Analyzing Model Performance of the online selling from India: Case study on Amazon Sales Report 2022

Executive Summary

This paper emphasizes the use of modeling techniques suitable for analyzing eCommerce data from India using the Amazon Sales Report of 2022, which dataset originally contained 128,975 observations and 24 variables (index, Order ID , Date , Status , Fulfilment, Sales Channel, ship-service-level , Style , SKU, Category, Size, ASIN, Courier Status, Qty, currency, Amount, ship-city, ship-state, ship-postal-code, ship-country, promotion-ids, B2B, fulfilled-by, and Unnamed:22) primarily consists of categorical variables (19). Initially, we conducted exploratory data analysis (EDA) on the dataset without transformation. Subsequently, we applied EDA to the dataset after performing dummy variable transformations. The dataset, sourced from Kaggle, is titled 'Amazon Sales Report' and pertains to online sales from India. The primary objective of this study is to recommend the most suitable modeling technique for such data. Specifically, we aim to compare four models: Multi linear Regression, Logistic Regression, Random Forest, and Decision Tree models . In the multi linear regression model, 'Amount' serves as the dependent variable, predicted by variables such as Index, Status, Fulfilment, Category, Size, Qty, and ship-postal-code. Conversely, in the logistic regression , random forest, and decision tree models, the binary dependent variable 'Fulfilment' is predicted by Index, Status, Category, Size, Qty, Amount, and ship-postal-code.

Introduction

The growth of the ecommerce industry is closely tied to online selling and internet availability, which remain pivotal factors globally. Despite significant efforts to improve internet accessibility worldwide, disparities persist between developed and developing nations. Developed countries typically boast internet penetration rates exceeding 80%, whereas in poorer countries, this figure may be less than 10%. As of 2022, India has achieved a commendable internet penetration rate of over 60% for its population exceeding 1.4 billion[1].

The potential for online selling in India is evident, as highlighted by Forbes Advisor's report on "E-Commerce Statistics for India in 2024". According to their findings, India's ecommerce sector is projected to grow to INR 4,416.68 billion by 2024, with an annual growth rate of 11.45%, reaching INR 7,591.94 billion by 2029. By 2029, the number of ecommerce users in India is expected to reach 501.6 million, with user penetration increasing from 22.1% in 2024 to 34.0% by 2029[2].

Despite India's lower internet penetration compared to developed nations, its substantial online user base forecasted for 2024 underscores its potential as a lucrative market. This study aims to leverage predictive modeling, specifically comparing multi linear regression, logistic regression, random forest, and decision tree models. The objective is to determine which of these models can provide the most insightful analysis of the dataset, facilitating informed decision-making for product development and market strategy implementation.

[1] Retrieved from https://en.wikipedia.org/wiki/List_of_countries_by_number_of_Internet_users
[2] Retrieved from https://www.forbes.com/advisor/in/business/ecommerce-statistics/#:~:text=India's%20e%2Dcommerce%20sector%20is,expected%20to%20reach%20501.6%20million.

Data Collection, Preparation , and Visualization

- Data Collection

The data utilized in this study were sourced from Kaggle and are described as follows: the 19 character variables listed in table1, 4 numerical variables, and 1 Boolean variable.

Table1: List of the character variables and their levels

	VarName	LevelsCount
0	Order ID	120378
1	Date	91
2	Status	13
3	Fulfilment	2
4	Sales Channel	2
5	ship-service-level	2
6	Style	1377
7	SKU	7195
8	Category	9
9	Size	11
10	ASIN	7190
11	Courier Status	3
12	currency	1
13	ship-city	8955
14	ship-state	69
15	ship-country	1
16	promotion-ids	5787
17	fulfilled-by	1
18	Unnamed: 22	1

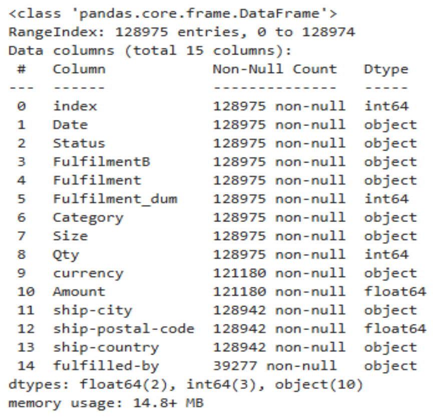
Table2: Descriptive statistic for the numerical variables

	Index	Qty	Amount	Ship-postal-code
count	128975.000000	128975.000000	121180.000000	128942.000000
mean	64487.000000	0.904431	648.561465	463966.236509
std	37232.019822	0.313354	281.211687	191476.764941
min	0.000000	0.000000	0.000000	110001.000000
25%	32243.500000	1.000000	449.000000	382421.000000
50%	64487.000000	1.000000	605.000000	500033.000000
75%	96730.500000	1.000000	788.000000	600024.000000
max	128974.000000	15.000000	5584.000000	989898.000000

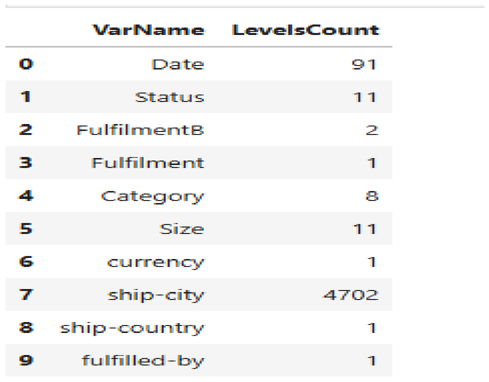
- ## Data Preparation

The original dataset 'AmazonSale' initially consisted of 24 variables and 128,975 observations, labeled as data11. Upon inspection for missing values using the function (data11.dropna()), the dataset was reduced to 19,379 observations while retaining the same number of variables. Before conducting exploratory data analysis, we removed unnecessary variables pertinent to online selling in India. Variables such as currency, ship-state, ship-city, Order Id, Date, Style, SKU, ASIN, and promotion-ids were dropped due to issues with creating dummy variables and memory constraints, especially those with levels exceeding 90. Additionally, variables like Sales channel, Courier Status, ship-service-level, fulfilled-by, and Unnamed were irrelevant to the study's objectives and were also dropped. It's important to note that the variable "Sales Channel" presented difficulties in dropping using the function and was consequently removed using Excel, resulting in data11 having 19,379 observations with 23 variables. Following the removal of unused variables, the dataset was left with 19,379 observations and 8 variables. Before proceeding with dummy variable transformation, we converted the content of the variable "Fulfilment" to numeric values (Amazon to 1, Merchant to 0) since it served as one of the dependent variables for logistic regression, random forest, and decision tree tests.

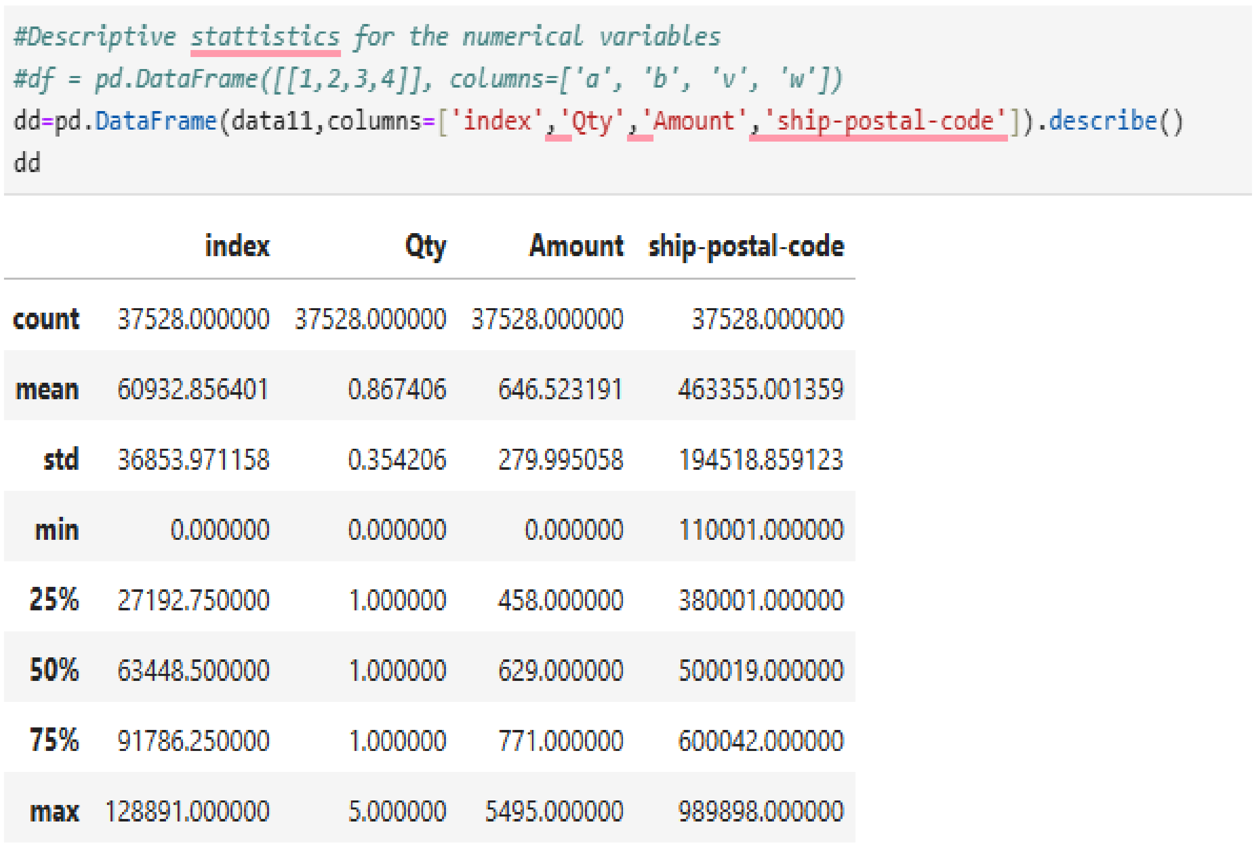
- Data Visualization



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128975 entries, 0 to 128974
Data columns (total 15 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   index               128975 non-null  int64  
 1   Date                128975 non-null  object 
 2   Status              128975 non-null  object 
 3   FulfilmentB         128975 non-null  object 
 4   Fulfilment          128975 non-null  object 
 5   Fulfilment_dum      128975 non-null  int64  
 6   Category            128975 non-null  object 
 7   Size                128975 non-null  object 
 8   Qty                 128975 non-null  int64  
 9   currency            121180 non-null  object 
 10  Amount              121180 non-null  float64
 11  ship-city           128942 non-null  object 
 12  ship-postal-code    128942 non-null  float64
 13  ship-country        128942 non-null  object 
 14  fulfilled-by        39277 non-null   object 
dtypes: float64(2), int64(3), object(10)
memory usage: 14.8+ MB
```



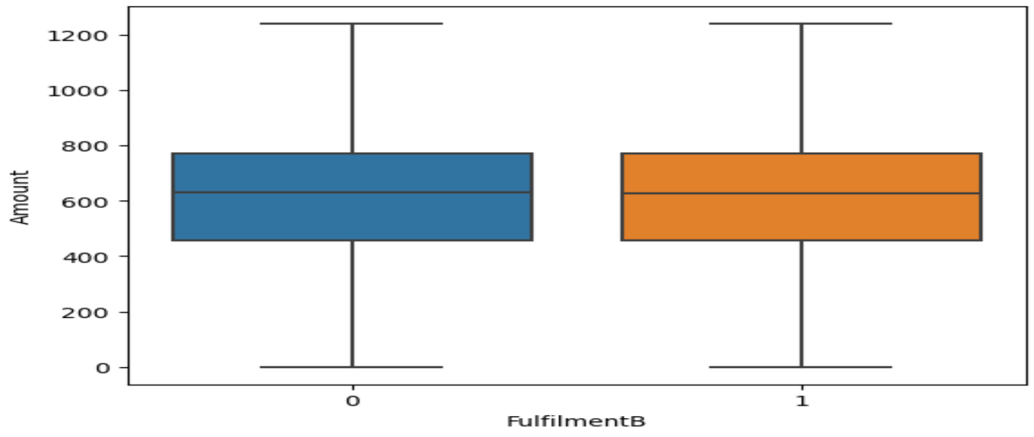
	VarName	LevelsCount
0	Date	91
1	Status	11
2	FulfilmentB	2
3	Fulfilment	1
4	Category	8
5	Size	11
6	currency	1
7	ship-city	4702
8	ship-country	1
9	fulfilled-by	1



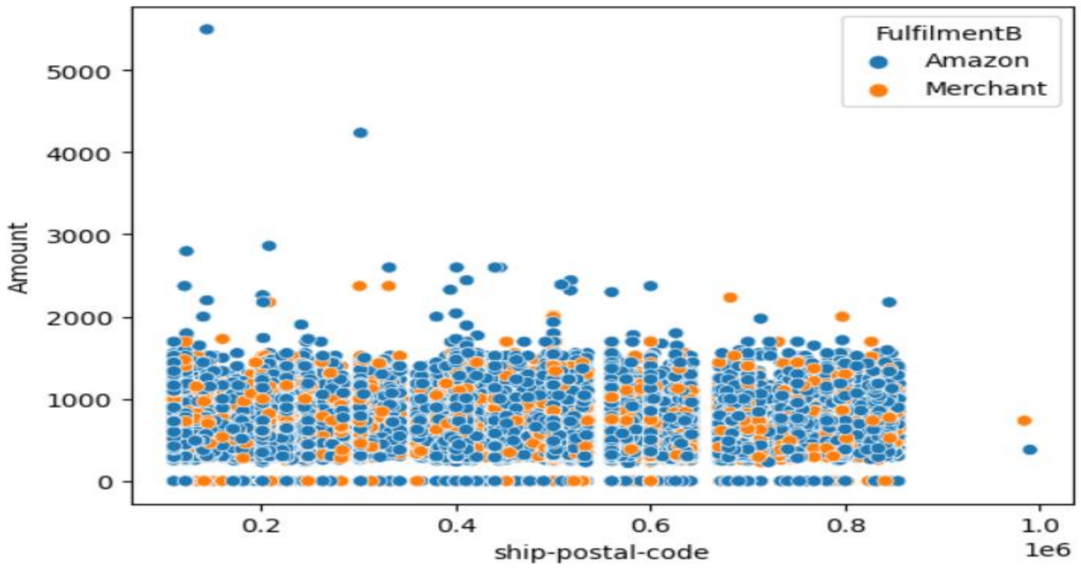
```
#Descriptive stattistics for the numerical variables
#df = pd.DataFrame([[1,2,3,4]], columns=['a', 'b', 'v', 'w'])
dd=pd.DataFrame(data11,columns=['index','Qty','Amount','ship-postal-code']).describe()
dd
```

	index	Qty	Amount	ship-postal-code
count	37528.000000	37528.000000	37528.000000	37528.000000
mean	60932.856401	0.867406	646.523191	463355.001359
std	36853.971158	0.354206	279.995058	194518.859123
min	0.000000	0.000000	0.000000	110001.000000
25%	27192.750000	1.000000	458.000000	380001.000000
50%	63448.500000	1.000000	629.000000	500019.000000
75%	91786.250000	1.000000	771.000000	600042.000000
max	128891.000000	5.000000	5495.000000	989898.000000

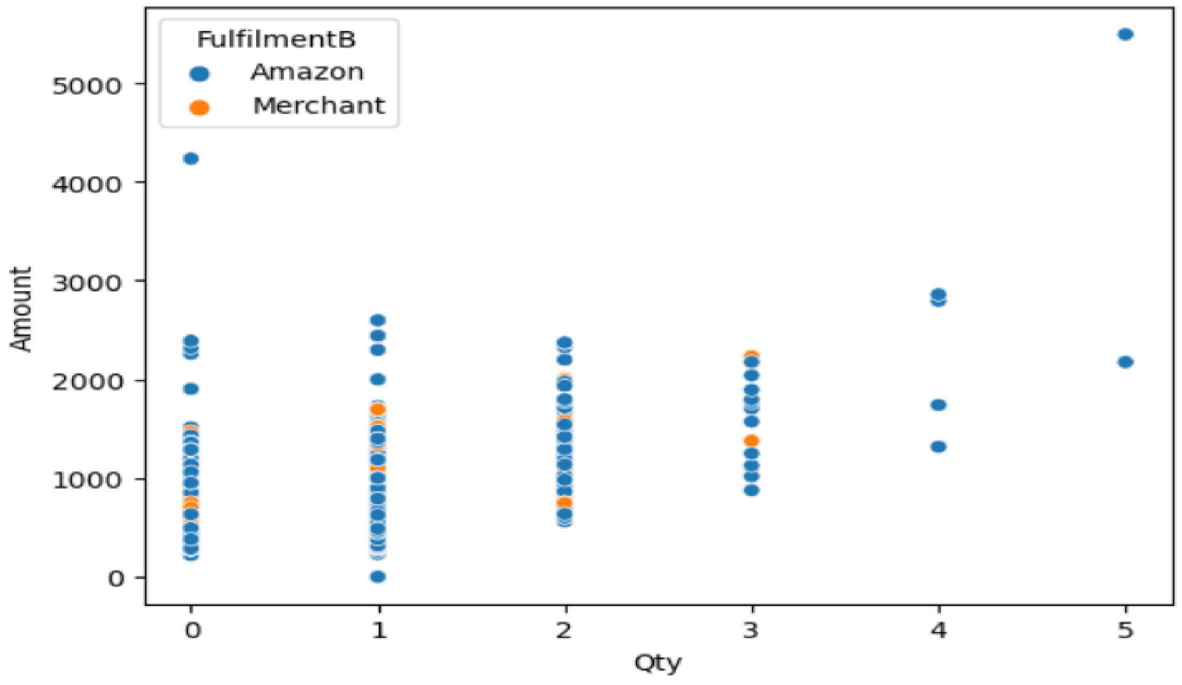
```python
# Making a scatter plot for FulfilmentB after being transformed to Amazon=1 and Merchant=0
_ = sns.boxplot(x='FulfilmentB', y='Amount', data=data11, showfliers=False)
```



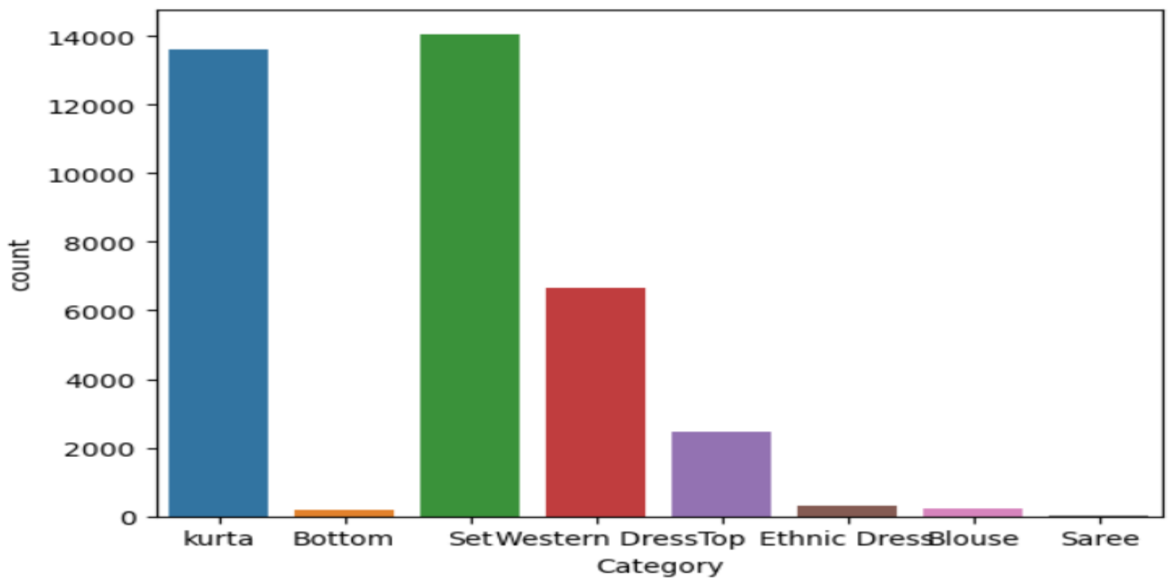
```python
# Make a scatterplot with ship-postal-code as x, Amount as y with regards to FulfilmentB
_ = sns.scatterplot(x='ship-postal-code', y='Amount', data=data11, hue='FulfilmentB')
```



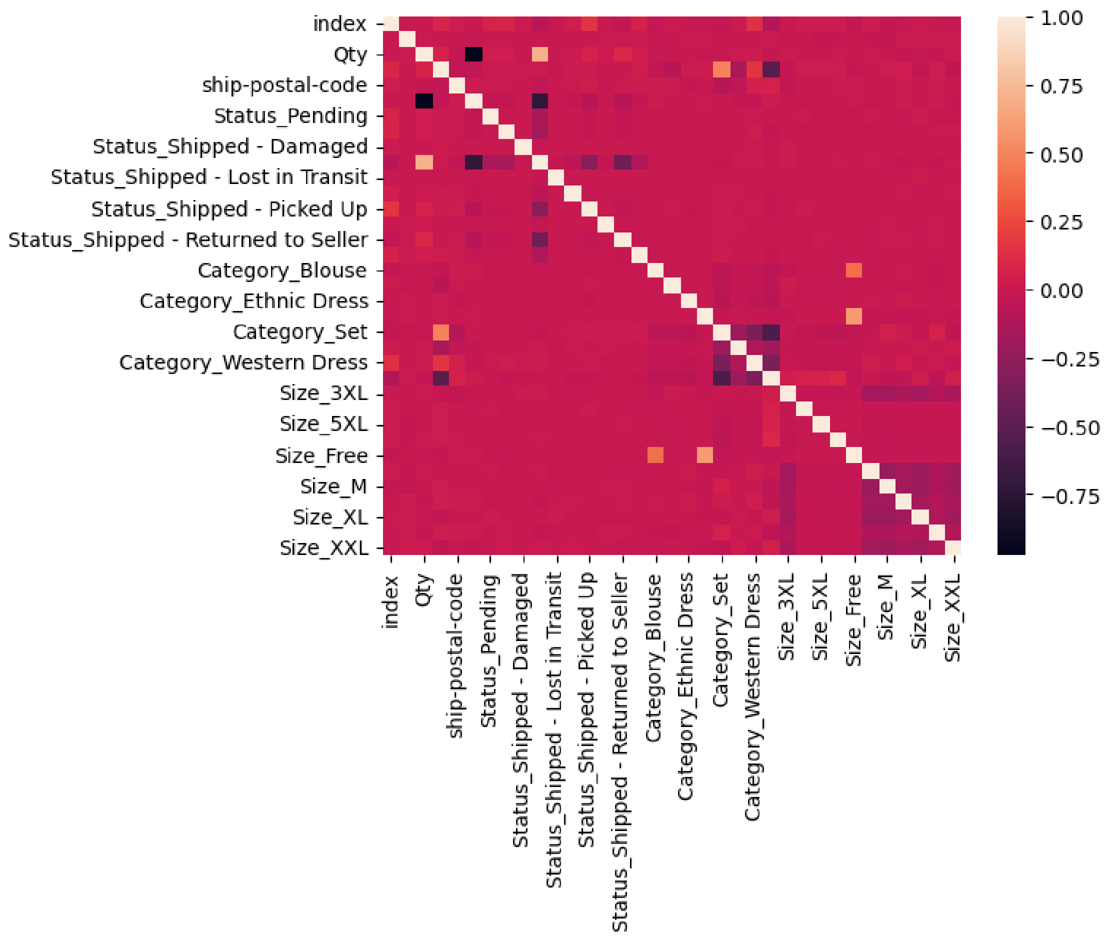
```
# Make a scatterplot with Qty as x, Amount as Y with regards to FulfilmentB
_ = sns.scatterplot(x='Qty', y='Amount', data=data11, hue='FulfilmentB')
```



```
ax=sns.countplot(x='Category' , data=data11)
```



```
# Make a heatmap of the data with the  created dummies variables
correlation_matrix = data11.corr()
_ = sns.heatmap(correlation_matrix)
```



Methodology

In this study, the data was divided into a 70% training set and a 30% test set for all four models, with a distinction in the dependent variable used. The multi linear model predicted 'Amount' as the dependent variable, while 'FulfilmentB', a dummy variable created from 'Fulfilment' (labeled as Fulfilment: Amazon and Fulfilment: Merchant), was used in logistic regression, random forest, and decision tree models. For the multi linear model, 'Amount' was predicted using predictors such as Index, Status, Fulfilment, Category, Size, Qty, and ship-postal-code. Conversely, the logistic regression, random forest, and decision tree models predicted

'FulfilmentB' as the dependent variable. The predictors for these models included Index, Status, Category, Size, Qty, Amount, and ship-postal-code. This approach allowed us to analyze and compare the performance of different models in predicting outcomes relevant to online selling in India. Furthermore, using "Amount" as the dependent variable allows us to gain a better understanding of profitability trends in online selling in India, aiding in making informed business implementation decisions. Similarly, selecting "Fulfilment" as another dependent variable enables us to assess whether products fulfilled by Amazon or Merchant are more likely to be returned or canceled. This analysis provides valuable insights into operational efficiency and customer satisfaction within the ecommerce sector.

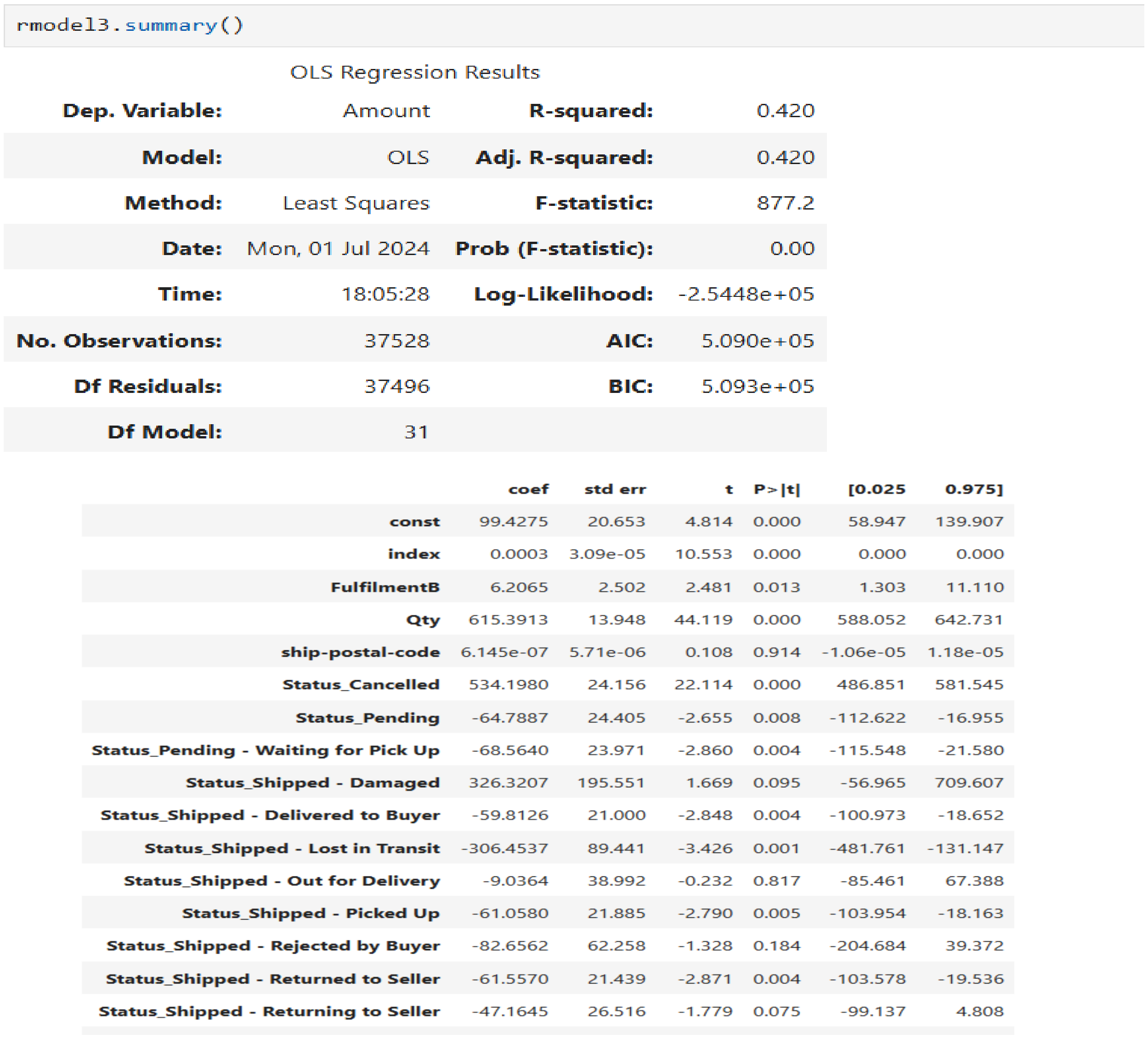
Modeling

Modeling involves translating conceptual ideas into a quantifiable framework with the goal of establishing a general rule to explain a phenomenon. Models serve various purposes, from elucidating complex data to proposing hypotheses. In scientific contexts, models are often developed to predict outcomes based on interactions within cycles of data. As stated on sciencelearn.com, scientists use these models to anticipate future scenarios. In our study, the response variables of interest are "Amount," modeled using multiple linear regression with 7 predictors, and "FulfilmentB," utilized in logistic regression, random forest, and decision tree models with the same set of predictors. Mathematically, the models are described in this following:

Model1) Amount : Index, Status, Fulfilment, Category, Size, Qty, and ship-postal-code

Model2,3& 4) FulfilmentB: Index, Status, Amount, Category, Size, Qty, and ship-postal-code

- Modeling with the multi linear model

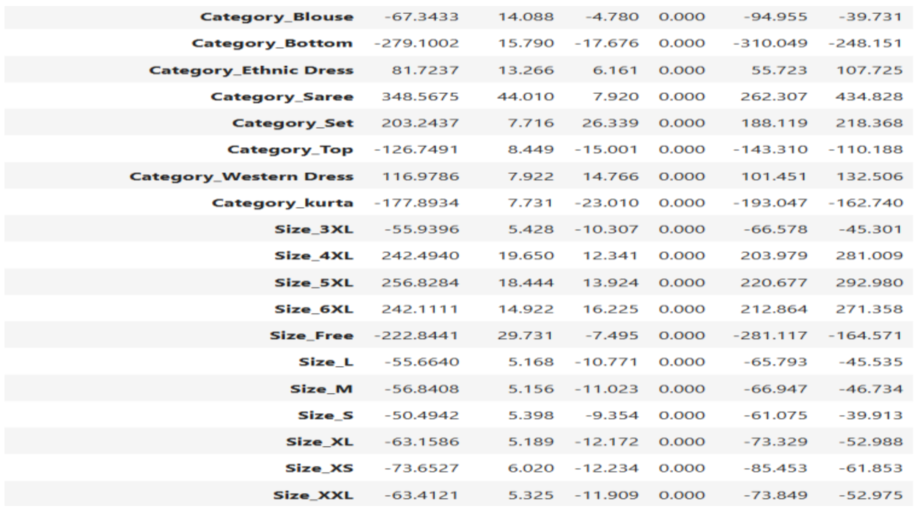


```
rmodel3.summary()
```

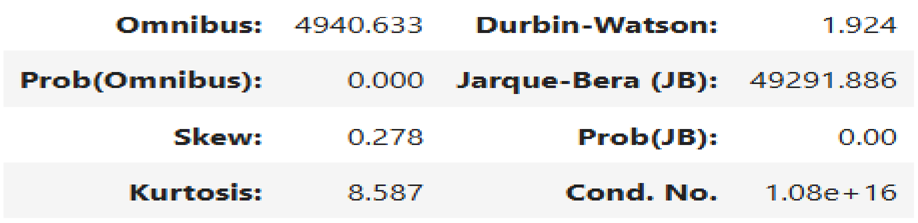
OLS Regression Results

Dep. Variable:	Amount	**R-squared:**	0.420
Model:	OLS	**Adj. R-squared:**	0.420
Method:	Least Squares	**F-statistic:**	877.2
Date:	Mon, 01 Jul 2024	**Prob (F-statistic):**	0.00
Time:	18:05:28	**Log-Likelihood:**	-2.5448e+05
No. Observations:	37528	**AIC:**	5.090e+05
Df Residuals:	37496	**BIC:**	5.093e+05
Df Model:	31		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	99.4275	20.653	4.814	0.000	58.947	139.907
index	0.0003	3.09e-05	10.553	0.000	0.000	0.000
FulfilmentB	6.2065	2.502	2.481	0.013	1.303	11.110
Qty	615.3913	13.948	44.119	0.000	588.052	642.731
ship-postal-code	6.145e-07	5.71e-06	0.108	0.914	-1.06e-05	1.18e-05
Status_Cancelled	534.1980	24.156	22.114	0.000	486.851	581.545
Status_Pending	-64.7887	24.405	-2.655	0.008	-112.622	-16.955
Status_Pending - Waiting for Pick Up	-68.5640	23.971	-2.860	0.004	-115.548	-21.580
Status_Shipped - Damaged	326.3207	195.551	1.669	0.095	-56.965	709.607
Status_Shipped - Delivered to Buyer	-59.8126	21.000	-2.848	0.004	-100.973	-18.652
Status_Shipped - Lost in Transit	-306.4537	89.441	-3.426	0.001	-481.761	-131.147
Status_Shipped - Out for Delivery	-9.0364	38.992	-0.232	0.817	-85.461	67.388
Status_Shipped - Picked Up	-61.0580	21.885	-2.790	0.005	-103.954	-18.163
Status_Shipped - Rejected by Buyer	-82.6562	62.258	-1.328	0.184	-204.684	39.372
Status_Shipped - Returned to Seller	-61.5570	21.439	-2.871	0.004	-103.578	-19.536
Status_Shipped - Returning to Seller	-47.1645	26.516	-1.779	0.075	-99.137	4.808



Category_Blouse	-67.3433	14.088	-4.780	0.000	-94.955	-39.731
Category_Bottom	-279.1002	15.790	-17.676	0.000	-310.049	-248.151
Category_Ethnic Dress	81.7237	13.266	6.161	0.000	55.723	107.725
Category_Saree	348.5675	44.010	7.920	0.000	262.307	434.828
Category_Set	203.2437	7.716	26.339	0.000	188.119	218.368
Category_Top	-126.7491	8.449	-15.001	0.000	-143.310	-110.188
Category_Western Dress	116.9786	7.922	14.766	0.000	101.451	132.506
Category_kurta	-177.8934	7.731	-23.010	0.000	-193.047	-162.740
Size_3XL	-55.9396	5.428	-10.307	0.000	-66.578	-45.301
Size_4XL	242.4940	19.650	12.341	0.000	203.979	281.009
Size_5XL	256.8284	18.444	13.924	0.000	220.677	292.980
Size_6XL	242.1111	14.922	16.225	0.000	212.864	271.358
Size_Free	-222.8441	29.731	-7.495	0.000	-281.117	-164.571
Size_L	-55.6640	5.168	-10.771	0.000	-65.793	-45.535
Size_M	-56.8408	5.156	-11.023	0.000	-66.947	-46.734
Size_S	-50.4942	5.398	-9.354	0.000	-61.075	-39.913
Size_XL	-63.1586	5.189	-12.172	0.000	-73.329	-52.988
Size_XS	-73.6527	6.020	-12.234	0.000	-85.453	-61.853
Size_XXL	-63.4121	5.325	-11.909	0.000	-73.849	-52.975

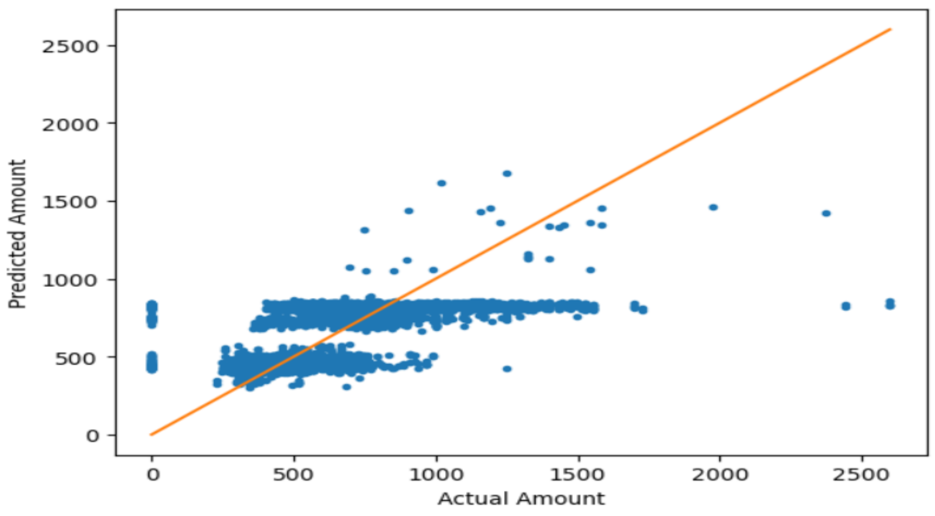


Omnibus:	4940.633	**Durbin-Watson:**	1.924
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	49291.886
Skew:	0.278	**Prob(JB):**	0.00
Kurtosis:	8.587	**Cond. No.**	1.08e+16

The R-squared coefficient above indicates that the model has poorly predicted the variation in the dataset, with an R-squared of 42%, suggesting that more work is needed to improve its predictive power. However, the model exhibits acceptable independence of errors, as indicated by a Durbin-Watson statistic of 1.92. Now, let’s plot the predictions for further details.

Plot the predictions

```
Text(0, 0.5, 'Predicted Amount')
```



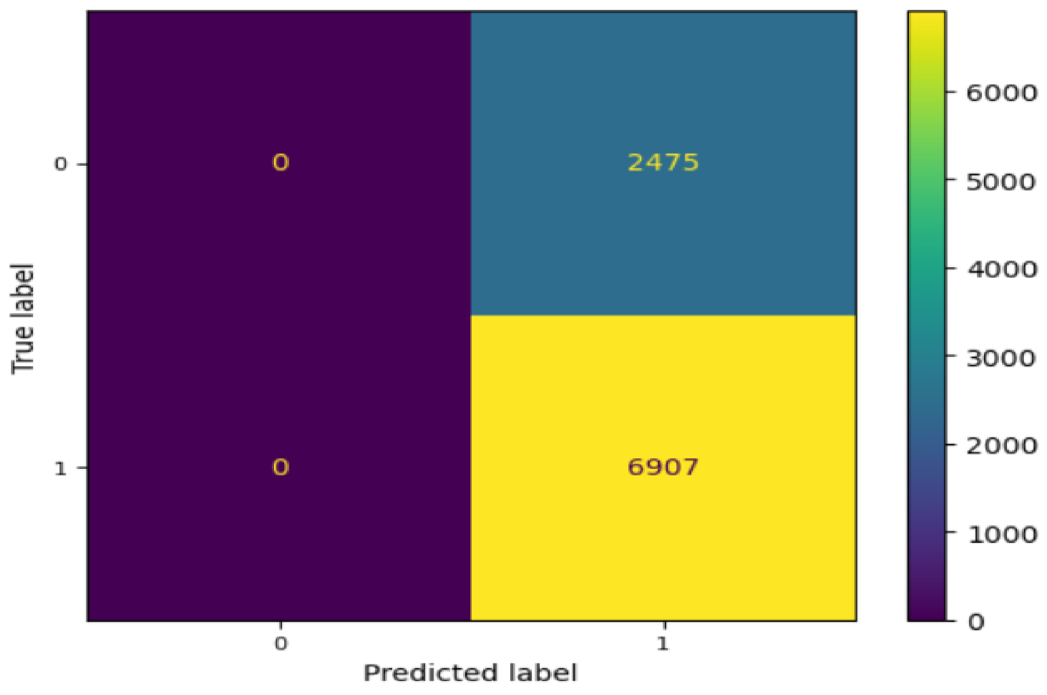
Based on this graph, it seems that the multi linear model is not a good fit for the data. Next, let's assess the data using a random forest model.

- Modeling with Logistic regression

```
y_predict_test = clf.predict(Xtestlrstrat)
y_predict_training = clf.predict(Xlrstrat)
print("[Test] Accuracy score: (ytestlrstrat, y_predict_test) [**note reversed order]",
      f'{accuracy_score(ytestlrstrat, y_predict_test):.2f}')

print("[Test] Accuracy score (y_predict_test, ytestlrstrat):",
      f'{accuracy_score(y_predict_test, ytestlrstrat):.2f}')
print("[Training] Accuracy score: (ylrstrat, y_predict_training)",
      f'{accuracy_score(ylrstrat, y_predict_training):.2f}')
```

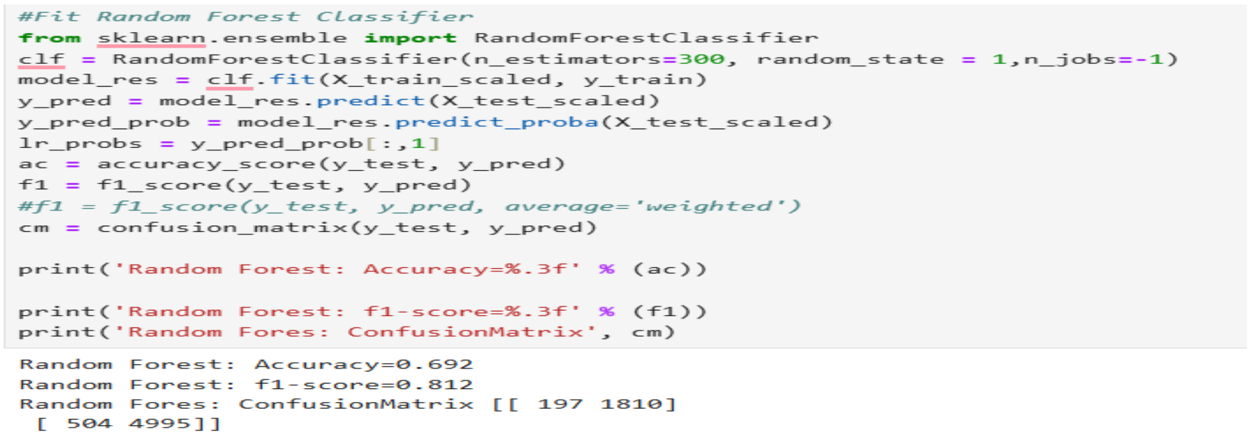
```
[Test] Accuracy score: (ytestlrstrat, y_predict_test) [**note reversed order] 0.74
[Test] Accuracy score (y_predict_test, ytestlrstrat): 0.74
[Training] Accuracy score: (ylrstrat, y_predict_training) 0.74
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x18b2eff8c10>
```



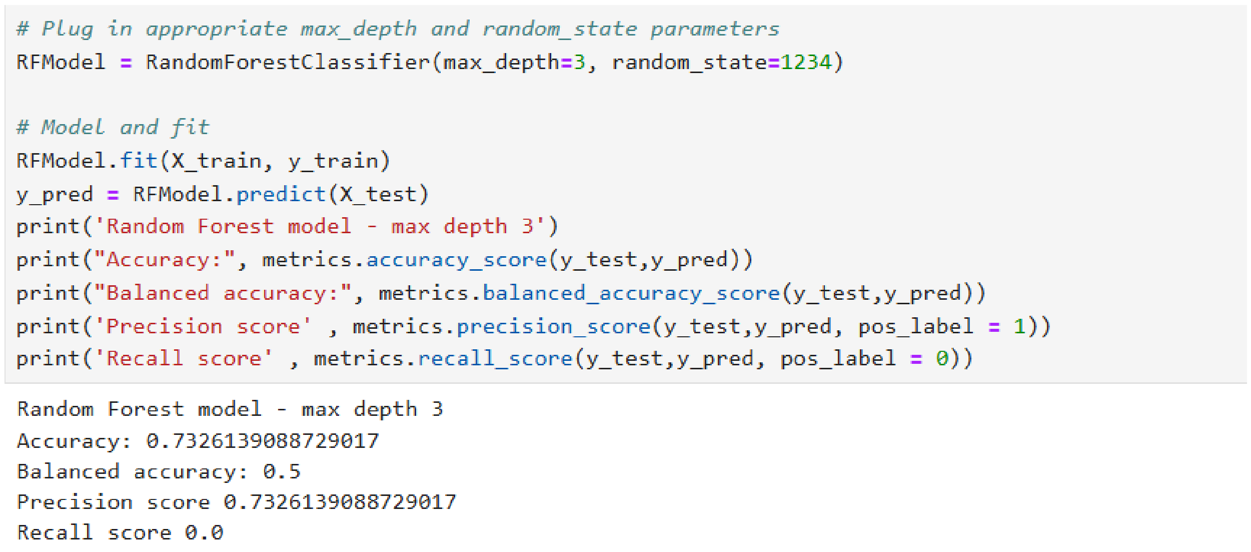
The confusion matrix reveals that for the class [FulfilmentB: Amazon = 1: True Positive: 6907 out of 9382 predictions, which corresponds to 74% While for the class FulfilmentB: Merchant = 0 %. False Negative: 2475 out of 9382 predictions, which accounts for 26%]. However, there were no instances of True Negative or False Positive predictions recorded in the matrix, both totaling 0 out of 9382 predicted. Based on the imbalance revealed in the confusion matrix, where there are no True Negatives or False Positives, there isn't enough evidence to conclusively determine that it is the best fit for the data. Further testing is necessary before determining its significance relative to the previous model. Now, let's proceed to evaluate the dataset using Random Forest.

- Modeling with the Random Forest model



```
#Fit Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=300, random_state = 1,n_jobs=-1)
model_res = clf.fit(X_train_scaled, y_train)
y_pred = model_res.predict(X_test_scaled)
y_pred_prob = model_res.predict_proba(X_test_scaled)
lr_probs = y_pred_prob[:,1]
ac = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
#f1 = f1_score(y_test, y_pred, average='weighted')
cm = confusion_matrix(y_test, y_pred)

print('Random Forest: Accuracy=%.3f' % (ac))

print('Random Forest: f1-score=%.3f' % (f1))
print('Random Fores: ConfusionMatrix', cm)
```

```
Random Forest: Accuracy=0.692
Random Forest: f1-score=0.812
Random Fores: ConfusionMatrix [[ 197 1810]
 [ 504 4995]]
```

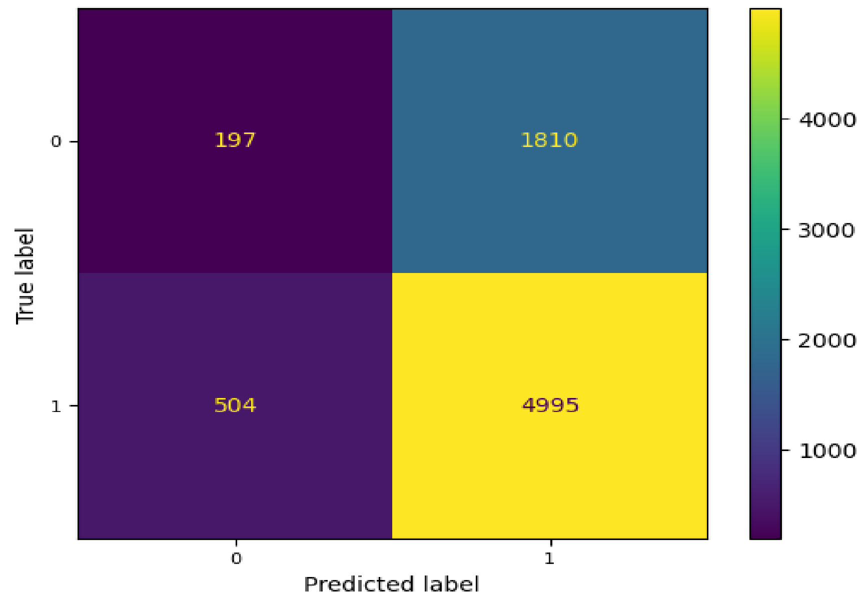


```
# Plug in appropriate max_depth and random_state parameters
RFModel = RandomForestClassifier(max_depth=3, random_state=1234)

# Model and fit
RFModel.fit(X_train, y_train)
y_pred = RFModel.predict(X_test)
print('Random Forest model - max depth 3')
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score' , metrics.precision_score(y_test,y_pred, pos_label = 1))
print('Recall score' , metrics.recall_score(y_test,y_pred, pos_label = 0))
```

```
Random Forest model - max depth 3
Accuracy: 0.7326139088729017
Balanced accuracy: 0.5
Precision score 0.7326139088729017
Recall score 0.0
```

```python
# Let's create the confusion matrix without max depht 3

from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred1, labels = clf.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = [0, 1])
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = [1, 0], fontsize = 8)
ax.set_yticklabels(labels = [0, 1], fontsize = 8)
display_cm.plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x27133aa6950>
```



Despite the drop in R-squared from 100% to 73.70% when limiting the maximum depth to 3, the Random Forest model still demonstrates a significant improvement compared to the Logistic Regression model. Now, let's examine the accuracy of the Random Forest model by examining its predictions through a confusion matrix. The results from this confusion matrix indicate a noticeable improvement in the model's classification compared to the Logistic Regression model. The confusion matrix from the Random Forest demonstrates a clearer understanding of the data's classification. Unlike the Logistic Regression model, which showed an unbalanced confusion matrix, this model shows a more balanced and accurate classification.

For example, the confusion matrix reveals that for the class FulfilmentB: Amazon = 1:

- True Positive 4995 out of 6805 predictions, which corresponds to 73%

- False Negative: 1810 out of 6805 predictions, which corresponds to 27%

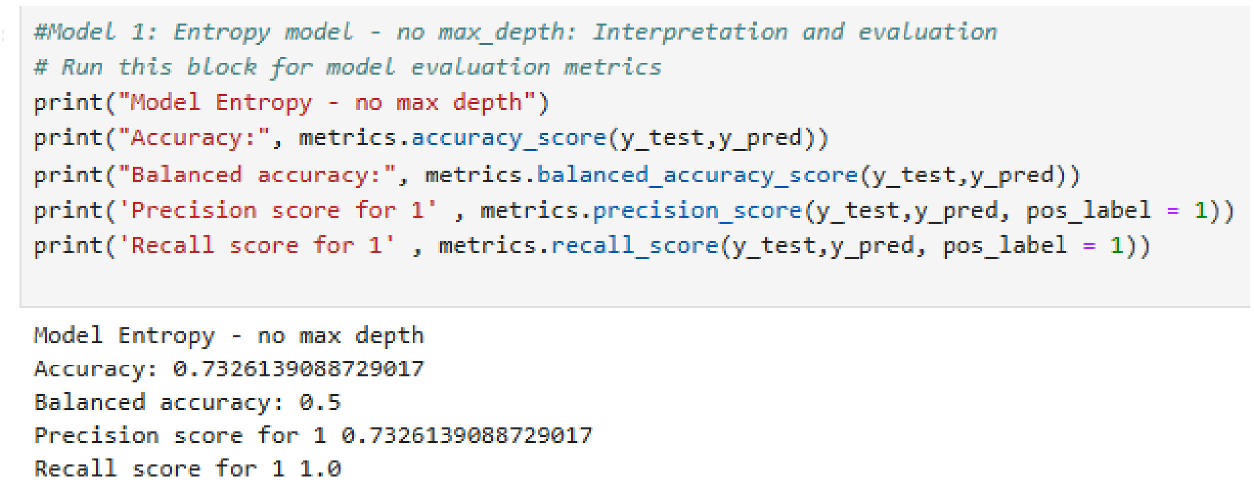
While for the class FulfilmentB: Merchant = 0.

- True se Negative 197 out 701 predictions, which accounts for 28%

- False Positive: 504 out of 701 predictions, which accounts for 72%. Now, let's proceed to evaluate the dataset using the Decision Tree.

- Modeling with Decision Tree

  a. *Modeling using Entropy*

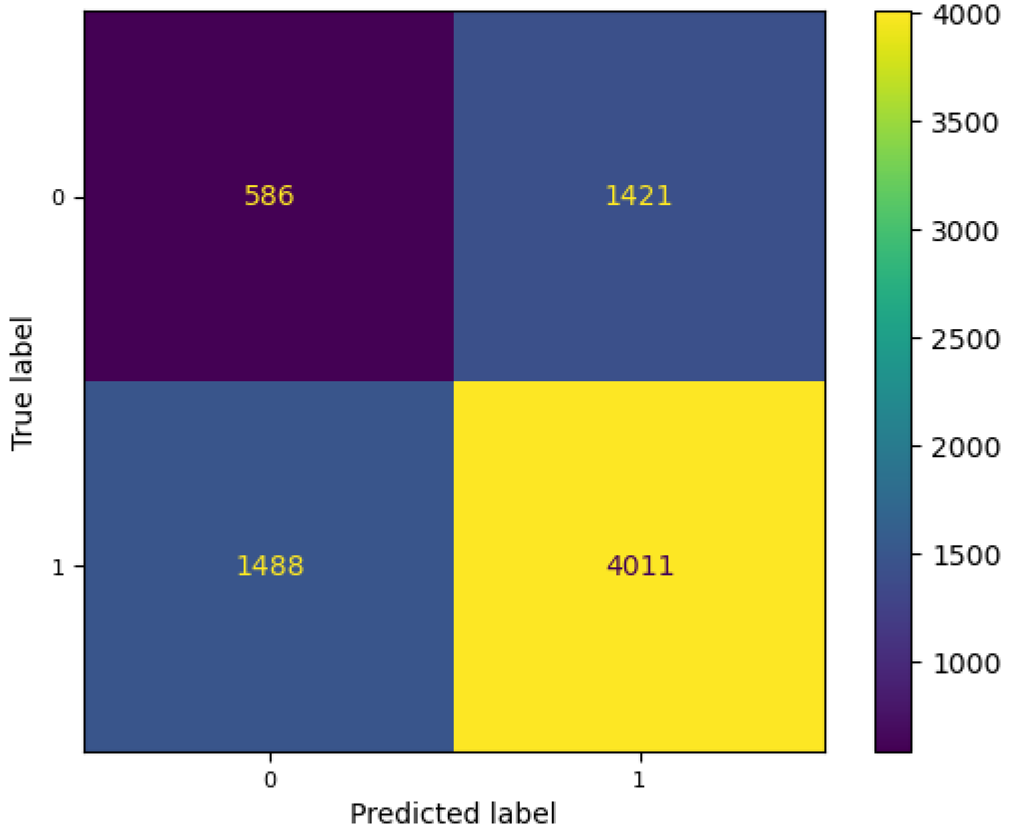


```
#Model 1: Entropy model - no max_depth: Interpretation and evaluation
# Run this block for model evaluation metrics
print("Model Entropy - no max depth")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score for 1' , metrics.precision_score(y_test,y_pred, pos_label = 1))
print('Recall score for 1' , metrics.recall_score(y_test,y_pred, pos_label = 1))
```

```
Model Entropy - no max depth
Accuracy: 0.7326139088729017
Balanced accuracy: 0.5
Precision score for 1 0.7326139088729017
Recall score for 1 1.0
```

```python
# Let's create the confusion matrix for Entropy without max depht 3

from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred1, labels = clf.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = [0, 1])
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = [1, 0], fontsize = 8)
ax.set_yticklabels(labels = [0, 1], fontsize = 8)
display_cm.plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x16605f56390>
```



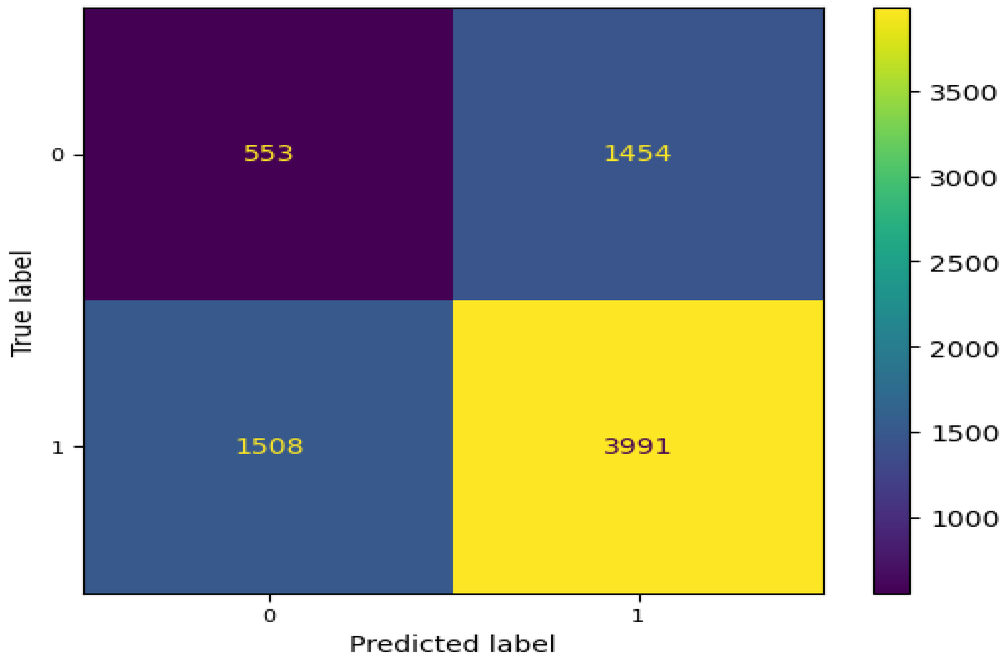
b. Modeling with Gini Impurity with no max depth 3.

```
# Run this block for model evaluation
print("Model Gini impurity model")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score' , metrics.precision_score(y_test,y_pred, pos_label = 1))
print('Recall score' , metrics.recall_score(y_test,y_pred, pos_label = 0))
```

```
Model Gini impurity model
Accuracy: 0.7326139088729017
Balanced accuracy: 0.5
Precision score 0.7326139088729017
Recall score 0.0
```

```
# Let's create the confusion matrix with Gini- no max depth 3

from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred2, labels = clf.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = [0, 1])
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = [1, 0], fontsize = 8)
ax.set_yticklabels(labels = [0, 1], fontsize = 8)
display_cm.plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x16605f54c90>
```



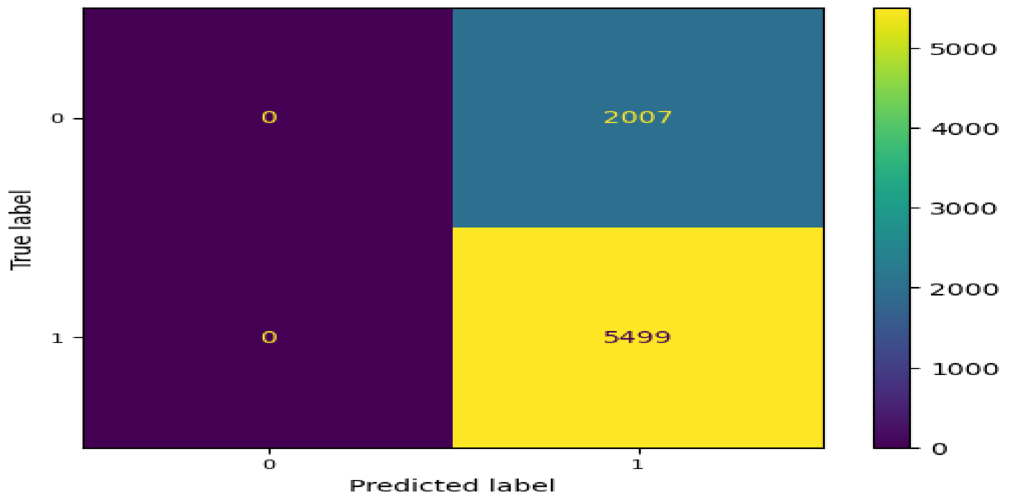
c. Modeling with Entropy model- max depth 3 included

```
# Run this block for model evaluation
print("Model Entropy model max depth 3")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score for 1 ' , metrics.precision_score(y_test,y_pred, pos_label = 1))
print('Recall score for 0' , metrics.recall_score(y_test,y_pred, pos_label = 0))
```

```
Model Entropy model max depth 3
Accuracy: 0.7326139088729017
Balanced accuracy: 0.5
Precision score for 1  0.7326139088729017
Recall score for 0 0.0
```

```
# Let's create the confusion matrix entropy with max depht 3

from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred3, labels = clf.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = [0, 1])
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = [1, 0], fontsize = 8)
ax.set_yticklabels(labels = [0, 1], fontsize = 8)
display_cm.plot(ax = ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x166064b2150>
```



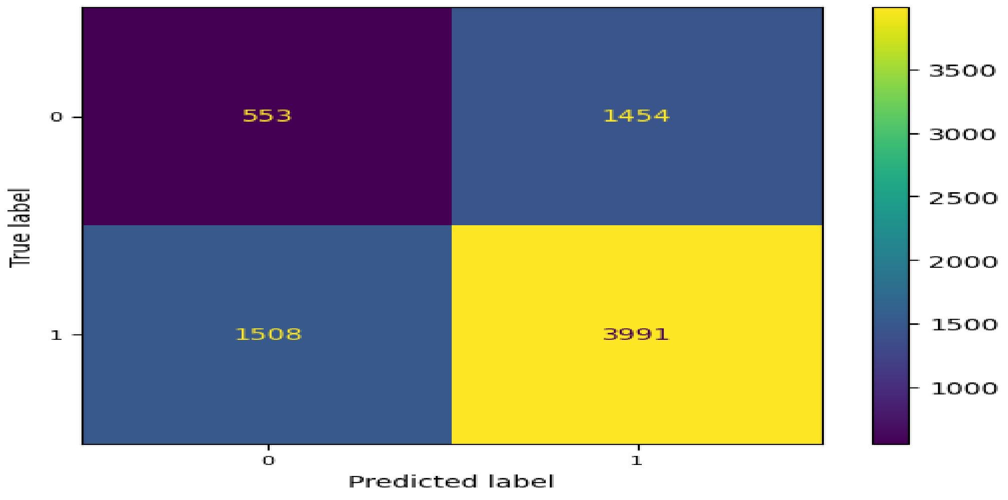
d. Modeling with Gini Impurity with max depth 3

```
print("Gini impurity  model - max depth 3")
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
print("Balanced accuracy:", metrics.balanced_accuracy_score(y_test,y_pred))
print('Precision score' , metrics.precision_score(y_test,y_pred, pos_label = 1 ))
print('Recall score' , metrics.recall_score(y_test,y_pred, pos_label = 0))
```

```
Gini impurity  model - max depth 3
Accuracy: 0.7326139088729017
Balanced accuracy: 0.5
Precision score 0.7326139088729017
Recall score 0.0
```

```
# Let's create the confusion matrix with Gini- no max depth 3

from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred2, labels = clf.classes_)
_, ax = plt.subplots()
display_cm = ConfusionMatrixDisplay(confusion_matrix = cm,
                                    display_labels = [0, 1])
ax.set_xticks([0, 1])
ax.set_yticks([0, 1])
ax.set_xticklabels(labels = [1, 0], fontsize = 8)
ax.set_yticklabels(labels = [0, 1], fontsize = 8)
display_cm.plot(ax = ax)
```

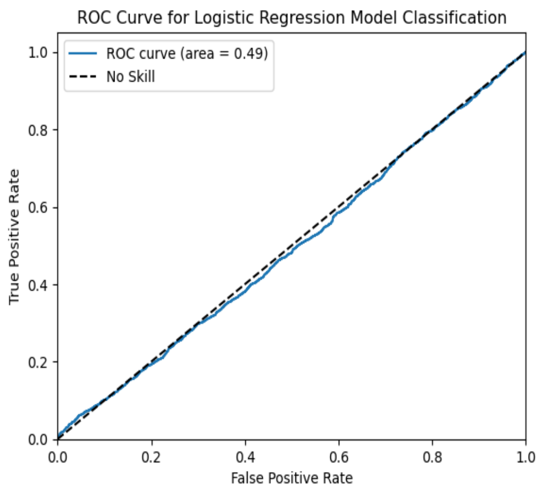
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x16605f54c90>
```

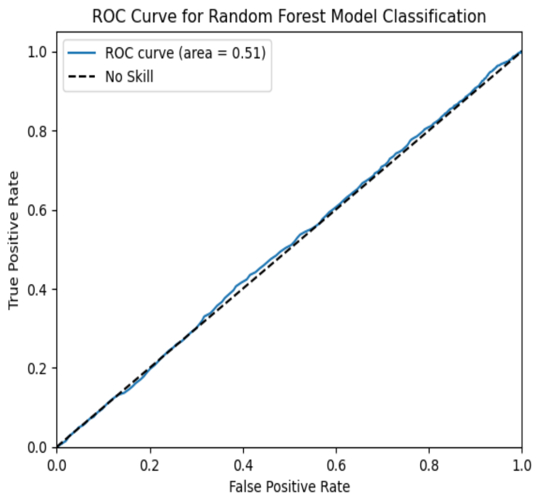


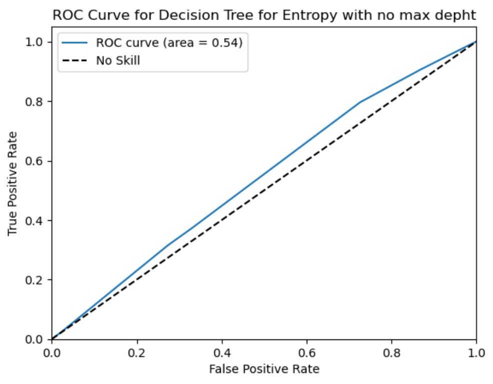
```
# Let's find the th R^2 for the four Decision Tree models.

rr1= aa.score(X_train, y_train)
rr2 = bb.score(X_train, y_train)
rr3 = cc.score(X_train, y_train)
rr4 = dd.score(X_train, y_train)
print(rr1)
print(rr2)
print(rr3)
print(rr4)
```

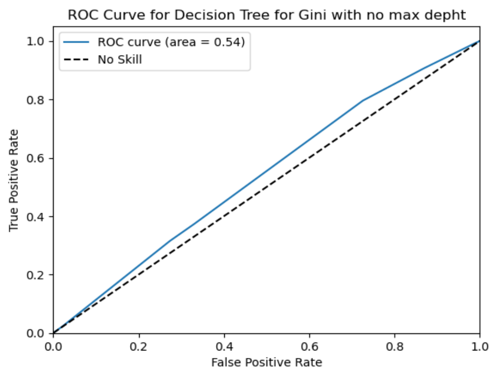
```
1.0
1.0
0.7370594897075478
0.7370594897075478
```

The R-squared values for the four models did not indicate any improvement compared to the Random Forest model, as their values were similar. At this juncture, it was essential to verify the equality of these potential results between these two models by assessing their confusion matrices, as described above. Similarly, the results from the confusion matrix are quite similar to those from the Random Forest, except when using entropy and Gini with a depth of 3, where they yield results similar to the Logistic Regression in terms of unbalanced predictions. Thus far, both the Random Forest and the Decision Tree (using entropy and Gini) seem to be suitable for the data. To determine which model is more advantageous for decision-making, further analysis of their ROC scores was required, which are described below.









Evaluation and Conclusion

Four models were evaluated on the dataset. The multilinear model showed fewer promising results with an R-squared of 42% and an R-score of less than 1%, indicating it is not a good fit for the dataset. This model was used solely for pre-testing, recognizing that its use of continuous variables could pose issues for Logistic Regression, which typically requires a binary dependent variable. Consequently, this model cannot be considered or included in this study. Moving forward, despite the Logistic Regression demonstrating an R-squared of 73.57%, its performance was lacking when assessed using a confusion matrix. Specifically, it only provided results for FulfilmentB: Amazon = 1: True Positive rate of 74% (6907 out of 9382 predictions) and a False Negative rate of 26% (2475 out of 9382 predictions). Notably, there were no instances of True Negatives or False Positives recorded in the matrix, with both categories totaling 0 out of 9382 predicted.

Given the imbalance observed in the Logistic Regression model, further investigation was conducted using the Random Forest Model. In the case of the Random Forest, two models were employed to assess the dataset: Random Forest without a maximum depth limit and Random Forest with a maximum depth of 3. The results indicated a decrease in R-squared from 100% to 73.70% when limiting the maximum depth to 3. However, this still represents a notable improvement compared to the Logistic Regression model. Regarding model accuracy, f1-score, and precision, the results for both tests are described as below:

Random Forest with no max depth
Random Forest: Accuracy=0.692
Random Forest: f1-score=0.81
Random Forest model - max depth 3
Accuracy: 0.7326139088729017
Balanced accuracy: 0.5
Precision score 0.7326139088729017
Recal score T.0

The model (Random Forest with max depth 3) shows a slight improvement in accuracy compared to the version without implementing max depth, which is similar to the results obtained when testing with Logistic Regression. Here are the detailed results for further clarification:

Accuracy score: (ytestlrstrat , y_predict_test) [**note reversed order] :	0.73
Accuracy score : (y_predict_test , ytestlrstrat)	0.73
Accuracy score: (ylrstrat , y_predict_training)	0.74

However, when testing using the confusion matrix, unlike the Logistic Regression model, which displayed an unbalanced confusion matrix, the Random Forest model exhibited a more balanced and accurate classification. Specifically, the specificity was not null, unlike the Logistic Regression model. For instance, the confusion matrix revealed that for the class FulfilmentB: Merchant = 0, the specificity was 28% (197 out of 701 predictions), while for FulfilmentB: Amazon = 1, the sensitivity was 73% (4995 out of 6805 predictions).Despite the promising results from the Random Forest model, we refrained from immediately favoring it over the Logistic Regression model. Instead, we delved deeper into evaluating the Decision Tree model. When testing with the Tree Model, the results demonstrate consistency across various applied tests. Here's a description of the findings:

Model Entropy - no max depth
Accuracy: 0.612443378630429
Balanced accuracy: 0.5106917115790315
Precision score for 1 0.7384020618556701
Precision score for 0 0.28254580520732886
Recall score for 1 0.729405346426623
Recall score for 0 0.29197807673143994

Model Gini impurity model with no max depth 3
Accuracy: 0.6053823607780442
Balanced accuracy: 0.5006519734122062
Precision score 0.7329660238751148
Recall score 0.27553562531141007

Model Entropy model max depth 3
Accuracy: 0.7326139088729017
Balanced accuracy: 0.5
Precision score for 1 0.7326139088729017
Recall score for 0 0.0

Gini impurity model - max depth 3
Accuracy: 0.7326139088729017
Balanced accuracy: 0.5
Precision score 0.7326139088729017
Recall score 0.0

R-squared results for the four models
(R-squared for Decision Tree, model Entropy with no max)   = 1.0
(R-squared for Decision Tree, model Gini with no max   )   = 1.00
(R-squared for Decision Tree, model Entropy with max 3)   = 0.7370
(R-squared for decision Tree, model Gini with max 3   )   = 0.78

Despite similar accuracy scores and R-squared values across all models, we lacked sufficient evidence to conclude which model emerged as a better fit. Therefore, we proceeded to analyze the confusion matrix for the first two Decision Tree models and the ROC curves for all three models (Logistic Regression, Random Forest, and Decision Tree). The results from the Decision Tree models did not differ significantly from those obtained from the Logistic Regression, particularly for the last Decision Tree model (using entropy and Gini with a maximum depth of 3). However, the results for the first two Decision Tree models (using entropy and Gini without a maximum depth limit) were similar to those from the Random Forest. This similarity makes it challenging to determine which model outperforms the others. For deeper insights, we examined the ROC curves, where all three models showed scores ranging between 0.5 to 0.6, with the Decision Tree using the Gini model achieving the highest score of 0.54.core of 0.54.

In terms of conclusion, the most convincing revelation is that this dataset is not suitable for multilinear regression analysis. As for determining which model performs better than another, it ultimately remains at the discretion of the reader, given that the results for both Random Forest and Decision Tree models are quite similar. For instance, the ROC score for the Decision Tree is 0.54, whereas it is 0.51 for the Random Forest. Similarly, the specificity is 29% for the Random Forest and 27% for the Decision Tree. From a statistical standpoint, however, Random Forest holds a slight edge due to its F1-score exceeding 80%. This suggests that the Random Forest model can achieve high precision and recall simultaneously when predicting Amazon Sales status. Indeed, both Random Forest and Decision Tree models appear suitable for assessing this type of dataset. Overall, the study's results did not provide undeniable evidence favoring one model over another, indicating there is room for improvement in future analysis for this type of dataset.