**Dataset Description:**

This dataset consists of 31645 records of mock rail journeys by passengers via National Rail in the England, Scotland and Wales during the period 01.Jan.2024 – 30.Apr.2024.

The data includes mock train ticket sales that contain details about passenger details, payment methods, ticket types, journey times, destinations, delays and refund requests.

Origin of data comes from Maven Analytics Challenge where it asked participants of the competition to create an exploratory dashboard to:

* Identify the most popular routes
* Determine peak travel times
* Analyze revenue from different ticket types & classes
* Diagnose on-time performance and contributing factors

**EDA:**

* **Variable:** Payment.Method

*Categories: Contactless, Credit Card, Debit Card.*

* **Variable:** Railcard

*Categories: Adult, Disabled, Senior.*

* **Variable:** Ticket.Class

*Categories: Standard, First Class.*

* **Variable:** Ticket.Type

*Categories:* Advance, Off-Peak, Anytime

* **Variable:** Price (GBP)

*Categories: Numerical - Range of Integers.*

* **Variable:** Departure.Station and Arrival.Station.

*Categories: Station Strings*

* **Variable:** Departure , Scheduled Arrival Time and Actual.Arrival.

*Categories: Datetime*

* **Variable:** Journey.Status

*Categories: On Time, Delayed*

* **Variable:** Reason.For.Delay

Categories: Reason Strings

* **Variable:** Refund Requests

*Categories: Boolean – Yes/No*

**Data pre-processing & Feature Engineering:**

After examining the data, it was noticeable that some columns (Departure, Scheduled.Arrival) had missing values and they were treated in different ways:  
1) Some rows were deleted due to insufficient information

2) Some rows were filled using hot-deck imputation -> Filling the values depending on other similar rows values. E.g: similar routes.

For "Reason.for.Delay," categories 'Staffing' and 'Staff' were merged into 'Staff-Related' due to their common cause.

**Univariate Analysis:**

The analysis is done to investigate gain intel about payment methods, tickets, arrival and departure stations and times.

It is observable from the figures and charts that **Credit Card** is the most common payment method for the passengers, representing 60.5% of the transactions. The dominance of Credit Card can be due to their established trusted in comparison to contactless payment which is relatively new, though, gaining popularity due to its convenience, moreover, lower adoption of debit card might be due to them not offering perks like the other 2 payment methods, e.g. credit cards offer deferred payments and rewards.

A pie chart with text on it

Description automatically generated

Figure 1: Payment Method Distribution

**Advance** tickets made up 55.5%, **Anytime** 16.9%, and **Off-Peak** 27.6%. Only 3,056 were First Class, compared to 28,500 Standard. This suggests a need to explore why Advance dominates and why First Class has few buyers—perhaps the experience doesn’t justify the cost.

A graph with a number of classes

Description automatically generated with medium confidence

Figure 2: Ticket Class Frequency & Ticket Type Distribution

In terms of journey status, 86.8% were **on time**, 7.2% were **delayed** (4165 journeys in total) from which:

Table 1: Reason for Delay vs number of occurrence

|  |  |
| --- | --- |
| **Reason For Delay** | **Number of Occurences** |
| Weather | 1372 |
| Signal Failure | 967 |
| Staff-Related | 806 |
| Technical Issue | 706 |
| Traffic | 314 |

A pie chart with different colored circles

Description automatically generated

Figure 3: Journey Status Distribution

Regarding journey specifications, the top five departure stations, ranked from highest to lowest, were Liverpool Lime Street (1st), Manchester Piccadilly, London Euston, London Paddington, and London Kings Cross

A graph showing different colored bars

Description automatically generated

Figure 4: Top 5 frequent departure stations

When plotting a Kernel Density Estimate for departure times, it is noticeable peak hours had the highest departures, from 5 AM – 8AM, then again from 16 PM till 19 PM.

A graph of a number of times

Description automatically generated with medium confidence

Figure 5: KDE for departure times

The top 5 arrival stations were Birmingham New Street, Liverpool Lime Street, York, Manchester Piccadilly, and Reading. Liverpool Lime Street and Manchester Piccadilly are the busiest, ranking in the top 5 for both arrivals and departures.A graph showing different colored bars

Description automatically generated

Figure 6: Top 5 Frequent Arrival stations

A graph of a graph

Description automatically generated

Figure 7: KDE for arrival times

Peak hours are similar to departure time KDE, though, is it observable that the KDE is shifted to the right, this might likely be because people are returning home rather than departing in later hours.

**Multivariate Analysis:**

This analysis explores questions like: "How do payment methods relate to price distribution?", "How do journey statuses/payment methods affect refund requests?", and "Which delay reasons impact different months most?"

**(1)**

A graph of credit card price

Description automatically generated

Figure 8: All payment methods were heavily skewed to the right showing concentration of ticket prices at lower end of the scale.

**(2)**

A graph of refund status

Description automatically generated

Figure 9: Journey Status by Refund Request heatmap

From the plot it is observable that no passengers requested a refund when the journey was on time, this shows that if the journey is on time, the passengers are satisfied.

Around 0.23% of passengers whose journey got delayed requested a refund and around 0.3% of passengers whose journey got cancelled requested a refund.

**(3)**

A graph of payment method

Description automatically generated

Figure 10: Payment Method by Refund Request heatmap

Key point from the heatmap is that almost 0.3% of passengers that paid by debit card requested a refund which is a very big percentage compared to other payment methods, even though, they had similar median value indicating similar ranges. (comparing median because no affect by skewness)

**(4)**

A graph of different colored bars

Description automatically generated

A comparison of a graph

Description automatically generated with medium confidence

Figure 11: Subplots for number of delays/cancelled due to X by month

March saw the highest signal failures, technical issues, and traffic incidents, suggesting a need for further investigation. January experienced highest in staff-related issues, possibly due to post-holiday laziness (burnout).

February logically, one of the coldest months, had the most weather-related delays and cancellations

**Question 3:**

**A close up of a computer screen

Description automatically generated**

Figure 12: Calculation of DelayInMinutes

**Question 4:**

**A screenshot of a computer code

Description automatically generated**

Figure 13: Probabilities for values 5 and 25 of requesting a refund

A math equations on a piece of paper

Description automatically generated

Figure 14: Calculation by hand

**Question 5:**

**Model Chosen:** Logistic Regression

**Reason of Choice:**  
(1) Binary Classification: Refund Request (yes/no).  
(2) Feature Importance: Explains how features influence the target using coefficients.

For this task, the classes were imbalanced, 30531 records ouput “No” for refund request and only 1114 output “Yes”.

**Class Under sampling:**

The majority class (RefundRequest = No) was under sampled to almost match the number of records for the Yes class.

**Model training and testing:**

A screenshot of a computer screen

Description automatically generatedA diagram of a confusion matrix

Description automatically generated with medium confidence

Figure 15: Classification report of the model

Figure 16: Confusion matrix of the predicted and actual records

**Features Chosen:**

[‘Payment.Method.Encoded’, ‘Journey.Status.Encoded’]

**Reason of choice:**

* EDA shows a correlation between Payment Method and refund requests, with debit card users more likely to request a refund (notice strong positive coefficient).
* Journey Status correlates strongly with refund likelihood, as no refunds occurred when trains were on time, but increases were seen with delays/cancellations (notice strong negative coefficient)

In Figure 14, while the model is more precise for predicting “No” (0.98), it is better at recalling “Yes” instances (0.98), this means while the model is highly precise in predicting No – True Negatives, it is better at identifying “Yes” cases – True Positive.

* Though, it dees not miss any Yes cases, it might classify some No instances as Yes. – False Positives

The equal F1 scores indicate balanced performance in both precision and recall.

**Probabilities and classification:**

**A screenshot of a computer screen

Description automatically generated**

Figure 17: ToPredict probabilities and classification

**Comparison with other Models:**

**Features: [Price, Payment.Method.Encoded]**

**A blue squares with white text

Description automatically generated**

Figure 18: Confusion matrix for Price, Payment method model

**A screenshot of a computer screen

Description automatically generated**

Figure 17: Classification report for Price, Payment Method model

For this model, it achieved very low recall for Class “Yes” meaning significant number of “Yes” instances are being missed. (False Negatives)

**A graph of a refund

Description automatically generated with medium confidenceA screenshot of a computer screen

Description automatically generatedFeatures: [Price, Journey.Status.Encoded]**

Figure 18: Classification report for Price, Journey Status model

Figure 19: Confusion matrix for Price, Journey Status model

For this model, it had precision and recall of 0 for class “Yes”, meaning all “Yes” instances are being classified as “No”. (False Negatives)

**Features: [Payment.Method.Encoded, Journey.Status.Encoded]**

**A screenshot of a computer screen

Description automatically generatedA blue squares with white text

Description automatically generated**

Figure 20: Classification report for Payment method, Journey Status model (NOT UNDERSAMPLED)

Figure 21: Confusion matrix for Payment Method, Journey Status (NOT UNDERSAMPLED)

For this model, I use the same chosen features for my chosen model but, without undersampling, we can see the precision and recall for “Yes” class are lower than the chosen model. (More False Negatives and False Positives)

**Appendix:**

**#** **NOTE: I only picked the "most interesting" plots to include in the report. However, in my code I have many more**

**Therefore you might see plots here that weren't included!**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

df = pd.read\_csv("MavenRail.csv")

num\_records = df.shape[0]

print(f"Number of Records: {num\_records}")

def attributeCol(Column):

print(f"{Column} Missing Values", df[Column].isna().sum())

for i in df.columns:

attributeCol(i)

missing\_departure\_station = df[df['Departure'].isna()]

print("Here are the rows where the Departure is missing:")

print(missing\_departure\_station)

# Firstly I will be assigning indexes to records so that I can handle them easier

df['Row\_ID'] = range(1,len(df)+1)

df.to\_csv('MavenRail.csv',index=False)

# Dropping the first record

df = df[df['Row\_ID'] != 23614]

# Calculating Journey Time for Liverpool - Manchester in other records to fill the second missing record

df['Departure'] = pd.to\_datetime(df['Departure'])

df['Scheduled.Arrival'] = pd.to\_datetime(df['Scheduled.Arrival'])

filtered\_stations = df[(df['Departure.Station'] == 'Liverpool Lime Street') & (df['Arrival.Station'] == 'Manchester Piccadilly')

& (df['Journey.Status'] == 'On Time')]

print(filtered\_stations['Scheduled.Arrival'] - filtered\_stations['Departure'])

from datetime import timedelta

# Now we know it is 30 minutes, we add it to the missing record

df.loc[df['Row\_ID']==23615,'Departure'] = df.loc[df['Row\_ID'] == 23615, 'Scheduled.Arrival'] - timedelta(minutes=30)

df.to\_csv('MavenRailModified.csv',index=False)

# Calculation of time for the 3rd departure record missing

filtered = df[(df['Departure.Station'] == 'York') & (df['Arrival.Station'] == 'Durham')

& (df['Journey.Status'] == 'On Time')]

print(filtered['Scheduled.Arrival']- filtered['Departure'])

df.loc[df['Row\_ID']==23619,'Departure'] = df.loc[df['Row\_ID'] == 23619, 'Scheduled.Arrival'] - timedelta(minutes=50)

df.to\_csv('MavenRailModified.csv',index=False)

# Now we check the Scheduled Arrival missing

missing\_arrival = df[df['Scheduled.Arrival'].isna()]

print(missing\_arrival)

# Since this route, all of its records have signal failure, this missing values one wont be informative

df = df[df['Row\_ID'] != 23610]

# I calculcate duration from scheduled arrival - departure to hot deck impute

filtered\_stations = df[(df['Departure.Station'] == 'York') & (df['Arrival.Station'] == 'Edinburgh')

& (df['Journey.Status'] == 'On Time')]

print(filtered\_stations['Scheduled.Arrival'] - filtered\_stations['Departure'])

# I calculcate duration from scheduled arrival - departure to hot deck impute

filtered\_stations = df[(df['Departure.Station'] == 'Liverpool Lime Street') & (df['Arrival.Station'] == 'London Euston')

& (df['Journey.Status'] == 'On Time')]

print(filtered\_stations['Scheduled.Arrival'] - filtered\_stations['Departure'])

# Add the missing values depending on hot deck impute

df.loc[df['Row\_ID']==23617,'Scheduled.Arrival'] = df.loc[df['Row\_ID'] == 23617, 'Departure'] + timedelta(hours=2,minutes=30)

df.loc[df['Row\_ID']==23625,'Scheduled.Arrival'] = df.loc[df['Row\_ID'] == 23625, 'Departure'] + timedelta(hours=2,minutes=15)

df.to\_csv('MavenRailModified.csv',index=False)

# Table like information for all univariates

import seaborn as sns

Payment\_Dist= df['Payment.Method'].value\_counts()

print(Payment\_Dist)

df['Railcard'] = df['Railcard'].fillna('None')

Railcard\_Dist = df['Railcard'].value\_counts()

print(Railcard\_Dist)

Ticket\_Dist = df['Ticket.Class'].value\_counts()

print(Ticket\_Dist)

Ticket\_Type\_Dist = df['Ticket.Type'].value\_counts()

print(Ticket\_Type\_Dist)

Journey\_Status\_Dist= df['Journey.Status'].value\_counts()

print(Journey\_Status\_Dist)

df['Reason.for.Delay'] = df['Reason.for.Delay'].replace({'Staffing' : 'Staff-Related', 'Staff' : 'Staff-Related'})

Reason4Delay\_Dist = df['Reason.for.Delay'].value\_counts()

print(Reason4Delay\_Dist)

print("All Delays: ",Reason4Delay\_Dist.sum())

df.to\_csv("MavenRailModified.csv",index=False)

import matplotlib.pyplot as plt

import seaborn as sns

colors = ['skyblue', 'salmon', 'lightgreen', 'gold', 'orange']

# creating a 2x2 grid of subplots

fig, axes = plt.subplots(2, 2, figsize=(12, 6))

# PAYMENT

axes[0, 0].pie(Payment\_Dist, labels=Payment\_Dist.index, colors=colors,

autopct='%1.1f%%', startangle=90)

axes[0, 0].set\_title('Payment Method Distribution')

# RAILCARD

axes[0, 1].pie(Railcard\_Dist, labels=Railcard\_Dist.index, colors=colors,

autopct='%1.1f%%', startangle=90)

axes[0, 1].set\_title('Railcard Distribution')

# TICKET (using countplot)

sns.countplot(x='Ticket.Class', data=df,hue='Ticket.Class', palette=colors, ax=axes[1, 0])

axes[1, 0].set\_title('Ticket Class Frequency')

# TYPE

axes[1, 1].pie(Ticket\_Type\_Dist, labels=Ticket\_Type\_Dist.index, colors=colors,

autopct='%1.1f%%', startangle=90)

axes[1, 1].set\_title('Ticket Type Distribution')

plt.tight\_layout()

plt.show()

# Histogram for price distribution

df['Price'].plot(kind='hist', bins=30, color='blue', alpha=0.7)

plt.title('Price Distribution')

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.show()

# Statistics

print(df['Price'].describe())

import matplotlib.pyplot as plt

# grouping the data by 'Payment.Method' and plot separate histograms for each group

payment\_methods = df['Payment.Method'].unique()

# figure for subplots

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

# plotting the histograms on axis

for i, payment\_method in enumerate(payment\_methods):

# flitering the dataframe for each payment method

payment\_data = df[df['Payment.Method'] == payment\_method]

axes[i].hist(payment\_data['Price'], bins=20, color='blue', alpha=0.7)

axes[i].set\_title(f'{payment\_method} Price Distribution')

axes[i].set\_xlabel('Price')

axes[i].set\_ylabel('Frequency')

plt.tight\_layout()

plt.show()

# Count plot for departure frequency, frequency on the X axis and Station name on Y.

sns.countplot(y="Departure.Station", data=df,order=df['Departure.Station'].value\_counts().head(5).index,hue="Departure.Station",palette='deep')

plt.title("Frequency of Departure Stations")

plt.xlabel("Frequency")

plt.ylabel("Station Name")

plt.show()  
  
# Similar as before but for Arrival stations

sns.countplot(y="Arrival.Station", data=df,order=df['Arrival.Station'].value\_counts().head(5).index,hue="Arrival.Station",palette='deep')

plt.title("Frequency of Arrival Stations")

plt.xlabel("Frequency")

plt.ylabel("Station Name")

plt.show()

#KDE for departure times

plt.figure(figsize=(10, 6))

sns.kdeplot(df['Departure'].dt.hour, shade=True, color='blue')

plt.title('Kernel Density Estimate of Departure Times')

plt.xlabel('Hour of the Day')

plt.ylabel('Density')

plt.xticks(range(0, 24))

plt.show()

# FREQ of departures per hour

time\_counts = df.groupby(df['Departure'].dt.hour).size()

plt.figure(figsize=(10, 6))

time\_counts.plot(kind='line', marker='o', color='blue')

plt.title('Hourly Departure Frequency')

plt.xlabel('Hour of the Day')

plt.ylabel('Frequency')

plt.xticks(range(0, 24))

plt.grid()

plt.show()

#KDE for arrival hours

plt.figure(figsize=(10, 6))

sns.kdeplot(df['Scheduled.Arrival'].dt.hour, shade=True, color='blue')

plt.title('Kernel Density Estimate of Arrival Times')

plt.xlabel('Hour of the Day')

plt.ylabel('Density')

plt.xticks(range(0, 24))

plt.show()

# FREQ of arrivals per hour

time\_counts = df.groupby(df['Scheduled.Arrival'].dt.hour).size()

plt.figure(figsize=(10, 6))

time\_counts.plot(kind='line', marker='o', color='blue')

plt.title('Hourly Arrival Frequency')

plt.xlabel('Hour of the Day')

plt.ylabel('Frequency')

plt.xticks(range(0, 24))

plt.grid()

plt.show()

# pie chart for journey status

journeycolors = ['skyblue','salmon','lightgreen']

plt.pie(Journey\_Status\_Dist,labels=Journey\_Status\_Dist.index,colors=journeycolors,autopct='%1.1f%%', startangle=90)

plt.title("Journey Status Distribution")

# count plot for reason for delay occurences.

plt.figure(figsize=(10,6))

sns.countplot(x='Reason.for.Delay', data=df,hue='Reason.for.Delay', palette=colors)

plt.title("Reasons For Delay Frequency")

plt.show()

# Cross table for journey status and refund requests

df1 = pd.read\_csv("MavenRailModified.csv")

crosstab\_analysis = pd.crosstab(df1['Journey.Status'], df1['Refund.Request'])

print(crosstab\_analysis)

# plotting a heatmap for the cross table

sns.heatmap(crosstab\_analysis, annot=True, fmt="d", cmap="YlGnBu", linewidths=0.5)

plt.title('Heatmap of Journey Status by Refund Request')

plt.xlabel('Refund Request')

plt.ylabel('Journey Status')

plt.show()

# Line plot for median for each hour and the hour

df['Departure'] = pd.to\_datetime(df['Departure'])

hourly\_skewness = df.groupby(df['Departure'].dt.hour)['Price'].apply(lambda x: x.skew())

print(hourly\_skewness)

# plotting a line plot with median as estimator

df1['Departure'] = pd.to\_datetime(df1['Departure'])

df1['Hour'] = df1['Departure'].dt.hour

plt.xticks(range(0,24,2))

sns.lineplot(x='Hour', y='Price', data=df1,estimator='median')

# Filter to Delayed

df\_Disrupt = df1[ (df1['Journey.Status'] == 'Delayed') | (df1['Journey.Status'] == 'Cancelled') ]

# ONLY TOP 5

stations\_Disrupt = df\_Disrupt['Departure.Station'].value\_counts().head(5)

# Plotting the frequencies

plt.figure(figsize=(10, 6))

sns.barplot(x=stations\_Disrupt.index, y=stations\_Disrupt.values, palette='viridis')

plt.title('Frequency of Disrupted Journeys by Departure Station')

plt.xlabel('Departure Station')

plt.ylabel('Number of Delayed/Cancelled Journeys')

plt.xticks(rotation=45, ha='right') # Rotate station names for better readability

plt.yticks(np.arange(0, max(stations\_Disrupt.values)+100, step=100))

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# filter aand aggregate by reason and journey status

def filter\_and\_aggregate(df, reason, journey\_status=['Delayed', 'Cancelled']):

grouped = df.groupby([df['Departure'].dt.month, 'Journey.Status', 'Reason.for.Delay']).size()

filtered = grouped.xs(reason, level='Reason.for.Delay')

filtered = filtered[filtered.index.get\_level\_values('Journey.Status').isin(journey\_status)]

combined = filtered.groupby(level=0).sum()

return combined

# plotting data

def plot\_aggregated\_data(ax, data, title, color\_palette):

sns.barplot(x=data.index, y=data.values, palette=color\_palette, ax=ax)

ax.set\_title(title)

ax.set\_xlabel('Month')

ax.set\_ylabel('Number of Journeys')

fig, axes = plt.subplots(2, 3, figsize=(18, 12))

axes = axes.flatten()

# Iterate over the reasons of delay and plot

for i, (reason, color, title) in enumerate([

('Staff-Related', 'viridis', 'Number of Delayed/Cancelled Journeys Due to Staff-Related by Month'),

('Traffic', 'magma', 'Number of Delayed/Cancelled Journeys Due to Traffic by Month'),

('Weather', 'deep', 'Number of Delayed/Cancelled Journeys Due to Weather by Month'),

('Signal Failure', 'magma', 'Number of Delayed/Cancelled Journeys Due to Signal Failure by Month'),

('Technical Issue', 'deep', 'Number of Delayed/Cancelled Journeys Due to Technical Issue by Month'),

]):

aggregated\_data = filter\_and\_aggregate(df1, reason)

plot\_aggregated\_data(axes[i], aggregated\_data, title, color)

plt.subplots\_adjust(left=0.01, right=1, top=0.9, bottom=0.1, wspace=0.4, hspace=0.4)

plt.show()

df2 = pd.read\_csv("MavenRailModified.csv")

# crosstab for payment method by refund request

crosstab\_analysis2 = pd.crosstab(df1['Payment.Method'], df1['Refund.Request'])

print(crosstab\_analysis2)

# plotting heatmap for the result

sns.heatmap(crosstab\_analysis2, annot=True, fmt="d", cmap="YlGnBu", linewidths=0.5)

plt.title('Heatmap of Payment Method by Refund Request')

plt.xlabel('Refund Request')

plt.ylabel('Payment Method')

plt.show()

payments = df2.groupby('Payment.Method')['Price'].median()

print(payments)

# I do actual arrival - scheduled arrival to get the delay time

df['Actual.Arrival'] = pd.to\_datetime(df['Actual.Arrival'])

df.loc[df['Journey.Status'] == 'Delayed', 'DelayInMinutes'] = (df['Actual.Arrival'] - df['Scheduled.Arrival']).dt.total\_seconds()/60

df.to\_csv('MavenRailModified.csv',index=False)

fourth\_df = df.loc[(df['Journey.Status'] == 'Delayed') | (df['Journey.Status'] == 'Cancelled')]

# for logistic regression I need to make this class binary, 0 and 1s.

fourth\_df['MediumPrice'] = fourth\_df['Price'].apply(lambda x:1 if 10 < x <= 30 else 0)

fourth\_df['Refund.Request'] = fourth\_df['Refund.Request'].map({'Yes': 1, 'No': 0})

fourth\_df.to\_csv("MavenRailMedium.csv",index=False)

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

mediumDF = pd.read\_csv("MavenRailMedium.csv")

# reshaping so that its in 2d

X = mediumDF['MediumPrice'].values.reshape(-1, 1)

y = mediumDF['Refund.Request']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# creating and training the model

model = LogisticRegression(random\_state=42)

model.fit(X\_train,y\_train)

#predicting the data

y\_predict = model.predict(X\_test)

# classification report on the model

print("Accuracy:", model.score(X\_test, y\_test))

print(classification\_report(y\_predict,y\_test))

# got intercept and coefficient variables.

intercept = model.intercept\_

coefficient = model.coef\_

print("intercept: ", intercept)

print("coefficient: ", coefficient)

# probability of requesting refund given paid 5 (Question 4.I)

probabilities\_5 = model.predict\_proba([[0]])[0][1]

print(probabilities\_5)

# probability of requesting refund given paid 25 (Question 4.II)

probabilities\_25 = model.predict\_proba([[1]])[0][1]

print(probabilities\_25)

df5 = pd.read\_csv('MavenRailMedium.csv')

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import classification\_report

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

df5['Payment.Method.Encoded'] = encoder.fit\_transform(df5['Payment.Method'])

df5['Journey.Status.Encoded'] = encoder.fit\_transform(df5['Journey.Status'])

feature\_combinations = [

['Price', 'Payment.Method.Encoded'],

['Price', 'Journey.Status.Encoded'],

['Payment.Method.Encoded', 'Journey.Status.Encoded']

]

X = df5[['Payment.Method.Encoded','Journey.Status.Encoded']]

y = df5['Refund.Request']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# evaluate the model over these combinations

for features in feature\_combinations:

X = df5[features]

y = df5['Refund.Request']

# splitting data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# fit the model

model = LogisticRegression(random\_state=42, max\_iter=1000)

model.fit(X\_train, y\_train)

# predict

y\_pred = model.predict(X\_test)

# Print classification report for evaluation

print(f"Classification Report for features: {features}")

print(classification\_report(y\_test, y\_pred))

print("\n")

cm = confusion\_matrix(y\_test,y\_pred)

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Refund', 'Refund'], yticklabels=['No Refund', 'Refund'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, roc\_auc\_score

from imblearn.under\_sampling import RandomUnderSampler

# Load the dataset

df10 = pd.read\_csv("MavenRail.csv")

# encoding the chosen features

encoder = LabelEncoder()

df10['Payment.Method.Encoded'] = encoder.fit\_transform(df10['Payment.Method'])

df10['Journey.Status.Encoded'] = encoder.fit\_transform(df10['Journey.Status'])

X = df10[['Payment.Method.Encoded','Journey.Status.Encoded']]

y = df10['Refund.Request']

# print classes before undersampling

print(f"Class distribution before undersampling:\n{y.value\_counts()}")

# Undersample the majority class

undersampler = RandomUnderSampler(sampling\_strategy='auto',random\_state=42)

X\_balanced, y\_balanced = undersampler.fit\_resample(X, y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_balanced, y\_balanced,

test\_size=0.2, random\_state=42)

# Print classes after undersampling

print(f"Class distribution after undersampling:\n{y\_train.value\_counts()}")

# initialize and train the model

model = LogisticRegression(random\_state=42, max\_iter=1000)

model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = model.predict(X\_test)

# Print evaluation metrics

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

print("Coefficients of features, [Payment.Method, Journey.Status]")

print(model.coef\_)

# Check that y\_test and y\_pred have the same length

print(f"Length of y\_test: {len(y\_test)}")

print(f"Length of y\_pred: {len(y\_pred)}")

# Get the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Create a heatmap of the confusion matrix

plt.figure(figsize=(6, 5))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Refund', 'Refund'], yticklabels=['No Refund', 'Refund'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

import pandas as pd

from sklearn.preprocessing import LabelEncoder

# Load the new data for prediction

predictdf = pd.read\_csv("ToPredict(1).csv")

encoder = LabelEncoder()

predictdf['Payment.Method.Encoded'] = encoder.fit\_transform(predictdf['Payment.Method'])

predictdf['Journey.Status.Encoded'] = encoder.fit\_transform(predictdf['Journey.Status'])

X\_predict = predictdf[['Payment.Method.Encoded','Journey.Status.Encoded']]

y\_prob = model.predict\_proba(X\_predict)

# we do [:,1] to extract the probabilities for YES (1) refund request

predictdf['Predicted.Refund.Probability'] = y\_prob[:,1]

# we predict the classification of the classes

predictdf['Predicted.Refund.Request'] = model.predict(X\_predict)

# for visualisation purposes

print(predictdf[[ 'Predicted.Refund.Request','Predicted.Refund.Probability']])