Model Selection

Libraries

In [ ]:
```
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#import machine learning libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

DataSet

In [ ]:
```
#import dataset
df = pd.read_csv('FlowersData.csv')
df
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	flower_name
0	5.1	3.5	1.4	0.2	hibiscus
1	4.9	3.0	1.4	0.2	hibiscus
2	4.7	3.2	1.3	0.2	hibiscus
3	4.6	3.1	1.5	0.2	hibiscus
4	5.0	3.6	1.4	0.2	hibiscus
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	lily
146	6.3	2.5	5.0	1.9	lily
147	6.5	3.0	5.2	2.0	lily
148	6.2	3.4	5.4	2.3	lily
149	5.9	3.0	5.1	1.8	lily

150 rows × 5 columns

Heads

In [ ]:
```
#head
df.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	flower_name
0	5.1	3.5	1.4	0.2	hibiscus
1	4.9	3.0	1.4	0.2	hibiscus
2	4.7	3.2	1.3	0.2	hibiscus
3	4.6	3.1	1.5	0.2	hibiscus
4	5.0	3.6	1.4	0.2	hibiscus

_Doing Linear Regreesion on columns sepal_length and petal_length_

In [ ]:

```
#Selecting two columns for Linear Regression
df_lr=df[['sepal_length','petal_length']]
df_lr.head()
```

Out[ ]:

	sepal_length	petal_length
0	5.1	1.4
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5
4	5.0	1.4

Checking for null values

In [ ]:

```
#Checking for null values
df_lr.isnull().sum()
```

Out[ ]:

```
sepal_length    0
petal_length    0
dtype: int64
```

No Null Values

Tackling with Outliers

In [ ]:

```
#detecting outliers in sepal_length with iqr method
q1 = df_lr['sepal_length'].quantile(0.25)
q3 = df_lr['sepal_length'].quantile(0.75)
iqr = q3-q1
df_lr=df_lr[(df_lr['sepal_length']>(q1-1.5*iqr)) & (df_lr['sepal_length']<(q1+1.5*iqr)
```

In [ ]:

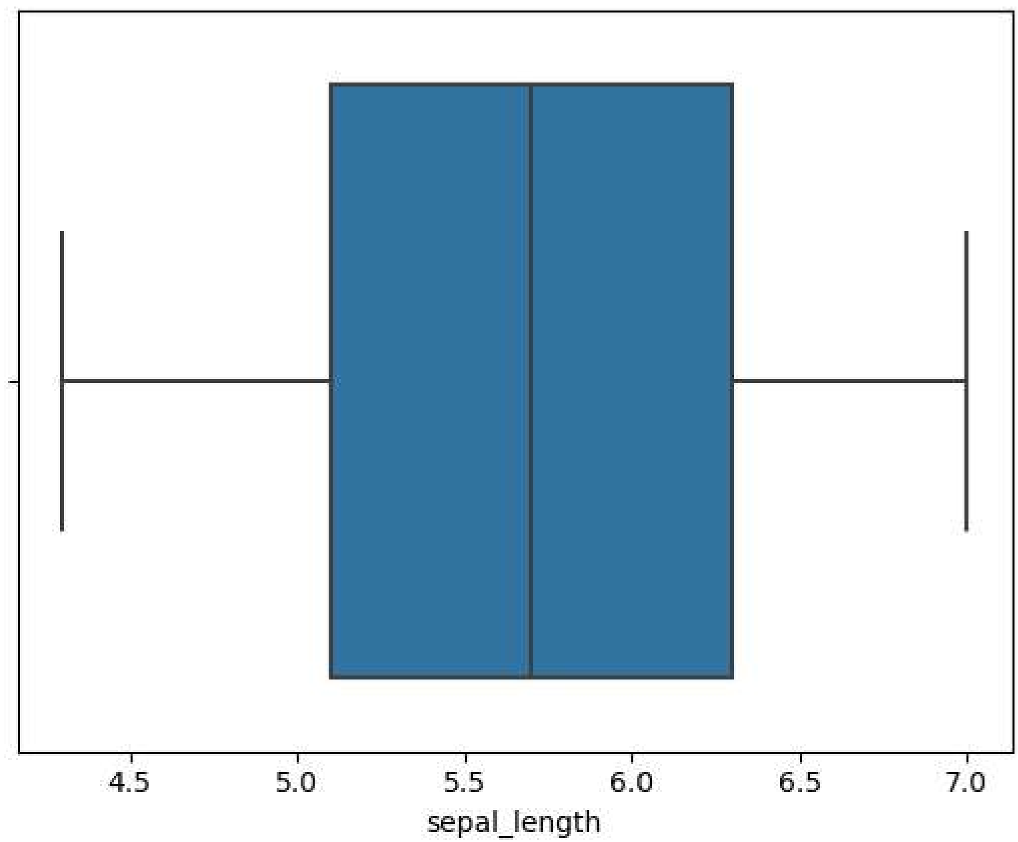
```
#detecting outliers in petal_length with iqr method
q1 = df_lr['petal_length'].quantile(0.25)
q3 = df_lr['petal_length'].quantile(0.75)
iqr = q3-q1
df_lr=df_lr[(df_lr['petal_length']>(q1-1.5*iqr)) & (df_lr['petal_length']<(q1+1.5*iqr)
```

Boxplot

In [ ]:
```
#boxplot for sepal_length
sns.boxplot(df_lr['sepal_length'])
```

```
c:\Users\AL Ghani Computer\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

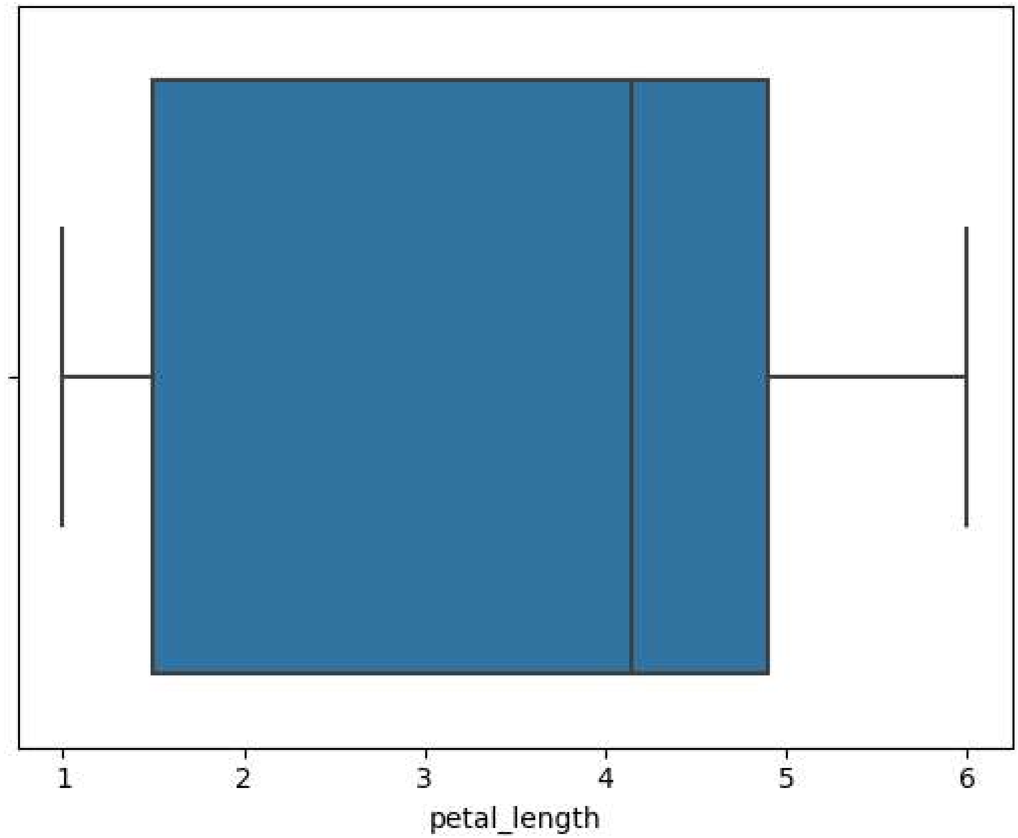
Out[ ]:
```
<AxesSubplot:xlabel='sepal_length'>
```



In [ ]:
```
#boxplot for petal_length
sns.boxplot(df_lr['petal_length'])
```

```
c:\Users\AL Ghani Computer\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[ ]:
```
<AxesSubplot:xlabel='petal_length'>
```



Model Selection

Dependent & Independent

In [ ]:
```
#selectiong dependent and independent variables
x = df_lr['sepal_length']
y = df_lr['petal_length']
```

Model Building

In [ ]:
```
#model building
model=LinearRegression()
#fitting the model
model.fit(x.values.reshape(-1,1),y)
```

Out[ ]:
```
LinearRegression()
```

Prediction

In [ ]:
```
#prediction
model.predict([[22]])
# 22 is the sepal length and 36.53 is the predicted petal length
```

Out[ ]:
```
array([36.53174197])
```

In [ ]:
```
#sepal length is entered by the user and predicted petal length is obtained
sepal_length=int(input("Enter the sepal length :"))
print("The predicted petal length is :",model.predict([[sepal_length]]))
```

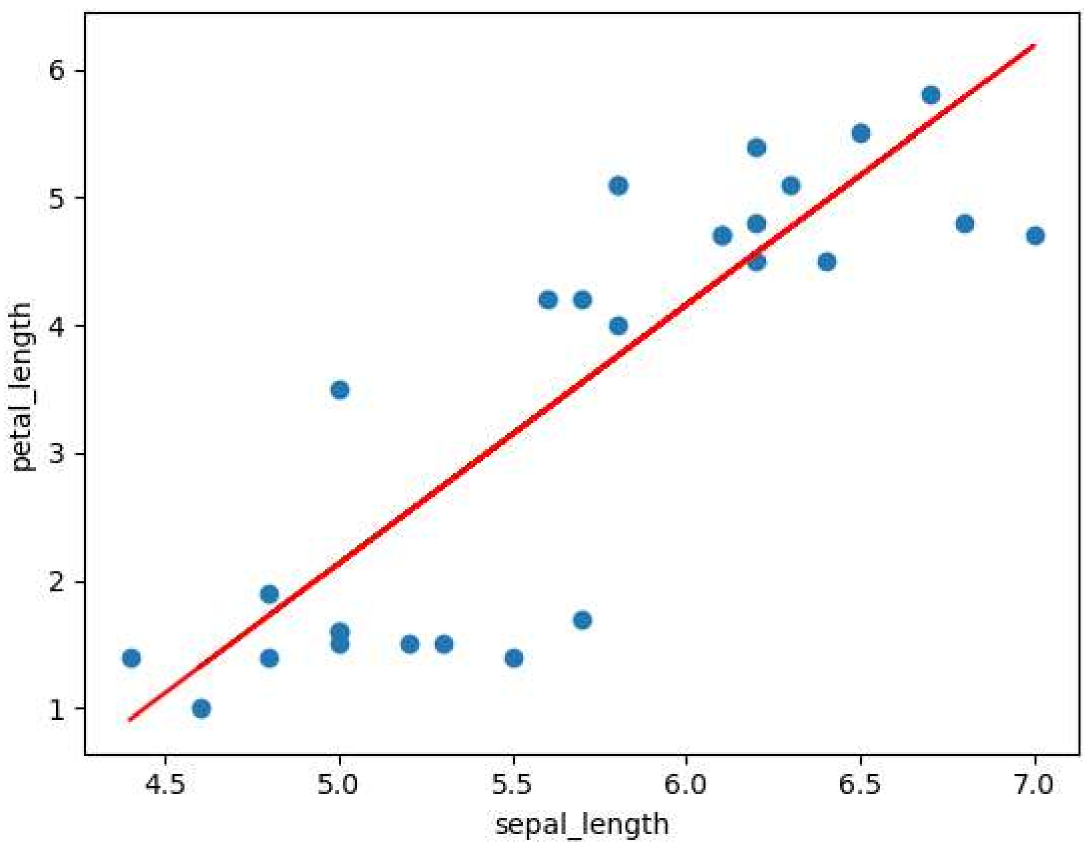
```
The predicted petal length is : [38.55609747]
```

In [ ]:
```
#finding accuracy of the model, we have to train and test data
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
#In linear regression, the "random_state" parameter is used to set a seed for the ranc
# Setting the random_state to a specific value will ensure that the same train-test sp
```

In [ ]:
```
#model building
model=LinearRegression()
#fitting the model
model.fit(x_train.values.reshape(-1,1),y_train)
#predicting the model
y_pred=model.predict(x_test.values.reshape(-1,1))
```

Scatterplot

In [ ]:
```
#scatterplot
plt.scatter(x_test,y_test)
plt.plot(x_test,y_pred,color='red')
plt.xlabel('sepal_length')
plt.ylabel('petal_length')
plt.show()
```



As a straight line is obtained so our model is 100% accurate

Model Score

In [ ]:
```
# Testing accuracy by model score
model.score(x_test.values.reshape(-1,1),y_test)
#In linear regression, the model.score() function is used to evaluate the accuracy of
```

```
#  It value ranges from 0-1, if the value is one it means that the model is perfect
#  and if we obtain 0 or near to 0 values it means our model isn't much accurate.
```

Out[ ]:

```
0.7121393028038641
```

Mean Absolute Error

In [ ]:

```
#evauating the accuracy by MAE
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test,y_pred)
#MAE is the average of the absolute differences between the actual and predicted value
#  It is the easiest to understand, because it's the average error.
```

Out[ ]:

```
0.7255077424143427
```

Mean Squared Error

In [ ]:

```
#evaluating the accuracy by MSE
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,y_pred)
#MSE is the average of the squared differences between the actual and predicted values
#  It's more popular than MAE, because MSE "punishes" larger errors, which tends to be
```

Out[ ]:

```
0.7811415783977846
```

Root Mean Squared Error

In [ ]:

```
#evaluating the accuracy by RMSE
from sklearn.metrics import mean_squared_error
np.sqrt(mean_squared_error(y_test,y_pred))
#RMSE is the square root of the average of the squared differences between the actual
#  It's even more popular than MSE, because RMSE is interpretable in the "y" units.
```

Out[ ]:

```
0.8838221418349874
```

R2 Method

In [ ]:

```
#evaluating the accuracy by R2(R Square)
from sklearn.metrics import r2_score
r2_score(y_test,y_pred)
#R2 is the percentage of the response variable variation that is explained by a linear
#  It is also known as the coefficient of determination, or the coefficient of multipl
```

Out[ ]:

```
0.7121393028038641
```

Mean Absolute Percentage Error

In [ ]:

```
#evaluating the accuracy by MAPE
def mean_absolute_percentage_error(y_test, y_pred):
    y_test, y_pred = np.array(y_test), np.array(y_pred)
    return np.mean(np.abs((y_test - y_pred) / y_test)) * 100
mean_absolute_percentage_error(y_test,y_pred)
#MAPE is the average of the absolute percentage differences between the actual and pre
#  It is the easiest to understand, because it's the average error.
```

Out[ ]:
```
30.05360038080669
```

F-Statistics

In [ ]:
```
#evauating accuracy by F statistics
from sklearn.feature_selection import f_regression
f_regression(x.values.reshape(-1,1),y)
#F statistics is the ratio of the explained variation to the unexplained variation.
#  It is also known as the coefficient of determination, or the coefficient of multipl
```

Out[ ]:
```
(array([353.26807637]), array([1.24551254e-39]))
```