ITDAA Assessment:

Eddie Theron

Question 1:

1.1

```
import pandas as pd #import pandas
import numpy as np #import numpy
from sklearn import datasets, linear_model #import datasets
from matplotlib import pyplot as plt #For use in plotting data
from sklearn.model_selection import train_test_split #Import for splitting data into 80/20


CarMarket = pd.read_csv("C:\\Users\\Eddie\\Desktop\\datasets\\car_market_analysis.csv")

CarMarket.head()

#CarMarket.summary() #Display the summary
```

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	fuelsystem	boreratio	st
0	1	3	alfa-romero giulia	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	
3	4	2	audi 100 ls	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	
4	5	2	audi 100ls	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	

5 rows × 26 columns

```
X = np.array(CarMarket.iloc[:,24]).reshape(-1,1)
y = np.array(CarMarket['price'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

1.2

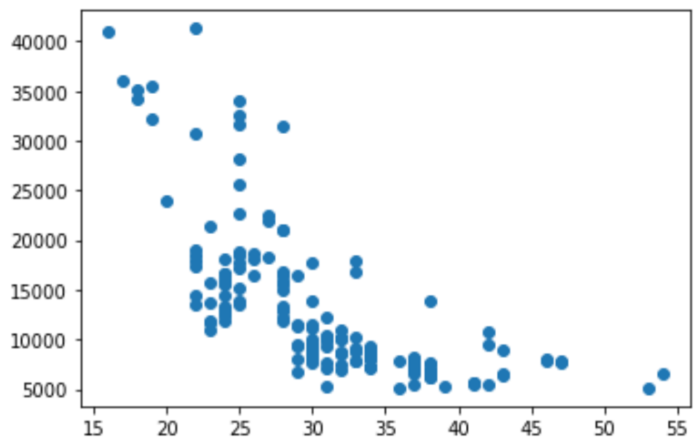
```
from sklearn.linear_model import LinearRegression #Import Linear Regression
LR = LinearRegression() #Create the linear regression model

LR.fit(X,y)#Fit the values
```

```
LinearRegression()
```

1.3

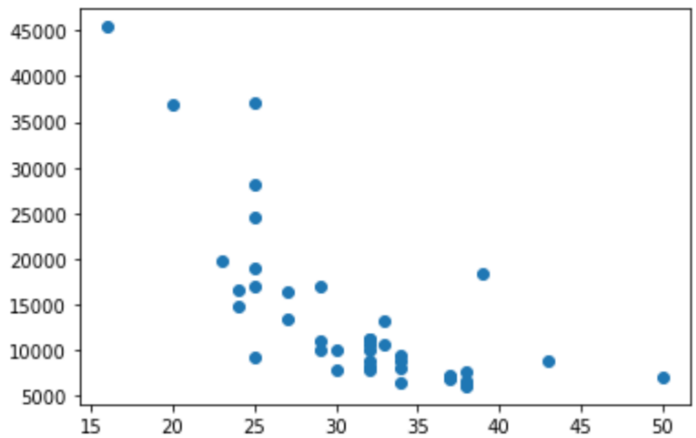
```
LR.fit(X_train, y_train) #Fit the training values
CoeffR2 = LR.score(X_train, y_train)
print(f"Train set coefficient: {CoeffR2}")


plt.scatter(X_train, y_train)

plt.show()
```

```
Train set coefficient: 0.4889120296433751
```



```
LR.fit(X_test, y_test) #Fit the testing values
CoeffR2 = LR.score(X_test, y_test)
print(f"Test set coefficient: {CoeffR2}")

plt.scatter(X_test, y_test)

plt.show()
```

```
Test set coefficient: 0.508822857528752
```



1.4

Pre-processing: The cleaning and preparation of data. This includes normalization, and managing of values that are missing. If the data is in a form that the model can understand for pre-processing, the predictive accuracy can be increased.

Hyperparameters: The optimal values for the hyperparameters of a dataset can be found by tuning it. Various search methods and optimization can be used for tuning which will lead to better prediction accuracy.

Question 2:

2.1

```
import pandas as pd #import pandas
import numpy as np #import numpy
from sklearn import datasets, linear_model #import datasets
from matplotlib import pyplot as plt #For use in plotting data
from sklearn.model_selection import train_test_split #Import for splitting data into 80/20

Diabetes = pd.read_csv("C:\\Users\\Eddie\\Desktop\\datasets\\diabetes_datasets.csv")
```

```
X= np.array(Diabetes.iloc[:,7]).reshape(-1, 1)#Choose all attributes except the class attribute
y = np.array(Diabetes['class']) #The class attribute will be the output
```

2.2

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2) #test percentage is 20 and training would then be 80
```

```
#Naive Bayes training
from sklearn.naive_bayes import GaussianNB
GNB = GaussianNB()
GNB.fit(X_train, y_train)
```

```
GaussianNB()
```

```
#Naive Bayes testing
from sklearn.naive_bayes import GaussianNB
GNB1 = GaussianNB()
GNB1.fit(X_test, y_test)
```

```
GaussianNB()
```

```
#Logistic Regression Training
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
LR.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
#Logistic Regression Testing
from sklearn.linear_model import LogisticRegression
LR1 = LogisticRegression()
LR1.fit(X_test, y_test)
```

```
LogisticRegression()
```

```
#Decision Tree Training
from sklearn import tree
DtC = tree.DecisionTreeClassifier()
DtC.fit(X_train, y_train)
```

```
DecisionTreeClassifier()
```

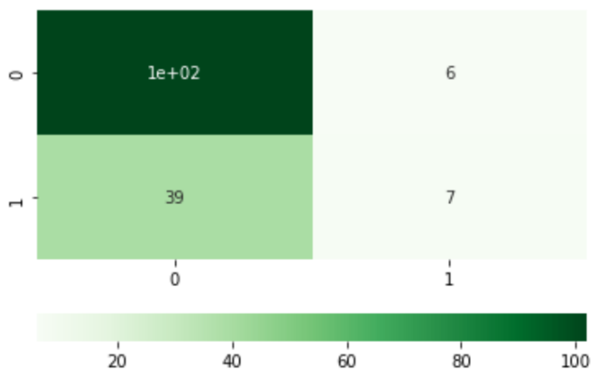
```
#Decision Tree Testing
from sklearn import tree
DtC1 = tree.DecisionTreeClassifier()
DtC1.fit(X_test, y_test)
```

```
DecisionTreeClassifier()
```

2.3

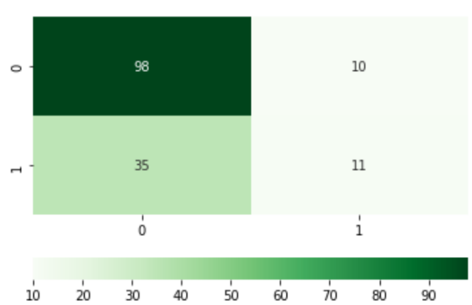
```python
#Confusion Matrix for Naive Bayes Training

import seaborn as sns #Importing for customizing the confusion matrix
from sklearn.metrics import confusion_matrix #Import the confusion Matrix

Fit = GNB
predict = Fit.predict(X_test)
ConfusionM = confusion_matrix(y_test,predict)
print(ConfusionM)

sns.heatmap(ConfusionM,cmap="Greens",annot=True,cbar_kws={"orientation":"horizontal"})
```

```
[[102   6]
 [ 39   7]]

<AxesSubplot:>
```




```python
#Confusion Matrix for Naive Bayes Testing
import seaborn as sns #Importing for customizing the confusion matrix
from sklearn.metrics import confusion_matrix #Import the confusion Matrix

Fit = GNB1
predict = Fit.predict(X_test)
ConfusionM = confusion_matrix(y_test,predict)
print(ConfusionM)

sns.heatmap(ConfusionM,cmap="Greens",annot=True,cbar_kws={"orientation":"horizontal"})
```

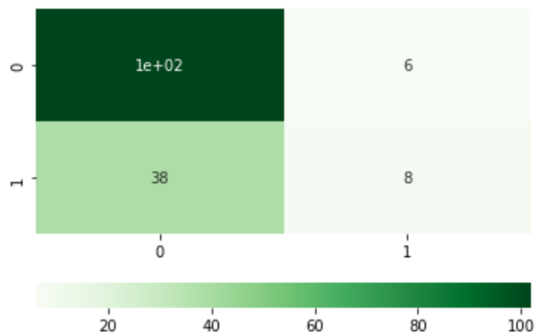
```
[[98 10]
 [35 11]]

<AxesSubplot:>
```



```python
#Confusion Matrix for Logistic Regression Training
import seaborn as sns #Importing for customizing the confusion matrix
from sklearn.metrics import confusion_matrix #Import the confusion Matrix

Fit = LR
predict = Fit.predict(X_test)
ConfusionM = confusion_matrix(y_test,predict)
print(ConfusionM)

sns.heatmap(ConfusionM,cmap="Greens",annot=True,cbar_kws={"orientation":"horizontal"})
```

```
[[102   6]
 [ 38   8]]
```

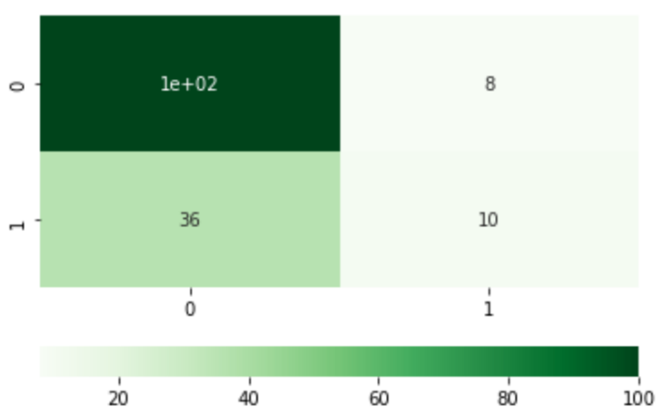
```
<AxesSubplot:>
```




```python
#Confusion Matrix for Logistic Regression Testing
import seaborn as sns #Importing for customizing the confusion matrix
from sklearn.metrics import confusion_matrix #Import the confusion Matrix

Fit = LR1
predict = Fit.predict(X_test)
ConfusionM = confusion_matrix(y_test,predict)
print(ConfusionM)

sns.heatmap(ConfusionM,cmap="Greens",annot=True,cbar_kws={"orientation":"horizontal"})
```

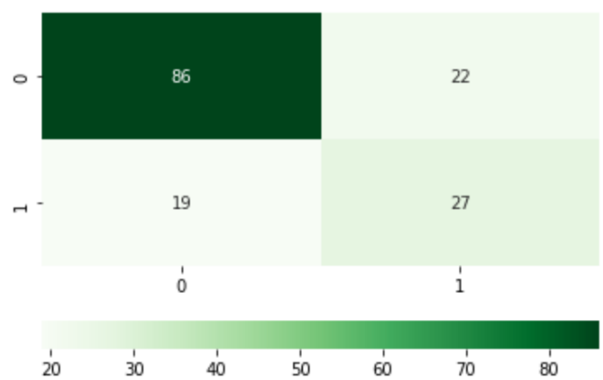
```
[[100   8]
 [ 36  10]]
```

```
<AxesSubplot:>
```



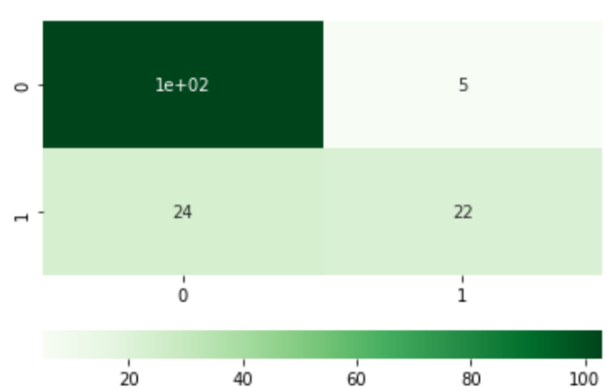
```python
#Confusion Matrix for Decision Tree Training
import seaborn as sns #Importing for customizing the confusion matrix
from sklearn.metrics import confusion_matrix #Import the confusion Matrix

Fit = DtC
predict = Fit.predict(X_test)
ConfusionM = confusion_matrix(y_test,predict)
print(ConfusionM)

sns.heatmap(ConfusionM,cmap="Greens",annot=True,cbar_kws={"orientation":"horizontal"})
```

```
[[86 22]
 [19 27]]

<AxesSubplot:>
```




```python
#Confusion Matrix for Decision Tree Testing
import seaborn as sns #Importing for customizing the confusion matrix
from sklearn.metrics import confusion_matrix #Import the confusion Matrix

Fit = DtC1
predict = Fit.predict(X_test)
ConfusionM = confusion_matrix(y_test,predict)
print(ConfusionM)

sns.heatmap(ConfusionM,cmap="Greens",annot=True,cbar_kws={"orientation":"horizontal"})
```

```
[[103   5]
 [ 24  22]]

<AxesSubplot:>
```



```
#Naive Bayes Training Accuracy
GNB.score(X_train, y_train)
```

```
0.6270358306188925
```

```
#Naive Bayes Testing Accuracy
GNB1.score(X_test, y_test)
```

```
0.7077922077922078
```

```
#Logistic Regression Training Accuracy
LR.score(X_train, y_train)
```

```
0.6302931596091205
```

```
#Logistic Regression Testing Accuracy
LR1.score(X_test, y_test)
```

```
0.7142857142857143
```

```
#Decision Tree Training Accuracy
DtC.score(X_train, y_train)
```

```
0.7133550488599348
```

```
#Decision Tree Testing Accuracy
DtC1.score(X_test, y_test)
```

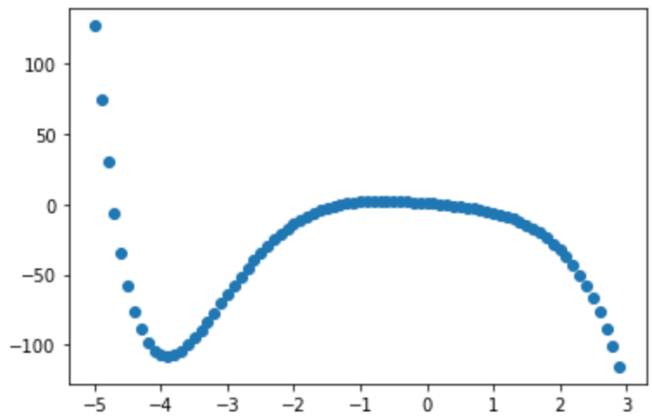
```
0.8116883116883117
```

Question 3:

3.1

```
import pandas as pd #import pandas
import numpy as np #import numpy
from sklearn import datasets, linear_model #import datasets
from matplotlib import pyplot as plt #For use in plotting data

RegressionD = pd.read_csv("C:\\Users\\Eddie\\Desktop\\datasets\\np_regression_dataset2.csv")
```

```
X = np.array(RegressionD['x']) #Fit x values into X
y = np.array(RegressionD['y']) #Fit y values into Y
plt.scatter(X, y)
plt.show()
```



3.2

```
#Training the Neural Network Model
X = np.array(RegressionD['x']).reshape(-1,1) #Fit x values into X
y = np.array(RegressionD['y']) #Fit y values into Y
from sklearn.neural_network import MLPClassifier #Import the MLPclassifier algorithm neccesary for Neural Network modelling
NeuralN = MLPClassifier(solver='lbfgs', alpha=1e-5,hidden_layer_sizes=(5, 2), random_state=1) #Create the Neural Network Model
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)
```

3.3

```
NeuralN.fit(X_train, y_train) #Calculating the coefficient
CoeffR2 = NeuralN.score(X_train, y_train)
print(f"Test set coefficient: {CoeffR2}")
```

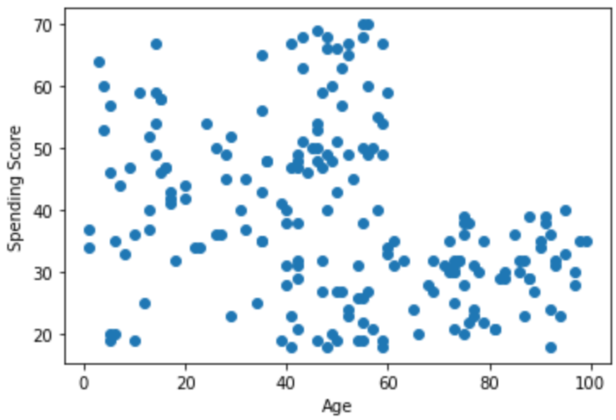
3.4

Question 4:

4.1

```
import pandas as pd #import pandas
import numpy as np #import numpy
from sklearn import datasets, linear_model #import datasets
from matplotlib import pyplot as plt #For use in plotting data

GuckTo = pd.read_csv("C:\\Users\\Eddie\\Desktop\\datasets\\guckto_customers.csv")
```

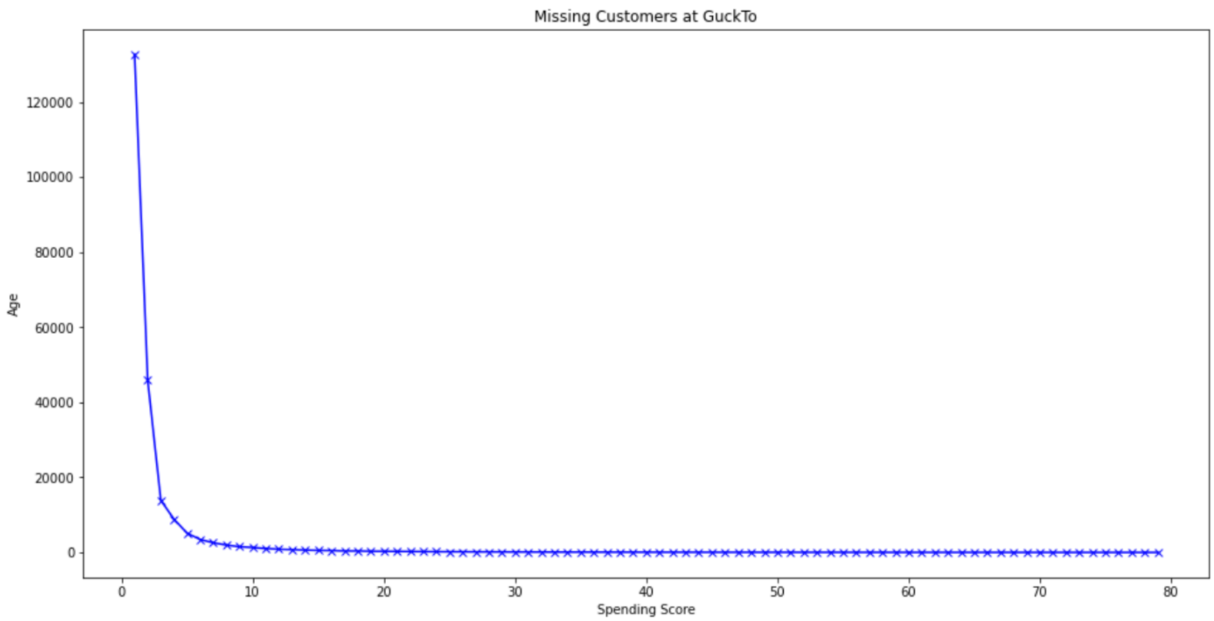
```
X= np.array(GuckTo["Spending Score (1-100)"]) #Assign spending score values to X
y= np.array(GuckTo["Age"]) #Assign Age values to y
plt.scatter(X,y)
plt.xlabel('Age')
plt.ylabel('Spending Score')
plt.show()
```



4.2

```
from sklearn.cluster import KMeans #Import to use kmeans within the program


X= np.array(GuckTo["Spending Score (1-100)"]).reshape(-1, 1) #Reshape to fit the data
distortions = []
K = range(1,80) #Random range for 80 various customers
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X,y)
    distortions.append(kmeanModel.inertia_)
```

```
plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('Spending Score')
plt.ylabel('Age')
plt.title('Missing Customers at GuckTo')
plt.show() #Show the total clusters
```

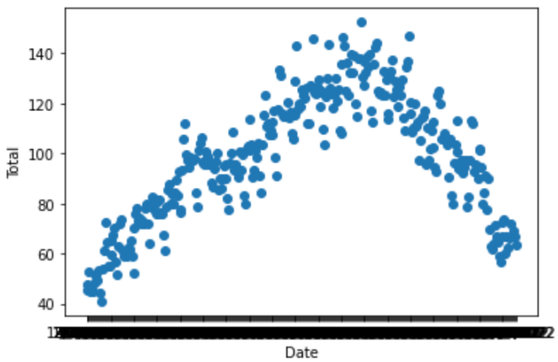


4.3

Question 5:

5.1

```
import pandas as pd #import pandas
import numpy as np #import numpy
from sklearn import datasets, linear_model #import datasets
from matplotlib import pyplot as plt #For use in plotting data
Air_sales = pd.read_csv("C:\\Users\\Eddie\\Desktop\\datasets\\djambo_air_sales_data.csv",parse_dates=True)

X = np.array(Air_sales['Date']) #Fit Date values into X
y = np.array(Air_sales['Total']) #Fit Total revenue values into Y
plt.scatter(X, y)
plt.xlabel('Date')
plt.ylabel('Total')
plt.show()
```



5.2

```
from statsmodels.tsa.stattools import adfuller #For using adfuller
def ad_test(Airdata): #Create a method for calculating the AIC
    dftest = adfuller(Airdata, autolag = 'AIC')
    print("1. Airsales:", dftest[0])
    for key, val in dftest[4].items():
            print("\t", key, ":", val)
```

```
ad_test(Air_sales['Total']) #Test run of the AIC
```

```
1. Airsales: -1.5899031233340124
         1% : -3.45169128009473
         5% : -2.8709394227049154
         10% : -2.5717780602423517
```

```
from pmdarima import auto_arima #Import the ARIMA model
import warnings #Import to suppres warnings
warnings.filterwarnings("ignore")
ArimaFit = auto_arima(Air_sales['Total'], trace=True,suppress_warnings=True)
ArimaFit.summary() #Create a summary and Identify the best model order
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=2312.807, Time=0.26 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=2390.976, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=2385.401, Time=0.03 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=2369.876, Time=0.04 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=2388.987, Time=0.01 sec
 ARIMA(1,1,2)(0,0,0)[0] intercept   : AIC=2328.450, Time=0.09 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=2321.965, Time=0.11 sec
 ARIMA(3,1,2)(0,0,0)[0] intercept   : AIC=inf, Time=0.41 sec
 ARIMA(2,1,3)(0,0,0)[0] intercept   : AIC=2300.372, Time=0.23 sec
 ARIMA(1,1,3)(0,0,0)[0] intercept   : AIC=2319.911, Time=0.14 sec
 ARIMA(3,1,3)(0,0,0)[0] intercept   : AIC=2314.925, Time=0.23 sec
 ARIMA(2,1,4)(0,0,0)[0] intercept   : AIC=inf, Time=0.47 sec
 ARIMA(1,1,4)(0,0,0)[0] intercept   : AIC=2309.763, Time=0.18 sec
 ARIMA(3,1,4)(0,0,0)[0] intercept   : AIC=2216.617, Time=0.58 sec
 ARIMA(4,1,4)(0,0,0)[0] intercept   : AIC=inf, Time=0.53 sec
 ARIMA(3,1,5)(0,0,0)[0] intercept   : AIC=inf, Time=0.55 sec
 ARIMA(2,1,5)(0,0,0)[0] intercept   : AIC=2281.026, Time=0.31 sec
 ARIMA(4,1,3)(0,0,0)[0] intercept   : AIC=2194.936, Time=0.44 sec
 ARIMA(4,1,2)(0,0,0)[0] intercept   : AIC=2261.837, Time=0.37 sec
 ARIMA(5,1,3)(0,0,0)[0] intercept   : AIC=2196.195, Time=0.50 sec
 ARIMA(5,1,2)(0,0,0)[0] intercept   : AIC=2260.479, Time=0.24 sec
 ARIMA(5,1,4)(0,0,0)[0] intercept   : AIC=inf, Time=0.57 sec
 ARIMA(4,1,3)(0,0,0)[0]             : AIC=2193.114, Time=0.34 sec
 ARIMA(3,1,3)(0,0,0)[0]             : AIC=2313.090, Time=0.15 sec
 ARIMA(4,1,2)(0,0,0)[0]             : AIC=2259.954, Time=0.23 sec
 ARIMA(5,1,3)(0,0,0)[0]             : AIC=2193.106, Time=0.40 sec
 ARIMA(5,1,2)(0,0,0)[0]             : AIC=2258.644, Time=0.17 sec
 ARIMA(5,1,4)(0,0,0)[0]             : AIC=inf, Time=0.51 sec
 ARIMA(4,1,4)(0,0,0)[0]             : AIC=inf, Time=0.46 sec

Best model:  ARIMA(5,1,3)(0,0,0)[0]
Total fit time: 8.570 seconds
```

The best model is found to be (5,1,3)

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	324
Model:	SARIMAX(5, 1, 3)	**Log Likelihood**	-1087.553
Date:	Sat, 17 Jun 2023	**AIC**	2193.106
Time:	11:55:13	**BIC**	2227.105
Sample:	0	**HQIC**	2206.678
	- 324		
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	-1.2957	0.133	-9.746	0.000	-1.556	-1.035
ar.L2	-0.6363	0.203	-3.128	0.002	-1.035	-0.238
ar.L3	-0.4552	0.178	-2.550	0.011	-0.805	-0.105
ar.L4	-0.7473	0.134	-5.581	0.000	-1.010	-0.485
ar.L5	-0.2030	0.093	-2.184	0.029	-0.385	-0.021
ma.L1	1.2060	0.125	9.684	0.000	0.962	1.450
ma.L2	0.0616	0.204	0.302	0.763	-0.338	0.461
ma.L3	-0.5084	0.113	-4.480	0.000	-0.731	-0.286
sigma2	49.1281	4.048	12.137	0.000	41.194	57.062

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	1.39
Prob(Q):	0.91	**Prob(JB):**	0.50
Heteroskedasticity (H):	1.74	**Skew:**	0.13
Prob(H) (two-sided):	0.00	**Kurtosis:**	3.20

5.3