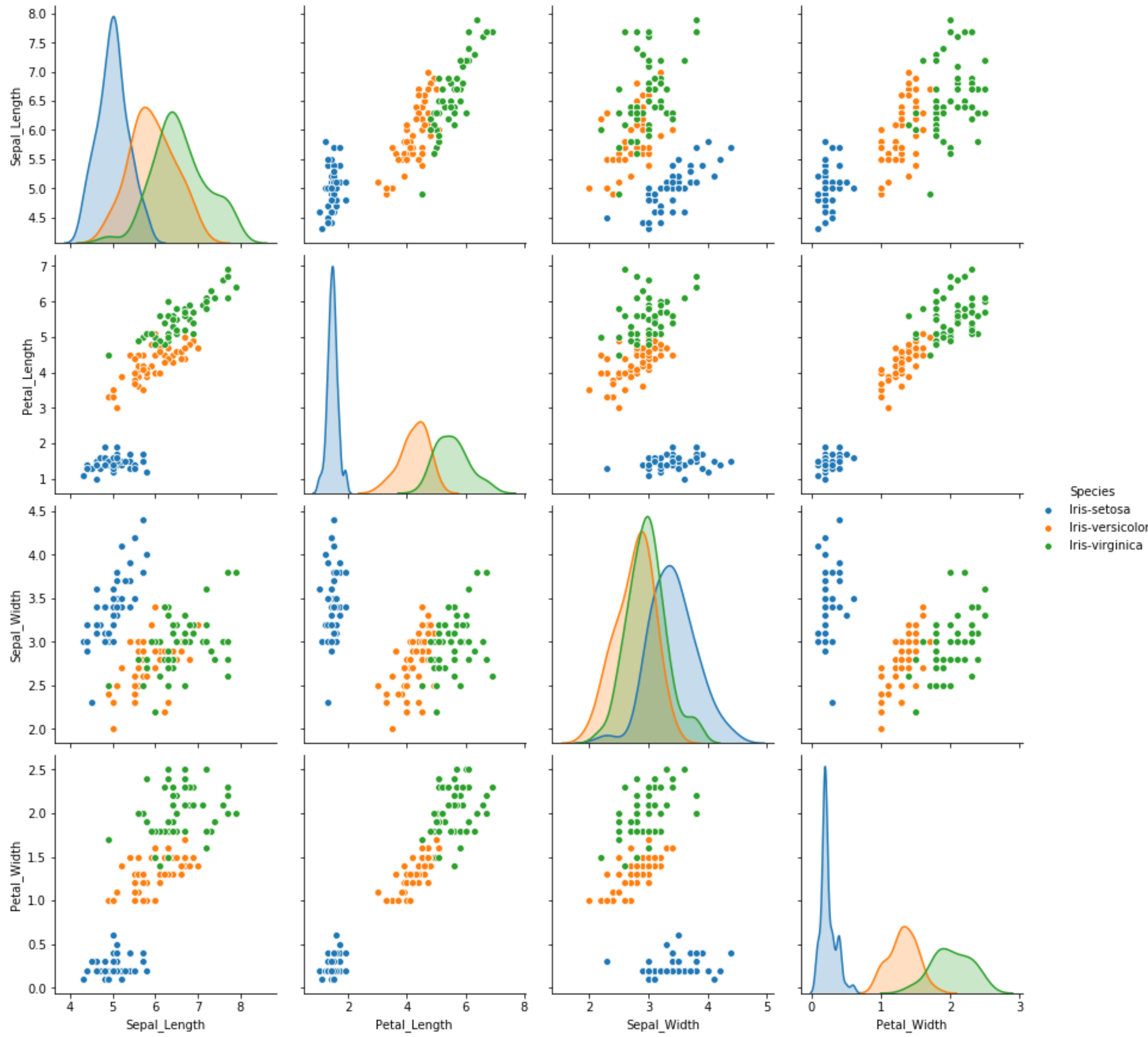
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000248D0893A48
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000248D09B5B88
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x00000248D09F1408
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x00000248D0A27C08
>]],
      dtype=object)
```



In [18]:
```
#Pairwise joint plot (scatter matrix)
sns.pairplot(data, hue='Species', size=3, diag_kind="kde")
```

```
C:\Users\mayam\anaconda3\lib\site-packages\seaborn\axisgrid.py:2079: UserWa
rning: The `size` parameter has been renamed to `height`; please update you
r code.
  warnings.warn(msg, UserWarning)
```

Out[18]: <seaborn.axisgrid.PairGrid at 0x248d0c62a88>

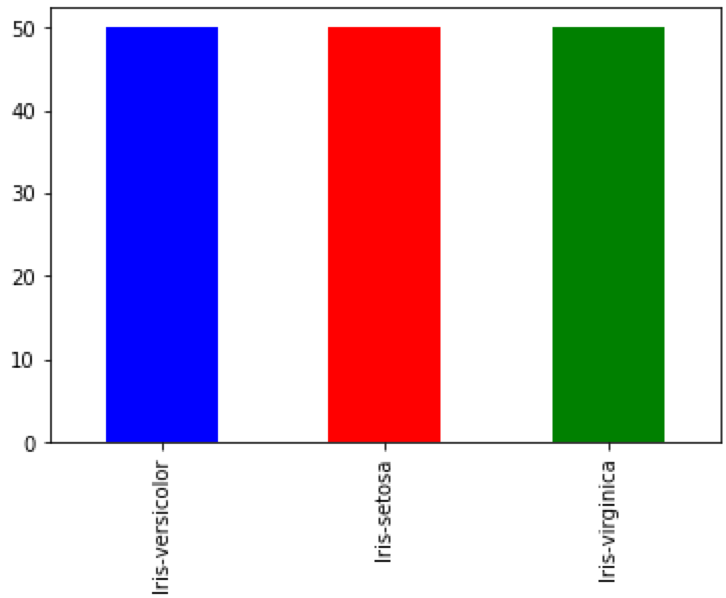


In [19]:

```
data.Species.value_counts().plot(kind="bar", color=["blue", "red","green"])
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x248cefa7848>
```



In [20]:

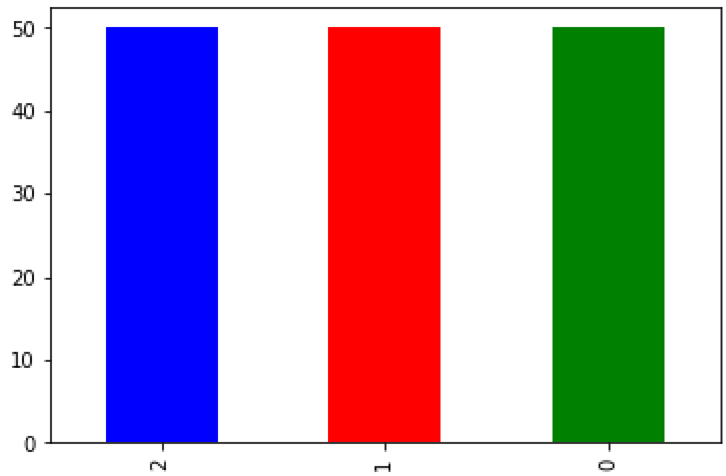
```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Species'.
data['Species']= label_encoder.fit_transform(data['Species'])

data['Species'].unique()
```

Out[20]:

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

In [21]:
```
# To find the species code for label in visualization
data.Species.value_counts().plot(kind="bar", color=["blue", "red","green"])
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x248cf160448>



In [22]:
```
categorical_features = []
continous_features = []
for column in data.columns:
    print('==============================')
    print(f"{column} : {data[column].unique()}")
    if len(data[column].unique()) <= 10:
        categorical_features.append(column)
    else:
        continous_features.append(column)
```

```
==============================
Species : [0 1 2]
==============================
Sepal_Length : [5.1 4.9 4.7 4.6 5.  5.4 4.4 4.8 4.3 5.8 5.7 5.2 5.5 4.5 5.3
7.  6.4 6.9
 6.5 6.3 6.6 5.9 6.  6.1 5.6 6.7 6.2 6.8 7.1 7.6 7.3 7.2 7.7 7.4 7.9]
==============================
Petal_Length : [1.4 1.3 1.5 1.7 1.6 1.1 1.2 1.  1.9 4.7 4.5 4.9 4.  4.6 3.3
3.9 3.5 4.2
 3.6 4.4 4.1 4.8 4.3 5.  3.8 3.7 5.1 3.  6.  5.9 5.6 5.8 6.6 6.3 6.1 5.3
 5.5 6.7 6.9 5.7 6.4 5.4 5.2]
==============================
Sepal_Width : [3.5 3.  3.2 3.1 3.6 3.9 3.4 2.9 3.7 4.  4.4 3.8 3.3 4.1 4.2
2.3 2.8 2.4
 2.7 2.  2.2 2.5 2.6]
==============================
Petal_Width : [0.2 0.4 0.3 0.1 0.5 0.6 1.4 1.5 1.3 1.6 1.  1.1 1.8 1.2 1.7
2.5 1.9 2.1
 2.2 2.  2.4 2.3]
```

In [23]:
```
categorical_features
```

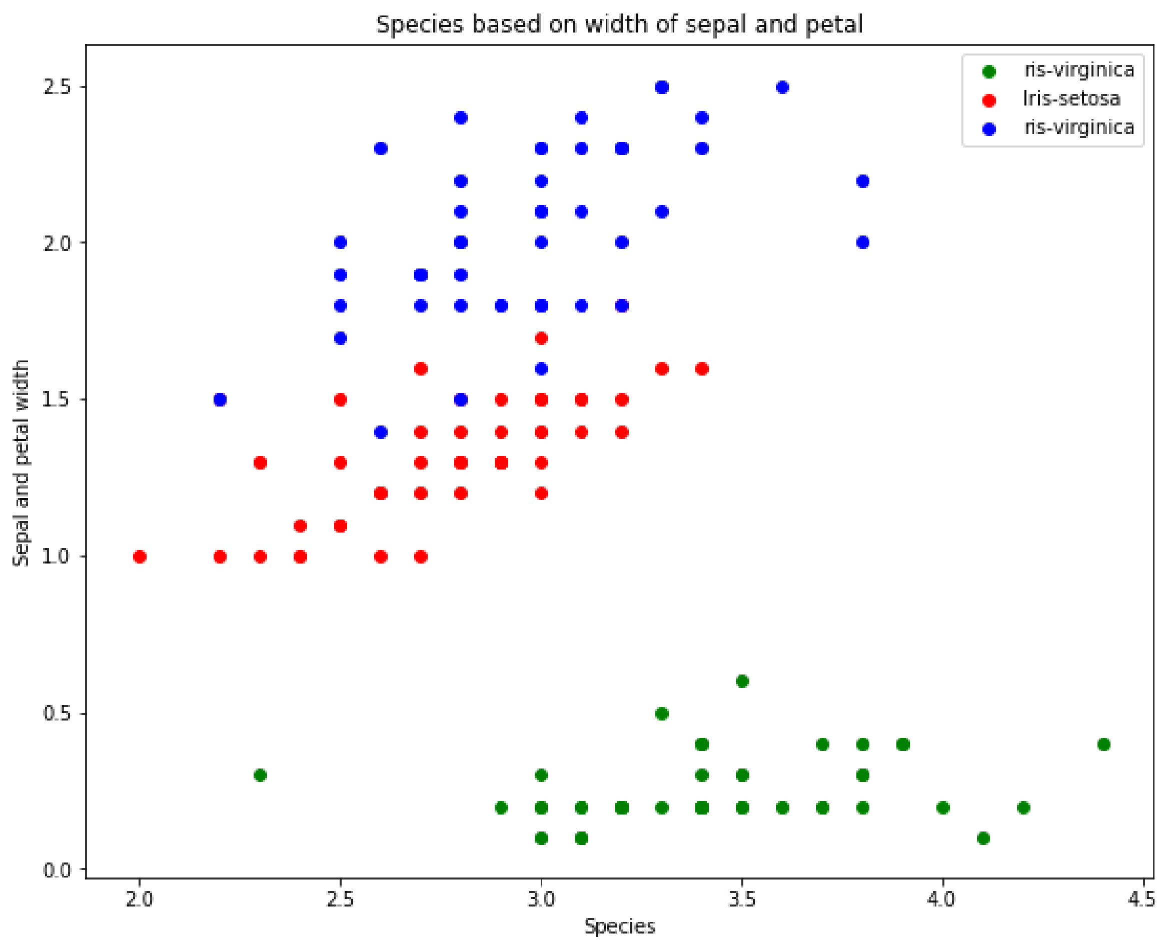
Out[23]: ['Species']

In [24]:
```
continous_features
```

Out[24]:
```
['Sepal_Length', 'Petal_Length', 'Sepal_Width', 'Petal_Width']
```

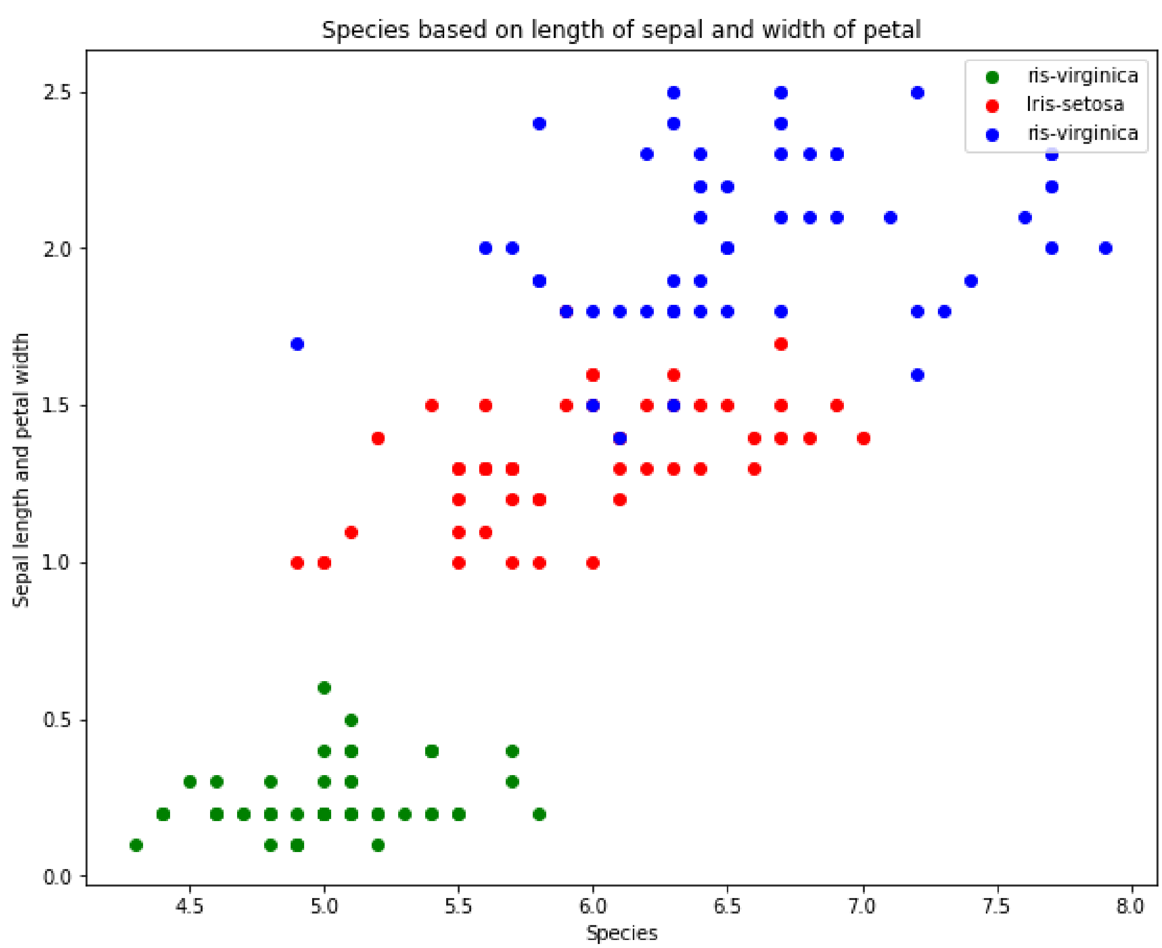
In [25]:

```
# Create another figure
plt.figure(figsize=(10, 8))

# Scatter with ris-virginica
plt.scatter(data.Sepal_Width[data.Species==0],
            data.Petal_Width[data.Species==0],
            c="green")

# Scatter with Iris-setosa
plt.scatter(data.Sepal_Width[data.Species==1],
            data.Petal_Width[data.Species==1],
            c="red")

# Scatter with ris-virginica
plt.scatter(data.Sepal_Width[data.Species==2],
            data.Petal_Width[data.Species==2],
            c="blue")

# Add some helpful info
plt.title("Species based on width of sepal and petal")
plt.xlabel("Species")
plt.ylabel("Sepal and petal width")
plt.legend(["ris-virginica", "Iris-setosa","ris-virginica"]);
```



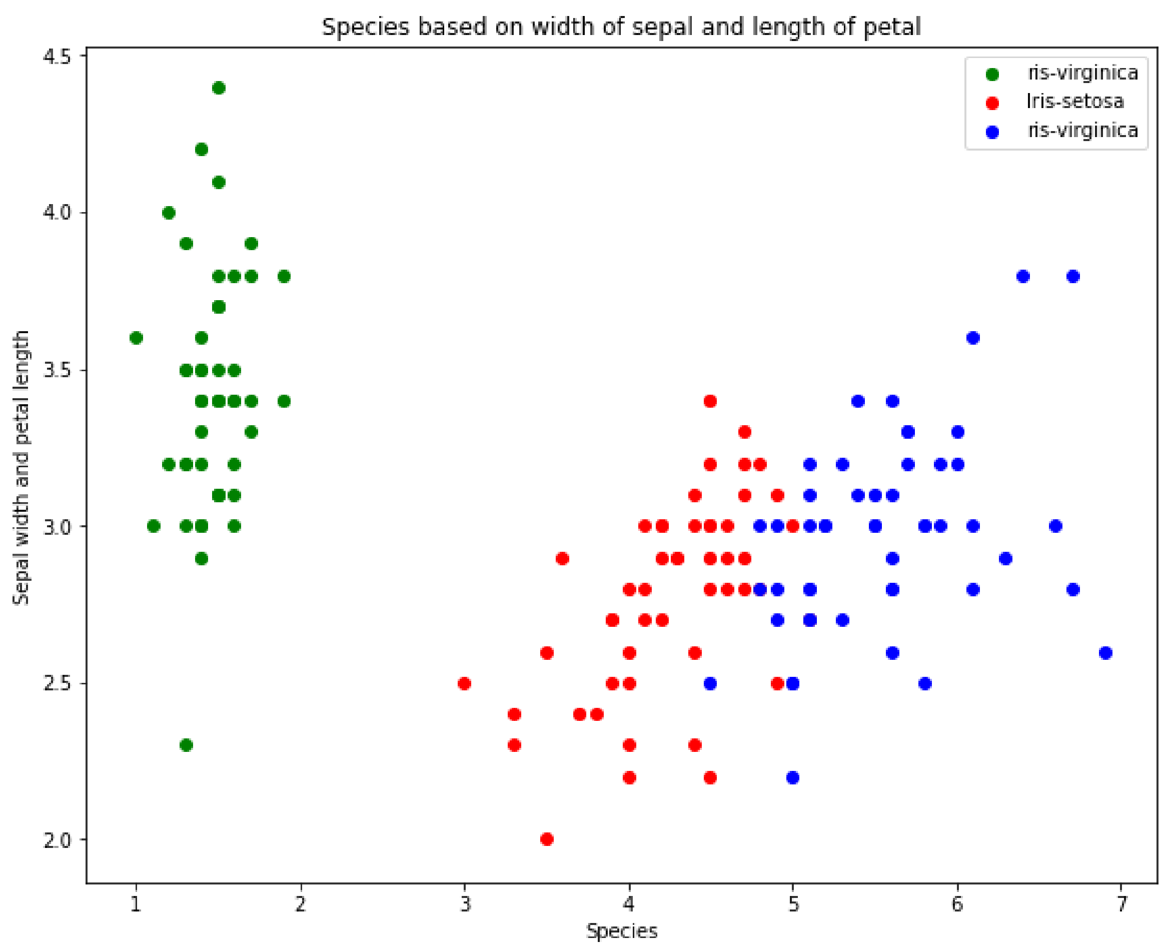
In [26]:

```python
# Create another figure PetalLengthCm Sepal_Length
plt.figure(figsize=(10, 8))

# Scatter with ris-virginica
plt.scatter(data.Sepal_Length[data.Species==0],
            data.Petal_Width[data.Species==0],
            c="green")

# Scatter with Iris-setosa
plt.scatter(data.Sepal_Length[data.Species==1],
            data.Petal_Width[data.Species==1],
            c="red")

# Scatter with ris-virginica
plt.scatter(data.Sepal_Length[data.Species==2],
            data.Petal_Width[data.Species==2],
            c="blue")

# Add some helpful info
plt.title("Species based on length of sepal and width of petal")
plt.xlabel("Species")
plt.ylabel("Sepal length and petal width")
plt.legend(["ris-virginica", "Iris-setosa","ris-virginica"]);
```



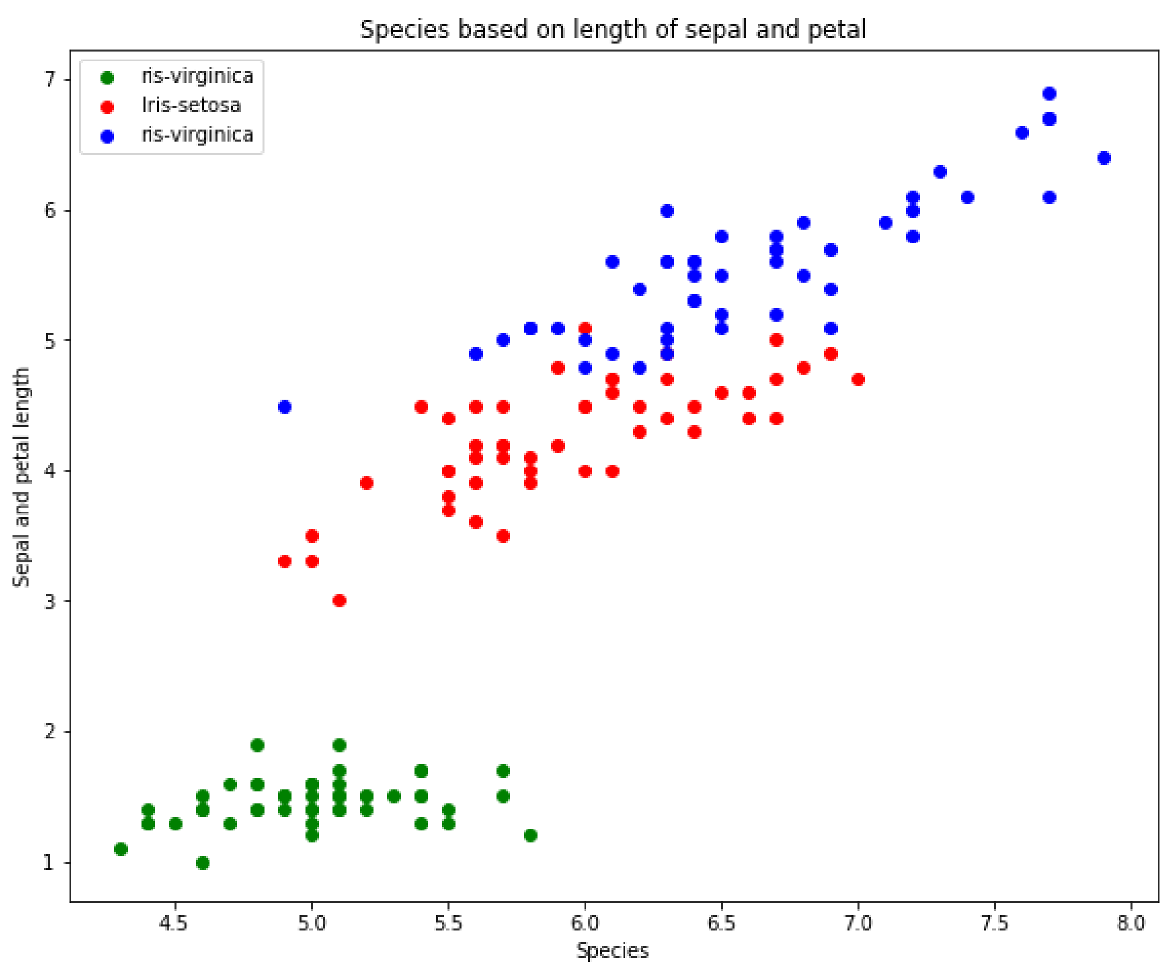
In [27]:

```
# Create another figure Petal_Length SepalLengthCm
plt.figure(figsize=(10, 8))

# Scatter with ris-virginica
plt.scatter(data.Petal_Length[data.Species==0],
            data.Sepal_Width[data.Species==0],
            c="green")

# Scatter with Iris-setosa
plt.scatter(data.Petal_Length[data.Species==1],
            data.Sepal_Width[data.Species==1],
            c="red")

# Scatter with ris-virginica
plt.scatter(data.Petal_Length[data.Species==2],
            data.Sepal_Width[data.Species==2],
            c="blue")

# Add some helpful info
plt.title("Species based on width of sepal and length of petal")
plt.xlabel("Species")
plt.ylabel("Sepal width and petal length")
plt.legend(["ris-virginica", "Iris-setosa","ris-virginica"]);
```



In [28]:

```
# Create another figure Petal_Length Sepal_Length
plt.figure(figsize=(10, 8))

# Scatter with ris-virginica
plt.scatter(data.Sepal_Length[data.Species==0],
            data.Petal_Length[data.Species==0],
            c="green")

# Scatter with Iris-setosa
plt.scatter(data.Sepal_Length[data.Species==1],
            data.Petal_Length[data.Species==1],
            c="red")

# Scatter with ris-virginica
plt.scatter(data.Sepal_Length[data.Species==2],
            data.Petal_Length[data.Species==2],
            c="blue")

# Add some helpful info
plt.title("Species based on length of sepal and petal")
plt.xlabel("Species")
plt.ylabel("Sepal and petal length")
plt.legend(["ris-virginica", "Iris-setosa","ris-virginica"]);
```



In [29]:

```
data.corr() # Pearson Correlation Coefficients
```

Out[29]:

	Species	Sepal_Length	Petal_Length	Sepal_Width	Petal_Width
Species	1.000000	0.782561	0.949043	-0.419446	0.956464
Sepal_Length	0.782561	1.000000	0.871754	-0.109369	0.817954
Petal_Length	0.949043	0.871754	1.000000	-0.420516	0.962757
Sepal_Width	-0.419446	-0.109369	-0.420516	1.000000	-0.356544
Petal_Width	0.956464	0.817954	0.962757	-0.356544	1.000000

In [30]:

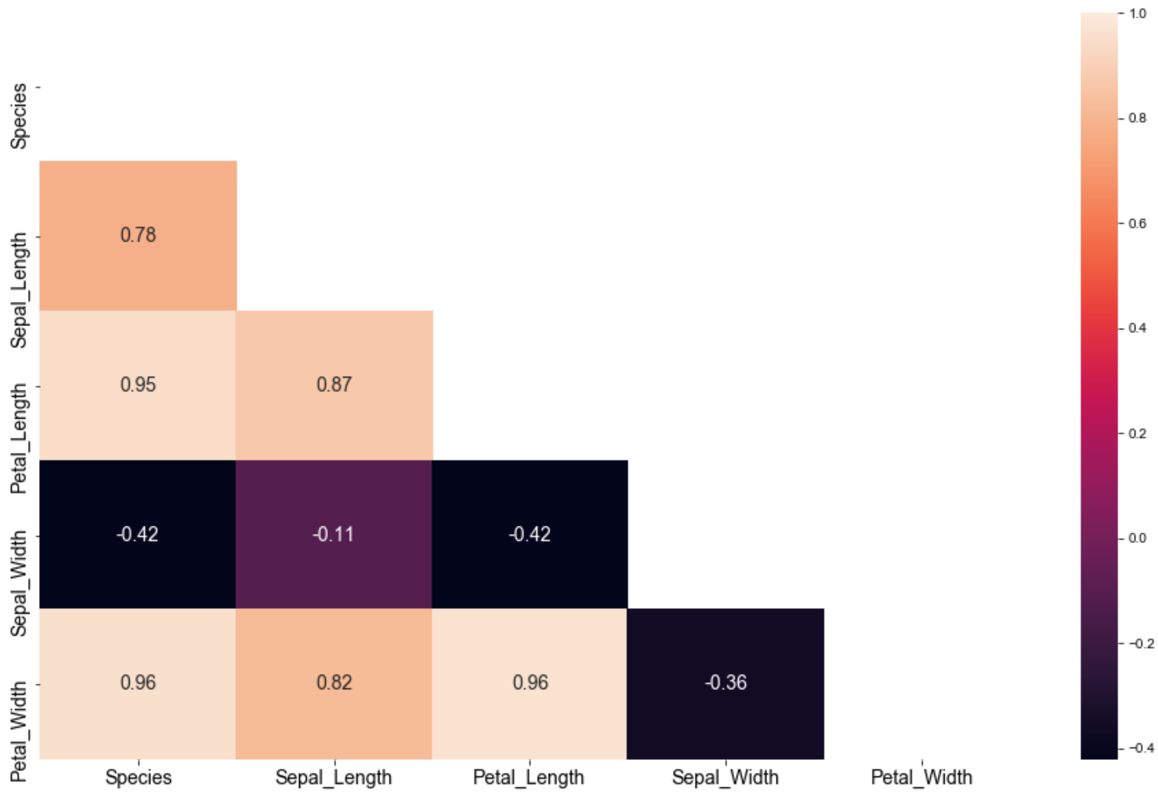
```
mask = np.zeros_like(data.corr())
triangle_indices = np.triu_indices_from(mask)
mask[triangle_indices] = True
mask
```

Out[30]:

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

In [31]:

```
plt.figure(figsize=(16,10))
sns.heatmap(data.corr(), mask=mask, annot=True, annot_kws={"size": 14})
sns.set_style('white')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



In [32]:

```
categorical_features.remove('Species')
dataset = pd.get_dummies(data, columns = categorical_features)
```

In [33]:

```
dataset.head()
```

Out[33]:

	Species	Sepal_Length	Petal_Length	Sepal_Width	Petal_Width
0	0	5.1	1.4	3.5	0.2
1	0	4.9	1.4	3.0	0.2
2	0	4.7	1.3	3.2	0.2
3	0	4.6	1.5	3.1	0.2
4	0	5.0	1.4	3.6	0.2

In [34]:

```
print(data.columns)
print(dataset.columns)
```

```
Index(['Species', 'Sepal_Length', 'Petal_Length', 'Sepal_Width',
       'Petal_Width'],
      dtype='object')
Index(['Species', 'Sepal_Length', 'Petal_Length', 'Sepal_Width',
       'Petal_Width'],
      dtype='object')
```

In [35]:

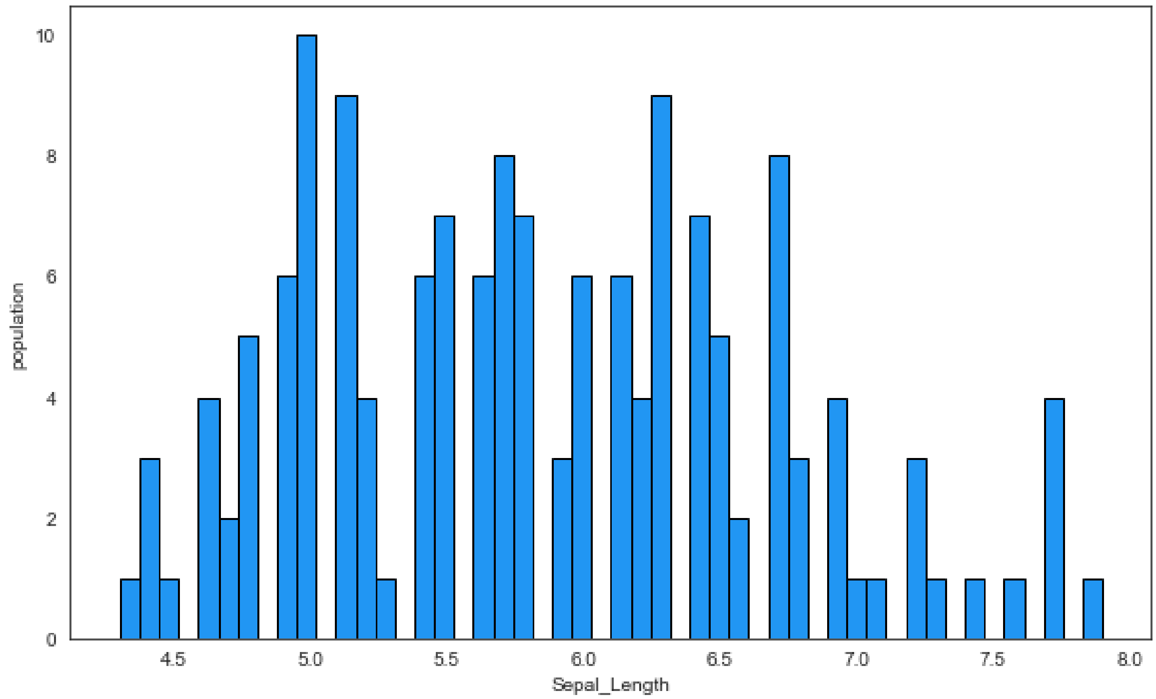
```
dataset.describe()
```

Out[35]:

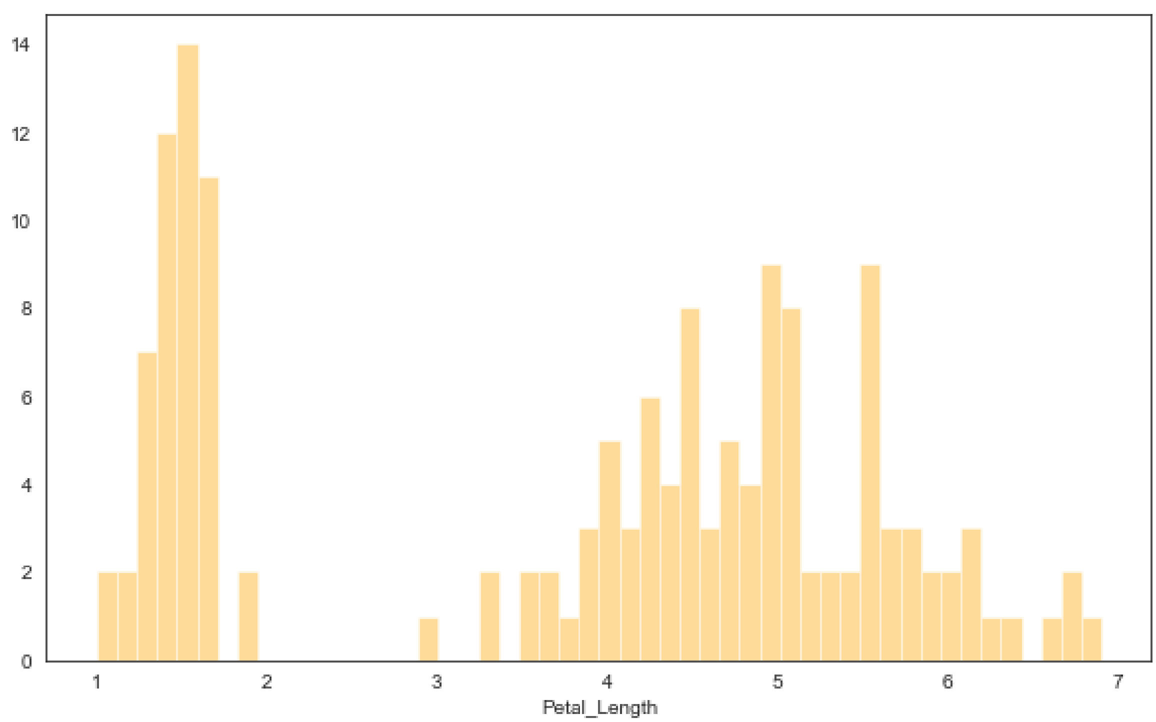
	Species	Sepal_Length	Petal_Length	Sepal_Width	Petal_Width
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	1.000000	5.843333	3.758667	3.054000	1.198667
std	0.819232	0.828066	1.764420	0.433594	0.763161
min	0.000000	4.300000	1.000000	2.000000	0.100000
25%	0.000000	5.100000	1.600000	2.800000	0.300000
50%	1.000000	5.800000	4.350000	3.000000	1.300000
75%	2.000000	6.400000	5.100000	3.300000	1.800000
max	2.000000	7.900000	6.900000	4.400000	2.500000

Visualising Data - Histograms, Distributions and Bar Charts

In [36]:
```
plt.figure(figsize=(10, 6))
plt.hist(dataset['Sepal_Length'], bins=50, ec='black', color='#2196f3')
plt.xlabel('Sepal_Length')
plt.ylabel('population')
plt.show()
```

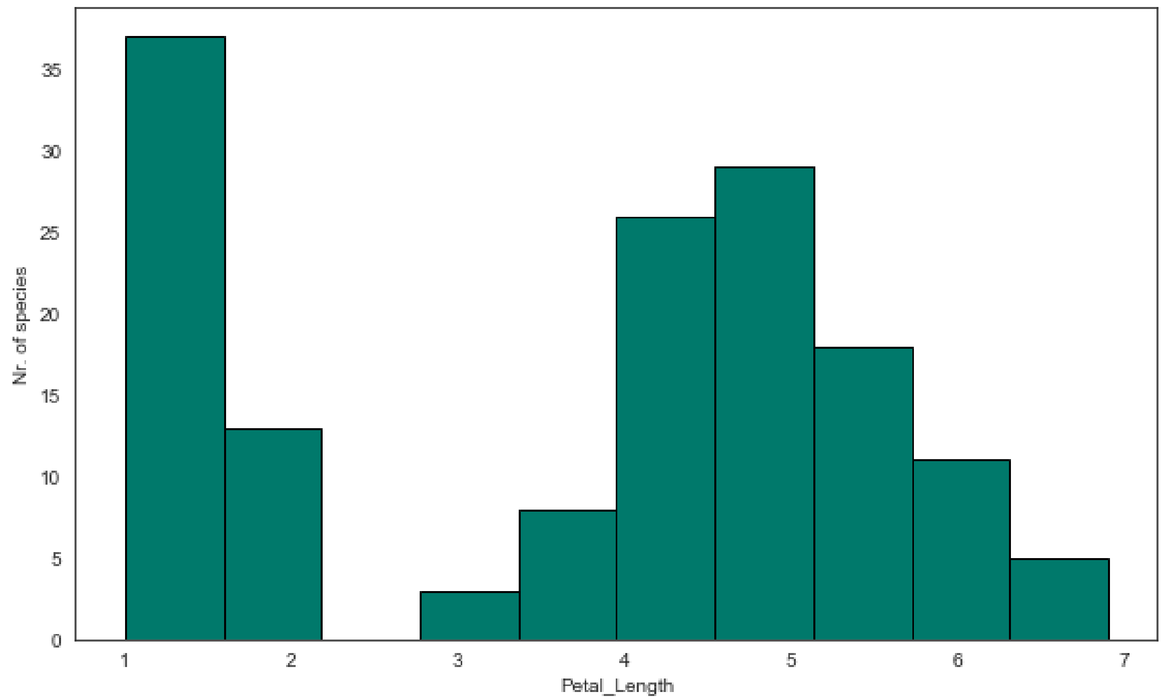


In [37]:
```
plt.figure(figsize=(10, 6))
sns.distplot(dataset['Petal_Length'], bins=50, hist=True, kde=False, color='c
plt.show()
```



In [38]:

```
plt.figure(figsize=(10, 6))
plt.hist(dataset['Petal_Length'], ec='black', color='#00796b')
plt.xlabel('Petal_Length')
plt.ylabel('Nr. of species')
plt.show()
```



In [40]:

```
from sklearn.preprocessing import StandardScaler

s_sc = StandardScaler()
col_to_scale = ['Sepal_Length', 'Petal_Length', 'Sepal_Width',  'Petal_Width'
dataset[col_to_scale] = s_sc.fit_transform(dataset[col_to_scale])
```

In [41]:

```
dataset.head()
```

Out[41]:

	Species	Sepal_Length	Petal_Length	Sepal_Width	Petal_Width
0	0	-0.900681	-1.341272	1.032057	-1.312977
1	0	-1.143017	-1.341272	-0.124958	-1.312977
2	0	-1.385353	-1.398138	0.337848	-1.312977
3	0	-1.506521	-1.284407	0.106445	-1.312977
4	0	-1.021849	-1.341272	1.263460	-1.312977

In [42]:

```
dataset.describe()
```

Out[42]:

	Species	Sepal_Length	Petal_Length	Sepal_Width	Petal_Width
count	150.000000	1.500000e+02	1.500000e+02	1.500000e+02	1.500000e+02
mean	1.000000	1.049161e-16	-2.649732e-16	-9.150088e-17	1.609823e-15
std	0.819232	1.003350e+00	1.003350e+00	1.003350e+00	1.003350e+00
min	0.000000	-1.870024e+00	-1.568735e+00	-2.438987e+00	-1.444450e+00
25%	0.000000	-9.006812e-01	-1.227541e+00	-5.877635e-01	-1.181504e+00
50%	1.000000	-5.250608e-02	3.362659e-01	-1.249576e-01	1.332259e-01
75%	2.000000	6.745011e-01	7.627586e-01	5.692513e-01	7.905908e-01
max	2.000000	2.492019e+00	1.786341e+00	3.114684e+00	1.710902e+00

In [43]:

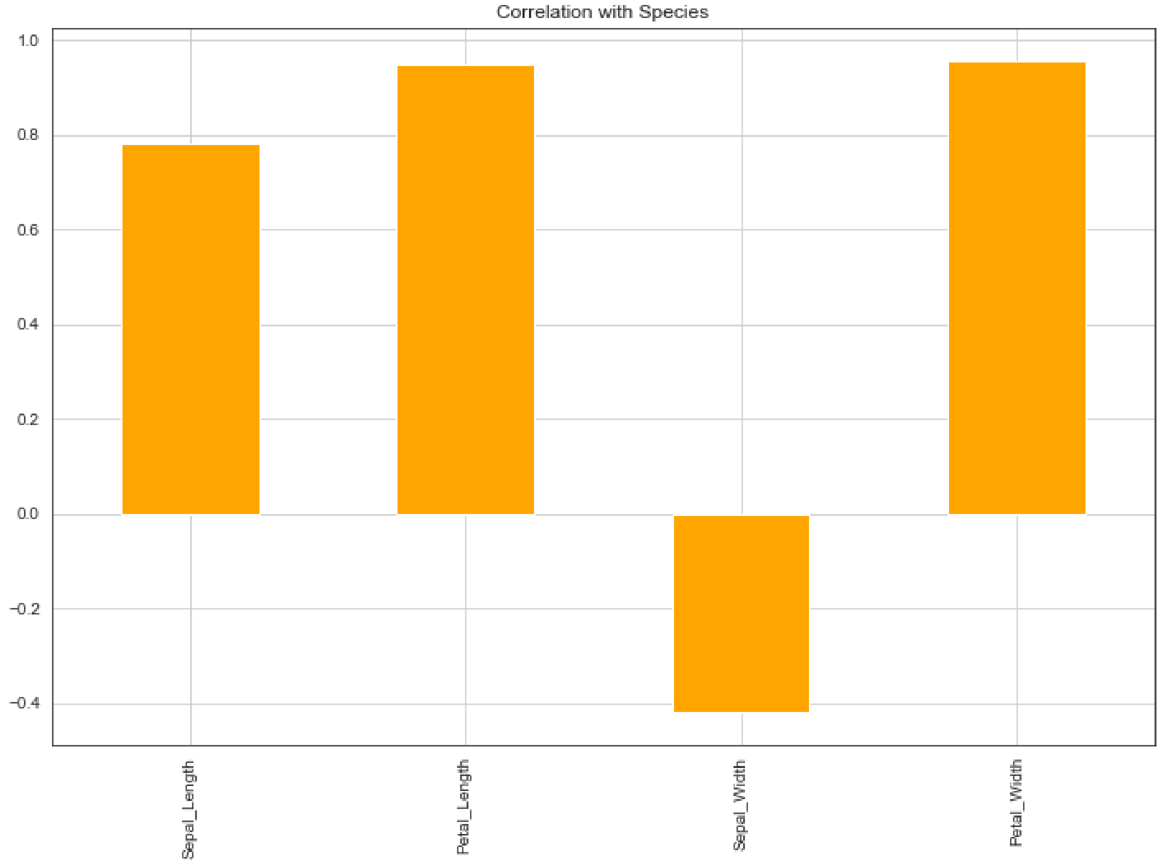
```
dataset.corr() # Pearson Correlation Coefficients
```

Out[43]:

	Species	Sepal_Length	Petal_Length	Sepal_Width	Petal_Width
Species	1.000000	0.782561	0.949043	-0.419446	0.956464
Sepal_Length	0.782561	1.000000	0.871754	-0.109369	0.817954
Petal_Length	0.949043	0.871754	1.000000	-0.420516	0.962757
Sepal_Width	-0.419446	-0.109369	-0.420516	1.000000	-0.356544
Petal_Width	0.956464	0.817954	0.962757	-0.356544	1.000000

In [44]:
```
dataset.drop('Species', axis=1).corrwith(dataset.Species).plot(kind='bar', gr
                                                                title="Correlation with Sp
```

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x248ce7265c8>



In [45]:
```
dataset.columns
```

Out[45]:
```
Index(['Species', 'Sepal_Length', 'Petal_Length', 'Sepal_Width',
       'Petal_Width'],
      dtype='object')
```

In [46]:
```
dataset['Species'].corr(dataset['Sepal_Length'])
```

Out[46]:
```
0.7825612318100819
```

In [47]:
```
dataset['Species'].corr(dataset['Sepal_Width'])
```

Out[47]:
```
-0.4194462002600275
```

In [48]:
```
dataset['Species'].corr(dataset['Petal_Length'])
```

Out[48]:
```
0.949042544852334
```

In [49]:
```
dataset['Species'].corr(dataset['Petal_Width'])
```

Out[49]:
```
0.9564638238016154
```

Machine Learning algorithms application

In [50]:

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

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        print("Train Result:\n================================================
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("Classification Report:", end='')
        print(f"\tPrecision Score: {precision_score(y_train, pred, average='m
        print(f"\t\t\tRecall Score: {recall_score(y_train, pred, average='mic
        print(f"\t\t\tF1 score: {f1_score(y_train, pred, average='micro') * 1
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        print("Test Result:\n================================================
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("Classification Report:", end='')
        print(f"\tPrecision Score: {precision_score(y_test, pred, average='mi
        print(f"\t\t\tRecall Score: {recall_score(y_test,pred, average='micro
        print(f"\t\t\tF1 score: {f1_score(y_test, pred, average='micro') * 10
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

In [51]:

```
from sklearn.model_selection import train_test_split

X = dataset.drop('Species', axis=1)
y = dataset.Species

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ranc
```

In [181]:

```
#array = data.values
#X= data.iloc[:, :-1].values
#Y=data.iloc[:, 4].values
#from sklearn.model_selection import train_test_split
#X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.2, ra
```

In [52]:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(solver='liblinear')
log_reg.fit(X_train, y_train)
```

Out[52]:

```
LogisticRegression(solver='liblinear')
```

In [53]:

```
print_score(log_reg, X_train, y_train, X_test, y_test, train=True)
print_score(log_reg, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 90.83%
_______________________________________________
Classification Report:  Precision Score: 90.83%
                        Recall Score: 90.83%
                        F1 score: 90.83%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 28  9]
 [ 0  2 42]]

Test Result:
================================================
Accuracy Score: 86.67%
_______________________________________________
Classification Report:  Precision Score: 86.67%
                        Recall Score: 86.67%
                        F1 score: 86.67%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 10  3]
 [ 0  1  5]]
```

In [54]:

```
test_score = accuracy_score(y_test, log_reg.predict(X_test)) * 100
train_score = accuracy_score(y_train, log_reg.predict(X_train)) * 100

results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_sco
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df
```

Out[54]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	90.833333	86.666667

K-nearest neighbors

In [55]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)

print_score(knn_classifier, X_train, y_train, X_test, y_test, train=True)
print_score(knn_classifier, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 95.83%
_______________________________________________
Classification Report:  Precision Score: 95.83%
                        Recall Score: 95.83%
                        F1 score: 95.83%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 34  3]
 [ 0  2 42]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

In [56]:

```
test_score = accuracy_score(y_test, knn_classifier.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_classifier.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_s
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[56]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	90.833333	86.666667
1	K-nearest neighbors	95.833333	100.000000

Support Vector machine

In [57]:
```
from sklearn.svm import SVC


svm_model = SVC(kernel='rbf', gamma=0.1, C=1.0)
svm_model.fit(X_train, y_train)
```

Out[57]:
```
SVC(gamma=0.1)
```

In [58]:
```
print_score(svm_model, X_train, y_train, X_test, y_test, train=True)
print_score(svm_model, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 96.67%
_______________________________________________
Classification Report:  Precision Score: 96.67%
                        Recall Score: 96.67%
                        F1 score: 96.67%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 35  2]
 [ 0  2 42]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

In [59]:
```
test_score = accuracy_score(y_test, svm_model.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_model.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Support Vector Machine", train_score, tes
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[59]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	90.833333	86.666667
1	K-nearest neighbors	95.833333	100.000000
2	Support Vector Machine	96.666667	100.000000

Decision Tree Classifier

In [60]:
```
from sklearn.tree import DecisionTreeClassifier


tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)

print_score(tree, X_train, y_train, X_test, y_test, train=True)
print_score(tree, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 37  0]
 [ 0  0 44]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

In [61]:
```
test_score = accuracy_score(y_test, tree.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Decision Tree Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[61]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	90.833333	86.666667
1	K-nearest neighbors	95.833333	100.000000
2	Support Vector Machine	96.666667	100.000000
3	Decision Tree Classifier	100.000000	100.000000

Random Forest

In [62]:

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

rand_forest = RandomForestClassifier(n_estimators=1000, random_state=42)
rand_forest.fit(X_train, y_train)

print_score(rand_forest, X_train, y_train, X_test, y_test, train=True)
print_score(rand_forest, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 37  0]
 [ 0  0 44]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

In [63]:

```
test_score = accuracy_score(y_test, rand_forest.predict(X_test)) * 100
train_score = accuracy_score(y_train, rand_forest.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Random Forest Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[63]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	90.833333	86.666667
1	K-nearest neighbors	95.833333	100.000000
2	Support Vector Machine	96.666667	100.000000
3	Decision Tree Classifier	100.000000	100.000000
4	Random Forest Classifier	100.000000	100.000000

XGBoost Classifer

In [254]:

```
#pip install xgboost     installing xgboost
```

In [64]:
```
from xgboost import XGBClassifier

xgb = XGBClassifier()
xgb.fit(X_train, y_train)

print_score(xgb, X_train, y_train, X_test, y_test, train=True)
print_score(xgb, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 37  0]
 [ 0  0 44]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

In [65]:
```
test_score = accuracy_score(y_test, xgb.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["XGBoost Classifier", train_score, test_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[65]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	90.833333	86.666667
1	K-nearest neighbors	95.833333	100.000000
2	Support Vector Machine	96.666667	100.000000
3	Decision Tree Classifier	100.000000	100.000000
4	Random Forest Classifier	100.000000	100.000000
5	XGBoost Classifier	100.000000	100.000000

Using Hyperparameter Tuning

Logistic Regression Hyperparameter Tuning

In [66]:
```
from sklearn.model_selection import GridSearchCV

params = {"C": np.logspace(-4, 4, 20),
          "solver": ["liblinear"]}

log_reg = LogisticRegression()

grid_search_cv = GridSearchCV(log_reg, params, scoring="accuracy", n_jobs=-1,
# grid_search_cv.fit(X_train, y_train)
```

In [67]:
```
# grid_search_cv.best_estimator_
```

In [68]:
```
log_reg = LogisticRegression(C=0.615848211066026,
                             solver='liblinear')

log_reg.fit(X_train, y_train)

print_score(log_reg, X_train, y_train, X_test, y_test, train=True)
print_score(log_reg, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 88.33%
_______________________________________________
Classification Report:  Precision Score: 88.33%
                        Recall Score: 88.33%
                        F1 score: 88.33%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 25 12]
 [ 0  2 42]]

Test Result:
================================================
Accuracy Score: 80.00%
_______________________________________________
Classification Report:  Precision Score: 80.00%
                        Recall Score: 80.00%
                        F1 score: 80.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0  8  5]
 [ 0  1  5]]
```

In [69]:

```
test_score = accuracy_score(y_test, log_reg.predict(X_test)) * 100
train_score = accuracy_score(y_train, log_reg.predict(X_train)) * 100

tuning_results_df = pd.DataFrame(data=[["Tuned Logistic Regression", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df
```

Out[69]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.333333	80.0

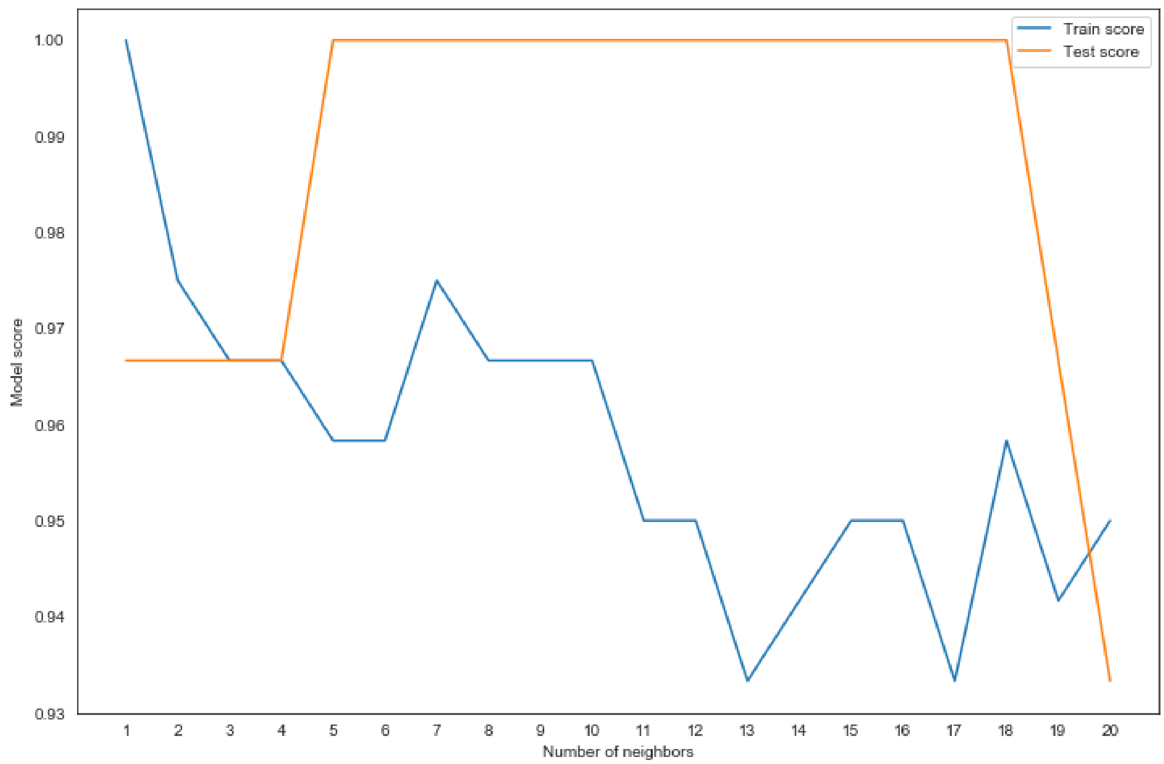
K-nearest neighbors Hyperparameter Tuning

In [70]:

```
train_score = []
test_score = []
neighbors = range(1, 21)

for k in neighbors:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    train_score.append(accuracy_score(y_train, model.predict(X_train)))
    test_score.append(accuracy_score(y_test, model.predict(X_test)))
```

In [71]:
```
plt.figure(figsize=(12, 8))

plt.plot(neighbors, train_score, label="Train score")
plt.plot(neighbors, test_score, label="Test score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

print(f"Maximum KNN score on the test data: {max(test_score)*100:.2f}%")
```

```
Maximum KNN score on the test data: 100.00%
```



In [72]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=19)
knn_classifier.fit(X_train, y_train)

print_score(knn_classifier, X_train, y_train, X_test, y_test, train=True)
print_score(knn_classifier, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 94.17%
_______________________________________________
Classification Report:  Precision Score: 94.17%
                        Recall Score: 94.17%
                        F1 score: 94.17%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 32  5]
 [ 0  2 42]]

Test Result:
================================================
Accuracy Score: 96.67%
_______________________________________________
Classification Report:  Precision Score: 96.67%
                        Recall Score: 96.67%
                        F1 score: 96.67%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
```

In [73]:

```
test_score = accuracy_score(y_test, knn_classifier.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_classifier.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned K-nearest neighbors", train_score,
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[73]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.333333	80.000000
1	Tuned K-nearest neighbors	94.166667	96.666667

In [74]:

```
### Support Vector Machine Hyperparameter Tuning
```

In [75]:
```
svm_model = SVC(kernel='rbf', gamma=0.1, C=1.0)

params = {"C":(0.1, 0.5, 1, 2, 5, 10, 20),
          "gamma":(0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 1),
          "kernel":('linear', 'poly', 'rbf')}

svm_grid = GridSearchCV(svm_model, params, n_jobs=-1, cv=5, verbose=1, scorir
# svm_grid.fit(X_train, y_train)
```

In [76]:
```
# svm_grid.best_estimator_
```

In [77]:
```
svm_model = SVC(C=5, gamma=0.01, kernel='rbf')
svm_model.fit(X_train, y_train)

print_score(svm_model, X_train, y_train, X_test, y_test, train=True)
print_score(svm_model, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 95.83%
_______________________________________________
Classification Report:  Precision Score: 95.83%
                        Recall Score: 95.83%
                        F1 score: 95.83%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 35  2]
 [ 0  3 41]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

In [78]:

```
test_score = accuracy_score(y_test, svm_model.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_model.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Support Vector Machine", train_scor
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[78]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.333333	80.000000
1	Tuned K-nearest neighbors	94.166667	96.666667
2	Tuned Support Vector Machine	95.833333	100.000000

Decision Tree Classifier Hyperparameter Tuning

In [79]:

```
params = {"criterion":("gini", "entropy"),
          "splitter":("best", "random"),
          "max_depth":(list(range(1, 20))),
          "min_samples_split":[2, 3, 4],
          "min_samples_leaf":list(range(1, 20))
          }

tree = DecisionTreeClassifier(random_state=42)
grid_search_cv = GridSearchCV(tree, params, scoring="accuracy", n_jobs=-1, ve
# grid_search_cv.fit(X_train, y_train)
```

In [80]:

```
# grid_search_cv.best_estimator_
```

In [81]:

```
tree = DecisionTreeClassifier(criterion='gini',
                              max_depth=3,
                              min_samples_leaf=2,
                              min_samples_split=2,
                              splitter='random')
tree.fit(X_train, y_train)

print_score(tree, X_train, y_train, X_test, y_test, train=True)
print_score(tree, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 88.33%
_______________________________________________
Classification Report:  Precision Score: 88.33%
                        Recall Score: 88.33%
                        F1 score: 88.33%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 1 36  0]
 [ 0 13 31]]

Test Result:
================================================
Accuracy Score: 90.00%
_______________________________________________
Classification Report:  Precision Score: 90.00%
                        Recall Score: 90.00%
                        F1 score: 90.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  3  3]]
```

In [82]:

```
test_score = accuracy_score(y_test, tree.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Decision Tree Classifier", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[82]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.333333	80.000000
1	Tuned K-nearest neighbors	94.166667	96.666667
2	Tuned Support Vector Machine	95.833333	100.000000
3	Tuned Decision Tree Classifier	88.333333	90.000000

Random Forest Classifier Hyperparameter Tuning

In [83]:
```
from sklearn.model_selection import RandomizedSearchCV

n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators, 'max_features': max_features,
               'max_depth': max_depth, 'min_samples_split': min_samples_split
               'min_samples_leaf': min_samples_leaf, 'bootstrap': bootstrap}


rand_forest = RandomForestClassifier(random_state=42)

rf_random = RandomizedSearchCV(estimator=rand_forest, param_distributions=rar
                               verbose=2, random_state=42, n_jobs=-1)
# rf_random.fit(X_train, y_train)
```

In [84]:
```
# rf_random.best_estimator_
```

In [85]:
```
rand_forest = RandomForestClassifier(bootstrap=True,
                                     max_depth=70,
                                     max_features='auto',
                                     min_samples_leaf=4,
                                     min_samples_split=10,
                                     n_estimators=400)
rand_forest.fit(X_train, y_train)
```

Out[85]:
```
RandomForestClassifier(max_depth=70, min_samples_leaf=4, min_samples_split=
10,
                       n_estimators=400)
```

In [86]:
```
print_score(rand_forest, X_train, y_train, X_test, y_test, train=True)
print_score(rand_forest, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 96.67%
_______________________________________________
Classification Report:  Precision Score: 96.67%
                        Recall Score: 96.67%
                        F1 score: 96.67%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 34  3]
 [ 0  1 43]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

In [87]:
```
test_score = accuracy_score(y_test, rand_forest.predict(X_test)) * 100
train_score = accuracy_score(y_train, rand_forest.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Random Forest Classifier", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[87]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.333333	80.000000
1	Tuned K-nearest neighbors	94.166667	96.666667
2	Tuned Support Vector Machine	95.833333	100.000000
3	Tuned Decision Tree Classifier	88.333333	90.000000
4	Tuned Random Forest Classifier	96.666667	100.000000

XGBoost Classifier Hyperparameter Tuning

In [88]:
```
n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
booster = ['gbtree', 'gblinear']
base_score = [0.25, 0.5, 0.75, 0.99]
learning_rate = [0.05, 0.1, 0.15, 0.20]
min_child_weight = [1, 2, 3, 4]

hyperparameter_grid = {'n_estimators': n_estimators, 'max_depth': max_depth,
                       'learning_rate' : learning_rate, 'min_child_weight' :
                       'booster' : booster, 'base_score' : base_score
                        }

xgb_model = XGBClassifier()

xgb_cv = RandomizedSearchCV(estimator=xgb_model, param_distributions=hyperpar
                               cv=5, n_iter=650, scoring = 'accuracy',n_jobs
                               verbose=1, return_train_score = True, random_s


# xgb_cv.fit(X_train, y_train)
```

In [89]:
```
# xgb_cv.best_estimator_
```

In [90]:
```
xgb_best = XGBClassifier(base_score=0.25,
                         booster='gbtree',
                         learning_rate=0.05,
                         max_depth=5,
                         min_child_weight=2,
                         n_estimators=100)
xgb_best.fit(X_train, y_train)
```

Out[90]:
```
XGBClassifier(base_score=0.25, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.05, max_delta_step=0, max_depth=5,
              min_child_weight=2, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [91]:

```
print_score(xgb_best, X_train, y_train, X_test, y_test, train=True)
print_score(xgb_best, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 98.33%
_______________________________________________
Classification Report:	Precision Score: 98.33%
			Recall Score: 98.33%
			F1 score: 98.33%
_______________________________________________
Confusion Matrix:
 [[39  0  0]
 [ 0 35  2]
 [ 0  0 44]]

Test Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:	Precision Score: 100.00%
			Recall Score: 100.00%
			F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```

In [92]:

```
test_score = accuracy_score(y_test, xgb_best.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb_best.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned XGBoost Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[92]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	88.333333	80.000000
1	Tuned K-nearest neighbors	94.166667	96.666667
2	Tuned Support Vector Machine	95.833333	100.000000
3	Tuned Decision Tree Classifier	88.333333	90.000000
4	Tuned Random Forest Classifier	96.666667	100.000000
5	Tuned XGBoost Classifier	98.333333	100.000000

In [93]:
```
results_df
```

Out[93]:

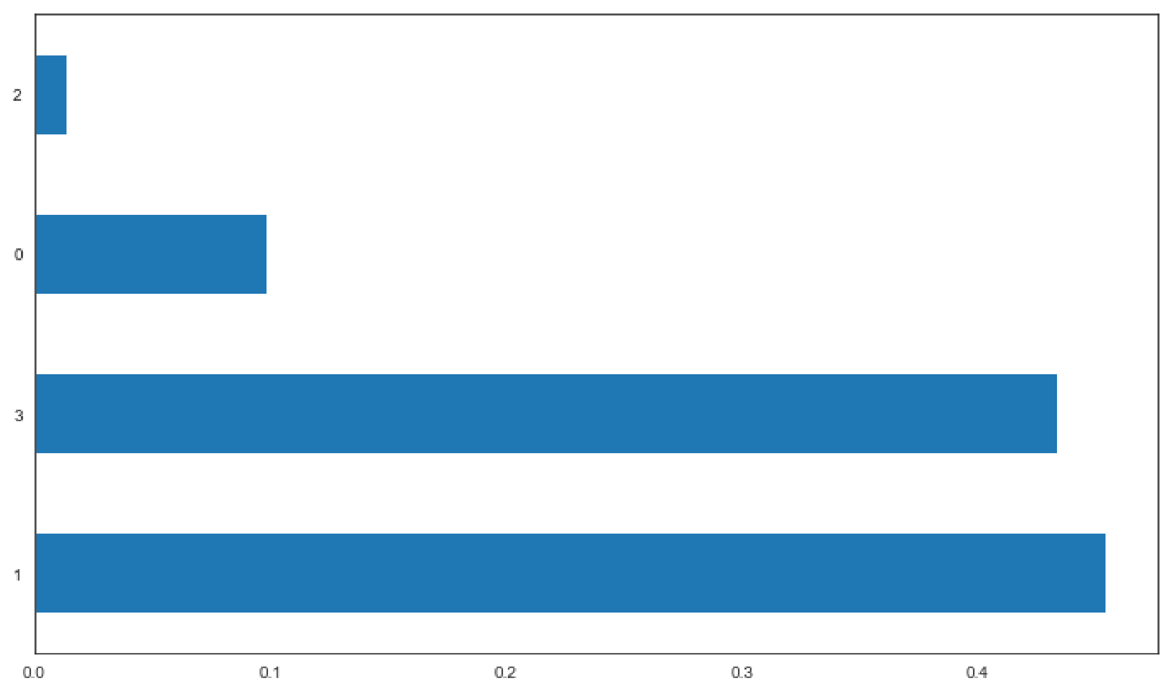
	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	90.833333	86.666667
1	K-nearest neighbors	95.833333	100.000000
2	Support Vector Machine	96.666667	100.000000
3	Decision Tree Classifier	100.000000	100.000000
4	Random Forest Classifier	100.000000	100.000000
5	XGBoost Classifier	100.000000	100.000000

Features Importance According to Random Forest and XGBoost

In [94]:
```
def feature_imp(df, model):
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importances_
    return fi.sort_values(by="importance", ascending=False)
```

In [95]:
```
feature_imp(X, rand_forest).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0x248ce978088>



In [96]:
```
feature_imp(X, xgb_best).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[96]: <matplotlib.axes._subplots.AxesSubplot at 0x248ce74d808>