THE UNIVERSITY OF MANCHESTER

2024 – 2025 Semester 1

MSc Data Science

DATA70121 Statistics and Machine Learning 1: Statistical Foundations

Coursework

EDA & REGRESSION

22.11.2024

Student ID: 10581125

Mark Muldoon and Diego Perez Ruiz

MANCHESTER 1824

The University of Manchester

1. Description of the Data

The dataset, named MavenRail.csv is a collection of mock data to create a simulation of rail journeys in the UK. It details fictional train trips taken by travelers in the UK between January 1 to April 30 in 2024.

The data originates from a visualization contest which is run by Maven Analytics, which is a company specializing in data science training. Although the dataset offers a reliable structure, it can be ranked as 3 stars according to the Spiegelhalter rating because it comes from a synthetic source, contains simulation data and is not identical to real data.

In this dataset, there are 31645 records, and the journey information of each passenger is included numerically or categorically, as well as journey information such as the starting point of the journey, the ending point, the departure time, and the lateness status.

If we examine each piece of information closely:

- **Payment Method:** The payment method used by passengers to purchase the ticket.
- **Rail card:** The type of rail card used (adult, senior).
- **Ticket Class/Type:** The type of ticket purchased.
- **Price:** The amount paid for the journey, in pounds.
- **Departure Station:** The passenger's departure station.
- **Arrival Station:** The passenger's arrival station.
- **Departure:** The train's departure time.
- **Scheduled Arrival:** The planned arrival time.
- **Actual Arrival:** The actual arrival time
- **Journey Status:** Indicates whether the train is delayed, cancelled or on time.
- **Reason for Delay:** The reason for the delay.
- **Refund Request:** Whether the passenger requests a refund.

2. Exploratory Data Analysis

In this section of the report, the univariate distributions within the various columns will examined and the relationship between the columns will also examined with multivariate analysis to perform explanatory data analysis.

Firstly, when analyzing the Price column, it was discovered that the graph was right-skewed. Looking at the bar graph and the KDE curve, the peak price range for tickets is £0-£20. There were also outliers for this column, with ticket prices close to £100 (Figure – 1).

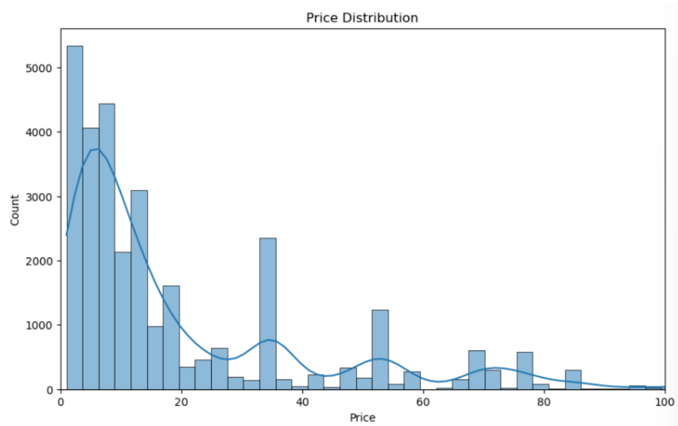


Figure – 1

How these payments were made was evaluated through payment method analysis. When payment methods were considered, it was analyzed that nearly 60% of the passengers made the payment by credit card, only 5.3% used debit cards, and the remaining passengers made contactless payments (Figure – 2).

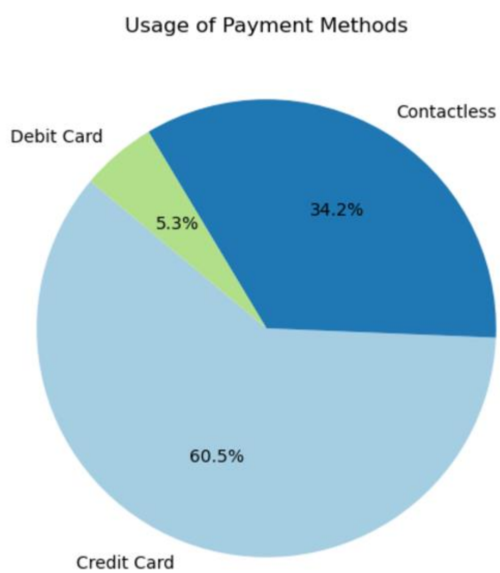


Figure – 2

When Journey status distribution was examined, it was seen that 27479 trips in the dataset were carried out as planned, 2289 were delayed, and 1077 were cancelled. To compare these numbers to the ratio, it was found that 86.8% of trips were completed on time and only 5.9% were cancelled (Figure - 3).

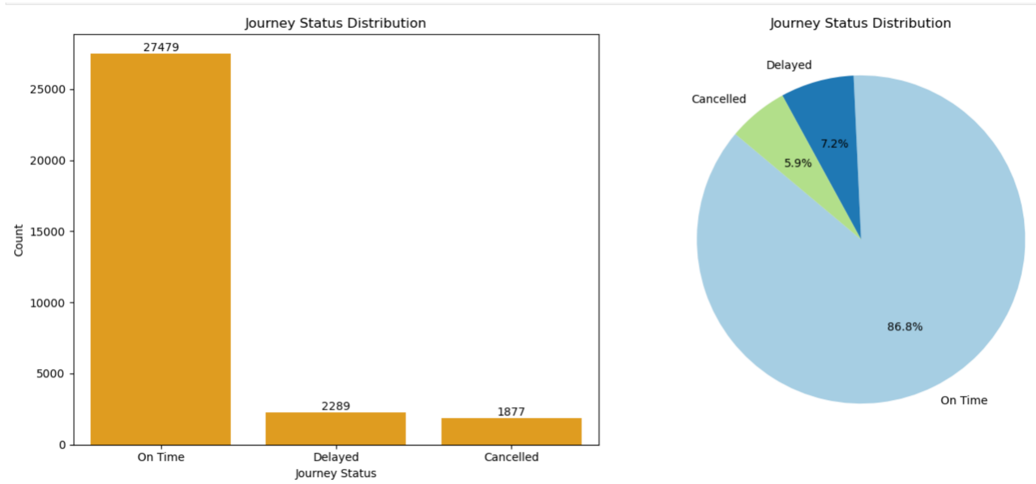


Figure – 3

The delay and cancellation reasons ratio were shown in the pie charts below. Staff and Staffing titles were combined because they represent the same. The reason with the largest share for cancel was signal failure, while the weather conditions wer the biggest factor for delay. For both cases, it was seen that heavy traffic was the reason with the lowest rate (Figure – 4).

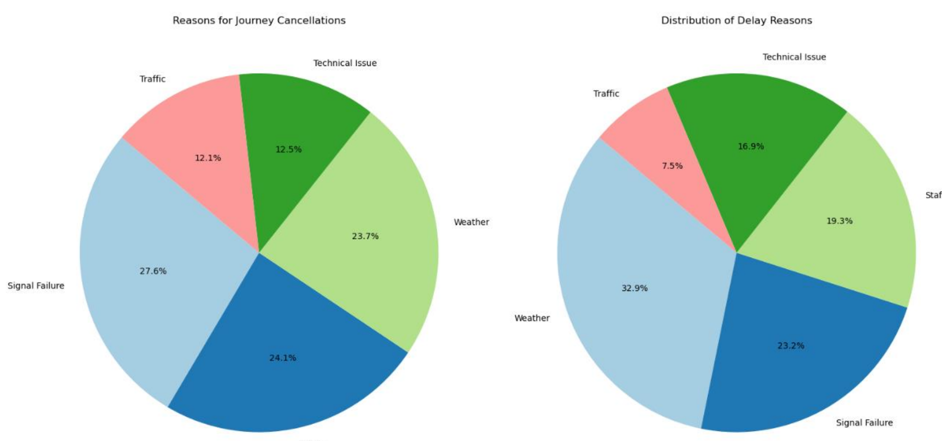


Figure – 4

The data collected from all these journeys was analyzed to find the most used routes. For this purpose, departure and arrival stations were grouped to find the most used routes. At the end of the analysis, the most popular route was Manchester Piccadilly - Liverpool Lime Street with 4626 trips (Figure – 5).

	Departure.Station	Arrival.Station	Count
0	Manchester Piccadilly	Liverpool Lime Street	4626
1	London Euston	Birmingham New Street	4208
2	London Kings Cross	York	3922
3	London Paddington	Reading	3873
4	London St Pancras	Birmingham New Street	3470
5	Liverpool Lime Street	Manchester Piccadilly	3001
6	Liverpool Lime Street	London Euston	1096
7	London Euston	Manchester Piccadilly	712
8	Birmingham New Street	London St Pancras	701
9	London Paddington	Oxford	485

Figure – 5

A departure-based chart was created to show the hours when stations are busy. The number of departures was seen to be busy before morning work/school hours (6-8 am) and after work/school hours (4-6 pm) (Figure – 6), and this density was also shown on the heat map on station basis (Figure – 7).

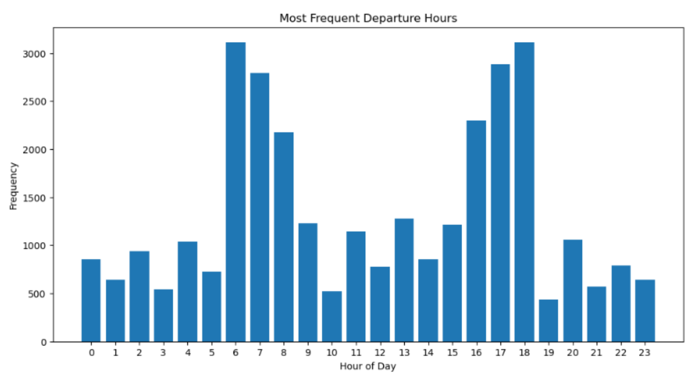


Figure – 6

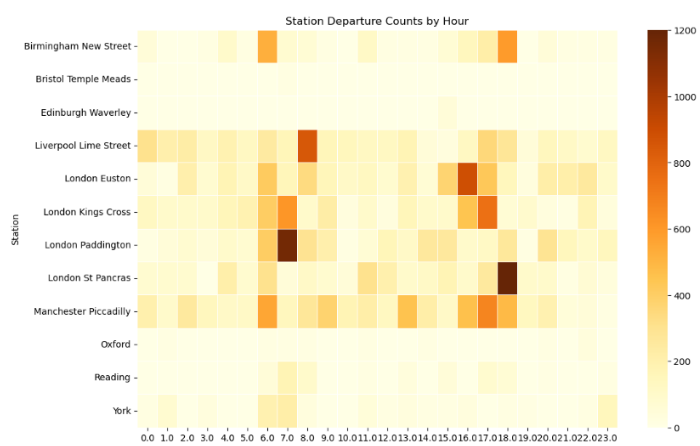


Figure - 7

At the end of the trips, only 3.5% of the passengers requested a refund. As a result of the multivariate analysis, there was either cancellation or delayed status in the trips for which a refund was requested. No refund request was observed in any trip that arrived on time (Figure – 8). Considering the Journey status, it was seen that 3.5% of the refund requests were made only in cases of delay and cancellation. On the other hand, 3052 tickets did not receive a refund request despite delay/cancellation (Figure – 9).

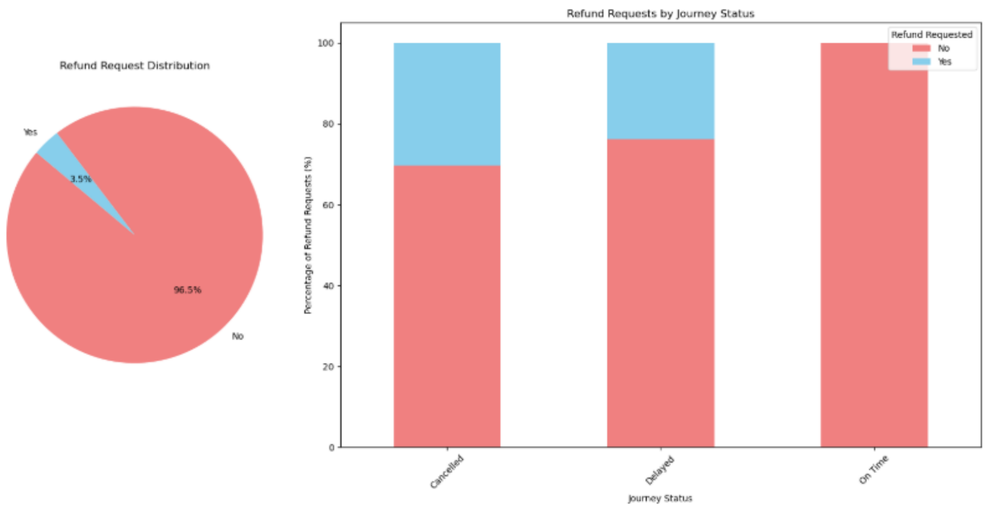


Figure - 8

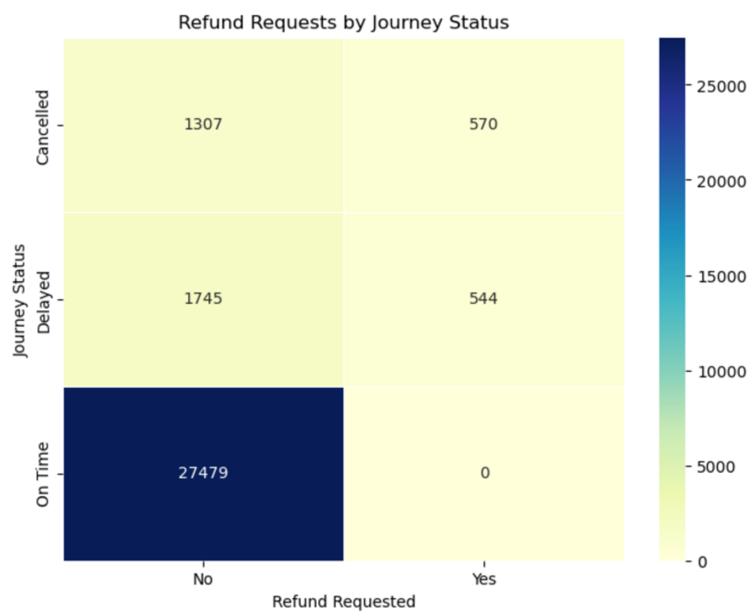


Figure - 9

The chart below compares the price ranges across different ticket types. Advance tickets had the lowest price range, typically under £20. Off-Peak tickets were in the middle range with slightly higher prices, while Anytime tickets had the widest price range and the highest median value. Anytime tickets were notable outliers with prices over £150. This clearly highlights the pricing strategies of ticket types based on flexibility and accessibility (Figure – 10).

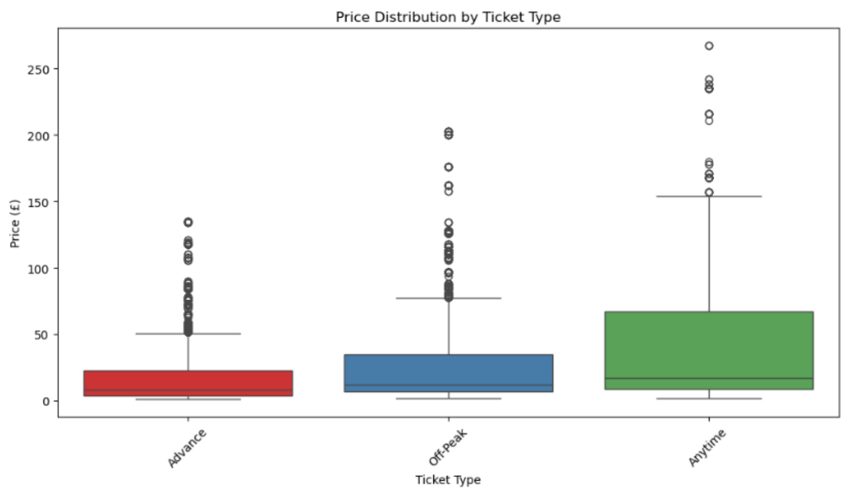


Figure – 10

3. Delay in Minutes

To find the amount of delay for delayed journeys, first the "DelayInMinutes" column was added to the dataset and initially set to Null for all data. Then the Scheduled and Actual Arrival columns were converted to datetime format. The "Actual.Arrival - Scheduled.Arrival" operation was performed in seconds and divided by 60, and the amount of delay was added to all rows. If the "DelayInMinutes" value was equal to 0, the value in these rows was changed to null (Figure – 11).

	Payment.Method	Railcard	Ticket.Class	Ticket.Type	Price	Departure.Station	Arrival.Station	Departure	Scheduled.Arrival	Actual.Arrival	Journey.Status	Reason.for.Delay	Refund.Request	DelayInMinutes
0	Contactless	Adult	Standard	Advance	43	London Paddington	Liverpool Lime Street	2024-01-01 11:00	2024-01-01 13:30:00	2024-01-01 13:30:00	On Time	NaN	No	NaN
1	Credit Card	Adult	Standard	Advance	23	London Kings Cross	York	2024-01-01 09:45	2024-01-01 11:35:00	2024-01-01 11:40:00	Delayed	Signal Failure	No	5.0
2	Credit Card	NaN	Standard	Advance	3	Liverpool Lime Street	Manchester Piccadilly	2024-01-02 18:15	2024-01-02 18:45:00	2024-01-02 18:45:00	On Time	NaN	No	NaN
3	Credit Card	NaN	Standard	Advance	13	London Paddington	Reading	2024-01-01 21:30	2024-01-01 22:30:00	2024-01-01 22:30:00	On Time	NaN	No	NaN
4	Contactless	NaN	Standard	Advance	76	Liverpool Lime Street	London Euston	2024-01-01 16:45	2024-01-01 19:00:00	2024-01-01 19:00:00	On Time	NaN	No	NaN

Figure – 11

4. Medium Price - Univariate Regression

After filtering the data with Journey.Status as On Time from dataset, it was assigned boolean values for the MediumPrice column according to whether the prices met the condition. Afterwards, because of the operations performed both in Python (Figure – 12) and manually (Figure – 13), the refund probability for £5 tickets was calculated as 0.257, while for £25 tickets this value was calculated as 0.319.

```
Probability of requesting a refund for £5 ticket: 0.26
Probability of requesting a refund for £25 ticket: 0.32
```

Figure – 12

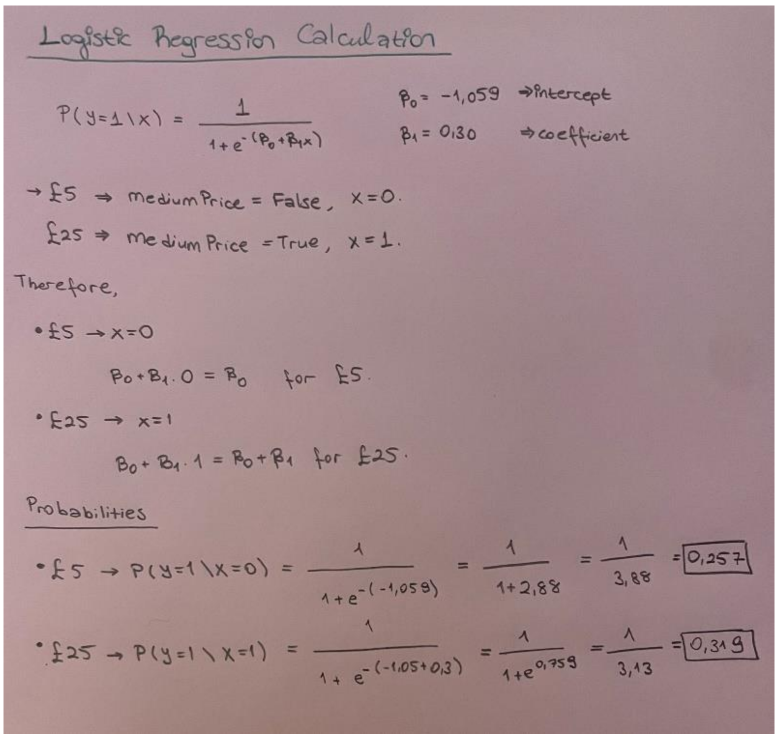


Figure – 13

5. Regression Model for Refund Prediction

Logistic Regression model was chosen because it is an easy to understand and effective model for binary classifications.

As seen in the EDA section for the selected features (Figure - 9), Journey.Status is an important factor in refund requests, when there was no delay or cancellation, the refund request was never observed. For Price, it was foreseen that expensive tickets are more likely to get refund request and used in the model. Therefore, a highly consistent model was obtained.

When the original data was used in training the model, it was determined that the model was biased because Refund.Request was observed as "No" at a high rate made the model biased. For this reason, while training the model, equal numbers of "Yes" and "No" data were mixed, and the model's feature of predicting over features was improved and made unbiased.

The predictions (Figure – 14) and results of the model (Figure – 15/16) are as follows:

	Price	Journey.Status_Delayed	Journey.Status_On Time	Refund.Probability	Refund
3	3	True	False	0.948203	Yes
5	3	False	False	0.921330	Yes
4	4	False	False	0.920594	Yes
7	22	False	False	0.906214	Yes
2	113	True	False	0.857410	Yes
6	126	True	False	0.840556	Yes
1	7	False	True	0.009763	No
0	54	False	True	0.006090	No

Figure – 14

```
              precision    recall  f1-score   support

          No       1.00      0.91      0.96       223
         Yes       0.92      1.00      0.96       223

    accuracy                           0.96       446
   macro avg       0.96      0.96      0.96       446
weighted avg       0.96      0.96      0.96       446
```

Figure – 15

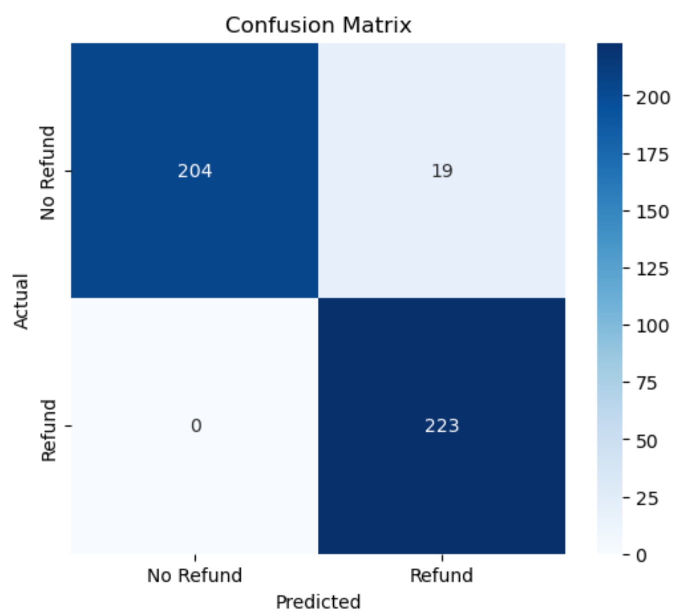


Figure – 16

6. Appendix

EXPLORATORY DATA ANALYSIS

```
#Data process and analysis libraries
import pandas as pd
import numpy as np
#Data visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("/Users/emrec/Desktop/SML Assignment/MavenRail.csv")
df.head()
row_count = len(df)
print(row_count)
null_numbers = df.isnull().sum()
print(null_numbers)
df.describe()
df.info()
```

Figure – 1

```
#Price distribution and KDE
plt.figure(figsize=(10, 6))
sns.histplot(df['Price'], bins=100, kde=True)
plt.title('Price Distribution')
plt.xlim(0, 100)
plt.show()
```

Figure – 2

```
#Payment method distribution
payment_methods = df['Payment.Method'].value_counts()
```

```
#Payment method % chart
plt.figure(figsize=(8, 6))
payment_methods.plot(kind='pie', autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
plt.title('Usage of Payment Methods')
plt.ylabel('')
plt.show()
```

Figure – 3

```
#Journey status distribution
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
#Bar chart for numbers
ax1 = sns.countplot(x='Journey.Status', data=df, color='orange', ax=axes[0])
ax1.set_title('Journey Status Distribution')
ax1.bar_label(ax1.containers[0])
ax1.set_xlabel('Journey Status')
ax1.set_ylabel('Count')
#Pie chart for percentage
journey_status_counts = df['Journey.Status'].value_counts()
axes[1].pie(journey_status_counts, labels=journey_status_counts.index, autopct='%1.1f%%',
startangle=140, colors=plt.cm.Paired.colors)
axes[1].set_title('Journey Status Distribution')
plt.tight_layout()
plt.show()
```

Figure – 4

```
#In dataset, Staff related delays anc cancellations labeled as Staff and Staffing, so we put them together
df['Reason.for.Delay'] = df['Reason.for.Delay'].replace({'Staffing': 'Staff'})

#Calculating the delay reasons and cancellation reasons
```

```
delay_reasons = df['Reason.for.Delay'].value_counts()

cancelled_journeys = df[df['Journey.Status'] == 'Cancelled']

cancellation_reasons = cancelled_journeys['Reason.for.Delay'].value_counts()

fig, axes = plt.subplots(1, 2, figsize=(16, 8))

#Pie chart for cancellation reasons - distribution

axes[0].pie(cancellation_reasons, labels=cancellation_reasons.index, autopct='%1.1f%%',
startangle=140, colors=plt.cm.Paired.colors)

axes[0].set_title('Reasons for Journey Cancellations')

#Pie chart for delay reasons - distribution

axes[1].pie(delay_reasons, labels=delay_reasons.index, autopct='%1.1f%%', startangle=140,
colors=plt.cm.Paired.colors)

axes[1].set_title('Distribution of Delay Reasons')

plt.tight_layout()

plt.show()
```

Figure – 5

```
#Most used routes among the dataset

top_routes = df.groupby(['Departure.Station',
'Arrival.Station']).size().sort_values(ascending=False).head(10)

top_routes = top_routes.reset_index(name='Count')

top_routes
```

Figure – 6

```
#Most frequent departure hours

df['Departure'] = pd.to_datetime(df['Departure'], errors='coerce')

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

hour_counts = df['Departure.Hour'].value_counts().sort_index()


plt.figure(figsize=(12, 6))

plt.bar(hour_counts.index, hour_counts.values)
```

```
plt.title('Most Frequent Departure Hours')

plt.xlabel('Hour of Day')

plt.ylabel('Frequency')

plt.xticks(range(0, 24))

plt.show()
```

Figure – 7

```
df['Departure.Hour'] = pd.to_datetime(df['Departure'], errors='coerce').dt.hour

#Most frequent hours for departures based on stations

station_hour_counts = df.groupby(['Departure.Station',
'Departure.Hour']).size().reset_index(name='Count')

heatmap_data = station_hour_counts.pivot(index='Departure.Station', columns='Departure.Hour',
values='Count').fillna(0)

plt.figure(figsize=(12, 8))

sns.heatmap(heatmap_data, cmap='YlOrBr', linewidths=0.5)

plt.title('Station Departure Counts by Hour')

plt.xlabel('Time (24 hour clock)')

plt.ylabel('Station')

plt.show()
```

Figure – 8

```
fig, axes = plt.subplots(1, 2, figsize=(16, 8), gridspec_kw={'width_ratios': [1, 2]})

#Pie chart for percentage of refund requests

refund_requests = df['Refund.Request'].value_counts(normalize=True) * 100

axes[0].pie(refund_requests, labels=refund_requests.index, autopct='%1.1f%%', startangle=140,
colors=['lightcoral', 'skyblue'])

axes[0].set_title('Refund Request Distribution')

#Bar chart for refund request - journey status relation

refund_status_relation = df.groupby(['Journey.Status', 'Refund.Request']).size().unstack().fillna(0)
```

```
refund_status_percentage = refund_status_relation.div(refund_status_relation.sum(axis=1), axis=0) *
100

refund_status_percentage.plot(kind='bar', stacked=True, color=['lightcoral', 'skyblue'], ax=axes[1])

axes[1].set_title('Refund Requests by Journey Status')

axes[1].set_xlabel('Journey Status')

axes[1].set_ylabel('Percentage of Refund Requests (%)')

axes[1].legend(title='Refund Requested', labels=['No', 'Yes'])

axes[1].tick_params(axis='x', rotation=45)

plt.tight_layout()

plt.show()
```

Figure – 9

```
#Heatmap for refund requests by journey status

plt.figure(figsize=(8, 6))

sns.heatmap(refund_status_relation, annot=True, cmap='YlGnBu', fmt=".0f", linewidths=0.5)

plt.title('Refund Requests by Journey Status')

plt.xlabel('Refund Requested')

plt.ylabel('Journey Status')

plt.show()
```

Figure – 10

```
#Price distribution according to ticket type

ticket_type_price = df[['Ticket.Type', 'Price']].dropna()

plt.figure(figsize=(12, 6))

sns.boxplot(data=ticket_type_price, x='Ticket.Type', y='Price',  palette='Set1')

plt.title('Price Distribution by Ticket Type')

plt.xlabel('Ticket Type')

plt.ylabel('Price (£)')

plt.xticks(rotation=45)
```

```
plt.show()
```

DELAY IN MINUTES

Figure – 11

```
df['DelayInMinutes'] = None
#Convert datees columns to datetime objects
df['Scheduled.Arrival'] = pd.to_datetime(df['Scheduled.Arrival'])
df['Actual.Arrival'] = pd.to_datetime(df['Actual.Arrival'])
#Calculate the delay in minutes
df['DelayInMinutes'] = (df['Actual.Arrival'] - df['Scheduled.Arrival']).dt.total_seconds() / 60
#Setting values to NA if it is not delayed
df.loc[df['DelayInMinutes'] <= 0, 'DelayInMinutes'] = pd.NA
df.head()
```

MEDIUM PRICE - UNIVARIATE REGRESSION

Figure – 12

```
filtered = df[df['Journey.Status'] != 'On Time']
filtered.head()
filtered['MediumPrice'] =  (filtered['Price'] > 10) & (filtered['Price'] <= 30)
print(filtered[['Price', 'MediumPrice','Refund.Request']].head(5))
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
filtered['Refund.Request'] = filtered['Refund.Request'].map({'Yes': 1, 'No': 0})
#Dependent and independent variables
X = filtered[['MediumPrice']]
y = filtered['Refund.Request']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#Logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
#Model prediction and test
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
#Calculation for £5 and £25
test_data = pd.DataFrame({'MediumPrice': [(5 > 10) & (5 <= 30), (25 > 10) & (25 <= 30)]})
#Prediction of probabilities
refund_probabilities = model.predict_proba(test_data)
print(f"Probability of requesting a refund for £5 ticket: {refund_probabilities[0][1]:.2f}")
print(f"Probability of requesting a refund for £25 ticket: {refund_probabilities[1][1]:.2f}")
model.intercept_
model.coef_
```

REGRESSION MODEL FOR REFUND PREDICTION

Figure – 14/15/16

```
predict_df = pd.read_csv("/Users/emrec/Desktop/SML Assignment/ToPredict.csv")
predict_df.head(8)
#Having a dataframe for just refund request = yes values
yes_df = df[df['Refund.Request'] == 'Yes']
total_yes = yes_df.shape[0]
print(total_yes)
#Having a dataframe for just refund request = no values
no_df = df[df['Refund.Request'] == 'No']
no_df.shape[0]
#Having the first 1114 data to have equal number of refund request = yes and refund request = no cases
new_no = no_df.head(1114)
#Combining and shuffling the 2 dataframes so that we will have a new dataset to train our model
```

```
#to get rid of the bias, since original data has way more refund request = no cases

combined_df = pd.concat([yes_df, new_no])

shuffled_df = combined_df.sample(frac=1, random_state=42).reset_index(drop=True)

shuffled_df.shape[0]

X = pd.get_dummies(shuffled_df[['Price','Journey.Status']], drop_first=True)

y = shuffled_df['Refund.Request']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify = y, random_state=42)

lm_model = LogisticRegression(max_iter=1000, random_state=42)

lm_model.fit(X_train, y_train)

y_pred = lm_model.predict(X_test)

print(classification_report(y_test, y_pred))

predict_df = pd.get_dummies(predict_df, drop_first=True)

missing_cols = set(X_train.columns) - set(predict_df.columns)

for col in missing_cols:

    predict_df[col] = 0

predict_df = predict_df[X_train.columns]

refund_probabilities = lm_model.predict_proba(predict_df)[:, 1]  # İade talebi olasılığı

predict_df['Refund.Probability'] = refund_probabilities

print(predict_df[['Refund.Probability']].head(8))

threshold = 0.5

predict_df['Refund'] = predict_df['Refund.Probability'].apply(lambda prob: 'Yes' if prob >= threshold else
'No')

predict_df.sort_values(by='Refund.Probability', ascending=False)

from sklearn.metrics import confusion_matrix

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()

#End of the Notebook

SCREENSHOTS OF THE NOTEBOOK

1. EDA

[1]:
```
#Data process and analysis libraries
import pandas as pd
import numpy as np

#Data visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
df = pd.read_csv("/Users/emrec/Desktop/SML Assignment/MavenRail.csv")
df.head()
```

[2]:

	Payment.Method	Railcard	Ticket.Class	Ticket.Type	Price	Departure.Station	Arrival.Station	Departure	Scheduled.Arrival	Actual.Arrival	Journey.Status	Reaso
0	Contactless	Adult	Standard	Advance	43	London Paddington	Liverpool Lime Street	2024-01-01 11:00	2024-01-01 13:30	2024-01-01 13:30	On Time	
1	Credit Card	Adult	Standard	Advance	23	London Kings Cross	York	2024-01-01 09:45	2024-01-01 11:35	2024-01-01 11:40	Delayed	S
2	Credit Card	NaN	Standard	Advance	3	Liverpool Lime Street	Manchester Piccadilly	2024-01-02 18:15	2024-01-02 18:45	2024-01-02 18:45	On Time	
3	Credit Card	NaN	Standard	Advance	13	London Paddington	Reading	2024-01-01 21:30	2024-01-01 22:30	2024-01-01 22:30	On Time	
4	Contactless	NaN	Standard	Advance	76	Liverpool Lime Street	London Euston	2024-01-01 16:45	2024-01-01 19:00	2024-01-01 19:00	On Time	

[5]:
```
row_count = len(df)
print(row_count)
```

```
31645
```

[7]:
```
null_numbers = df.isnull().sum()
print(null_numbers)
```

```
Payment.Method           0
Railcard             20911
Ticket.Class             0
Ticket.Type              0
Price                    0
Departure.Station        0
Arrival.Station          0
Departure                3
Scheduled.Arrival        4
Actual.Arrival        1880
Journey.Status           0
Reason.for.Delay     27479
Refund.Request           0
dtype: int64
```

[9]:
```
df.describe()
```

[9]:

	Price
count	31645.000000
mean	23.434634
std	29.990379
min	1.000000
25%	5.000000
50%	11.000000
75%	35.000000
max	267.000000

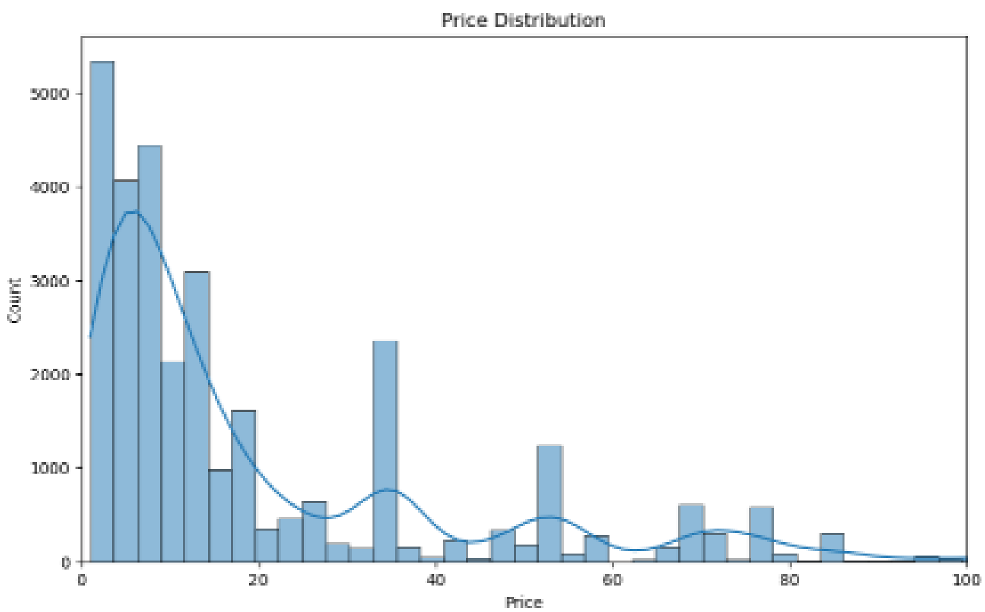
[11]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31645 entries, 0 to 31644
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Payment.Method     31645 non-null  object
 1   Railcard           10734 non-null  object
 2   Ticket.Class       31645 non-null  object
 3   Ticket.Type        31645 non-null  object
 4   Price              31645 non-null  int64 
 5   Departure.Station  31645 non-null  object
 6   Arrival.Station    31645 non-null  object
 7   Departure          31642 non-null  object
 8   Scheduled.Arrival  31641 non-null  object
 9   Actual.Arrival     29765 non-null  object
 10  Journey.Status     31645 non-null  object
 11  Reason.for.Delay   4166 non-null   object
 12  Refund.Request     31645 non-null  object
dtypes: int64(1), object(12)
memory usage: 3.1+ MB
```

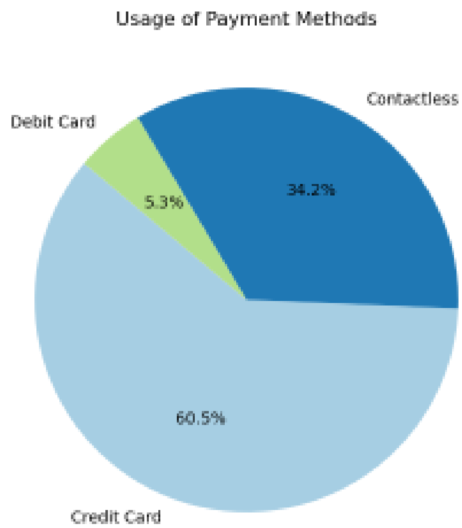
[13]:

```
#Price distribution and KDE
plt.figure(figsize=(10, 6))
sns.histplot(df['Price'], bins=100, kde=True)
plt.title('Price Distribution')
plt.xlim(0, 100)
plt.show()
```

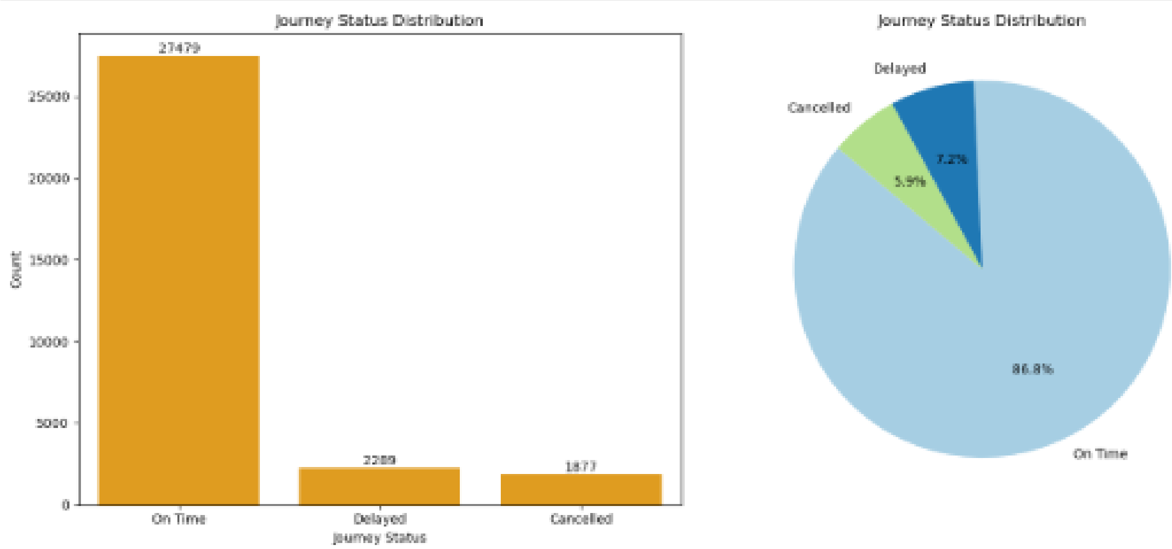


[14]:

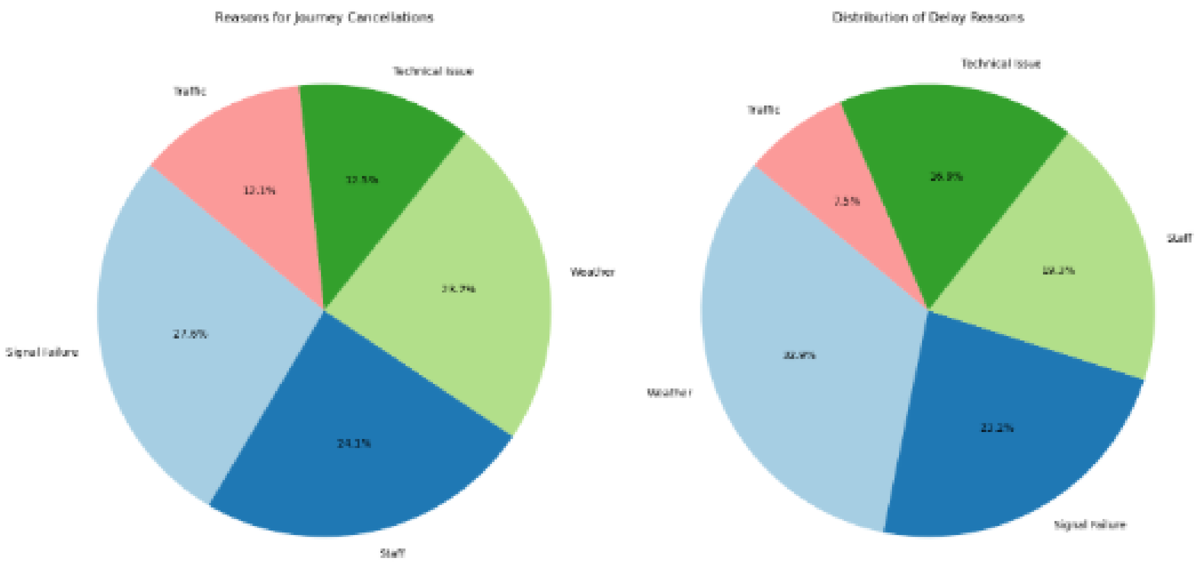
```
#Payment method distribution
payment_methods = df['Payment.Method'].value_counts()

#Payment method % chart
plt.figure(figsize=(8, 6))
payment_methods.plot(kind='pie', autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
plt.title('Usage of Payment Methods')
plt.ylabel('')
plt.show()
```



[15]:
```python
#Journey status distribution
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

#Bar chart for numbers
ax1 = sns.countplot(x='Journey.Status', data=df, color='orange', ax=axes[0])
ax1.set_title('Journey Status Distribution')
ax1.bar_label(ax1.containers[0])
ax1.set_xlabel('Journey Status')
ax1.set_ylabel('Count')

#Pie chart for percentage
journey_status_counts = df['Journey.Status'].value_counts()
axes[1].pie(journey_status_counts, labels=journey_status_counts.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
axes[1].set_title('Journey Status Distribution')

plt.tight_layout()
plt.show()
```



[16]:
```python
#In dataset, Staff related delays anc cancellations labeled as Staff and Staffing, so we put them together
df['Reason.for.Delay'] = df['Reason.for.Delay'].replace({'Staffing': 'Staff'})

#Calculating the delay reasons and cancellation reasons
delay_reasons = df['Reason.for.Delay'].value_counts()
cancelled_journeys = df[df['Journey.Status'] == 'Cancelled']
cancellation_reasons = cancelled_journeys['Reason.for.Delay'].value_counts()

fig, axes = plt.subplots(1, 2, figsize=(16, 8))

#Pie chart for cancellation reasons - distribution
axes[0].pie(cancellation_reasons, labels=cancellation_reasons.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
axes[0].set_title('Reasons for Journey Cancellations')

#Pie chart for delay reasons - distribution
axes[1].pie(delay_reasons, labels=delay_reasons.index, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired.colors)
axes[1].set_title('Distribution of Delay Reasons')

plt.tight_layout()
plt.show()
```



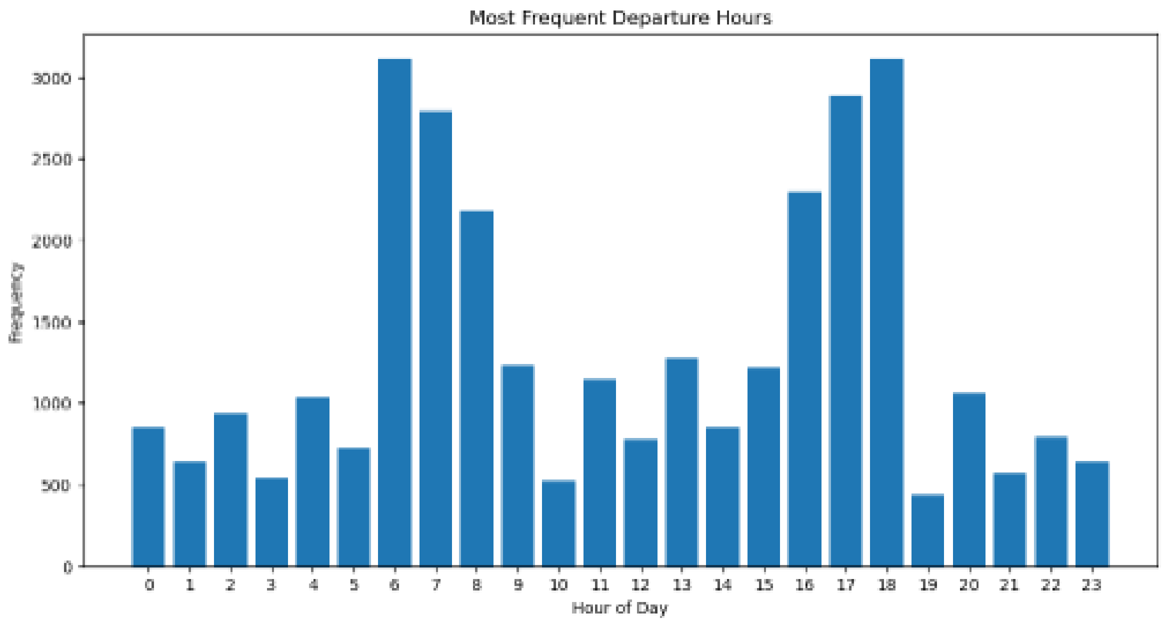
[17]:
```python
#Most used routes among the dataset
top_routes = df.groupby(['Departure.Station', 'Arrival.Station']).size().sort_values(ascending=False).head(10)
top_routes = top_routes.reset_index(name='Count')

top_routes
```

[17]:

	Departure.Station	Arrival.Station	Count
0	Manchester Piccadilly	Liverpool Lime Street	4626
1	London Euston	Birmingham New Street	4208
2	London Kings Cross	York	3922
3	London Paddington	Reading	3873
4	London St Pancras	Birmingham New Street	3470
5	Liverpool Lime Street	Manchester Piccadilly	3001
6	Liverpool Lime Street	London Euston	1096
7	London Euston	Manchester Piccadilly	712
8	Birmingham New Street	London St Pancras	701
9	London Paddington	Oxford	485

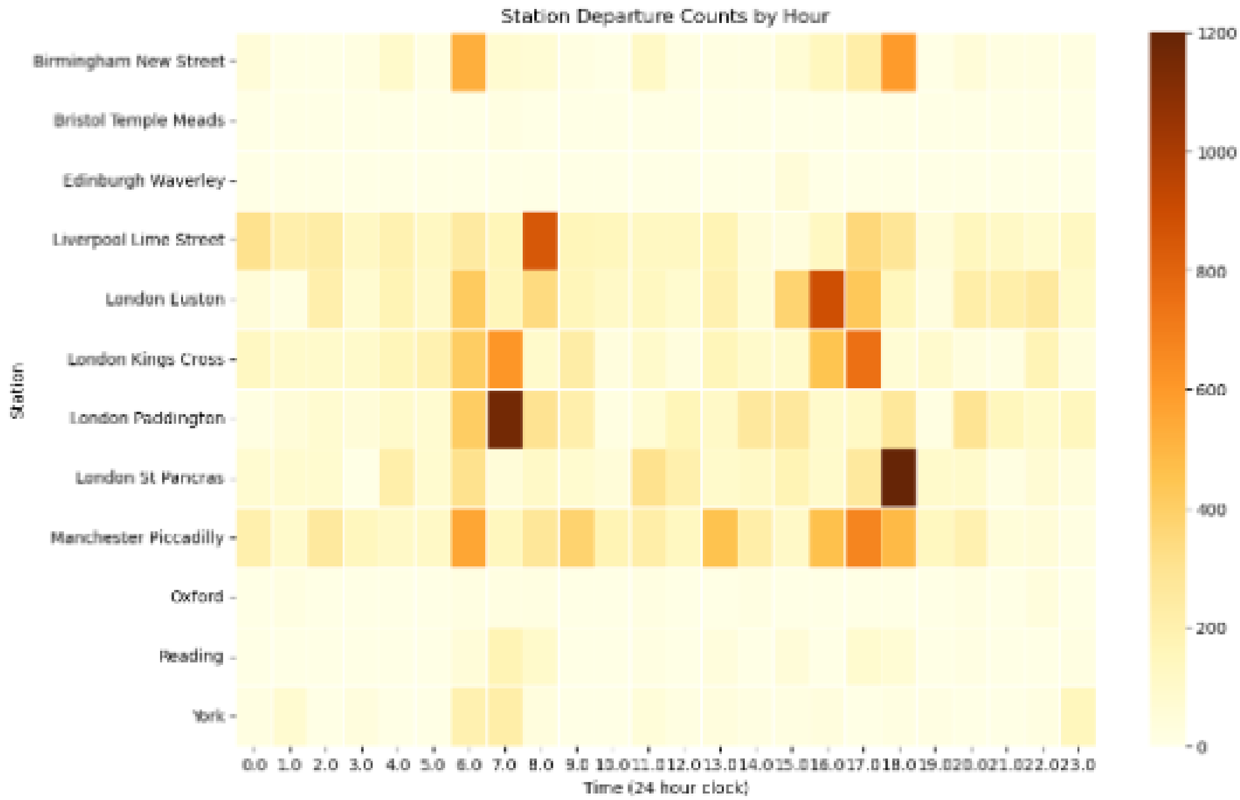
[18]:

```
#Most frequent departure hours
df['Departure'] = pd.to_datetime(df['Departure'], errors='coerce')

df['Departure.Hour'] = df['Departure'].dt.hour
hour_counts = df['Departure.Hour'].value_counts().sort_index()

plt.figure(figsize=(12, 6))
plt.bar(hour_counts.index, hour_counts.values)
plt.title('Most Frequent Departure Hours')
plt.xlabel('Hour of Day')
plt.ylabel('Frequency')
plt.xticks(range(0, 24))
plt.show()
```



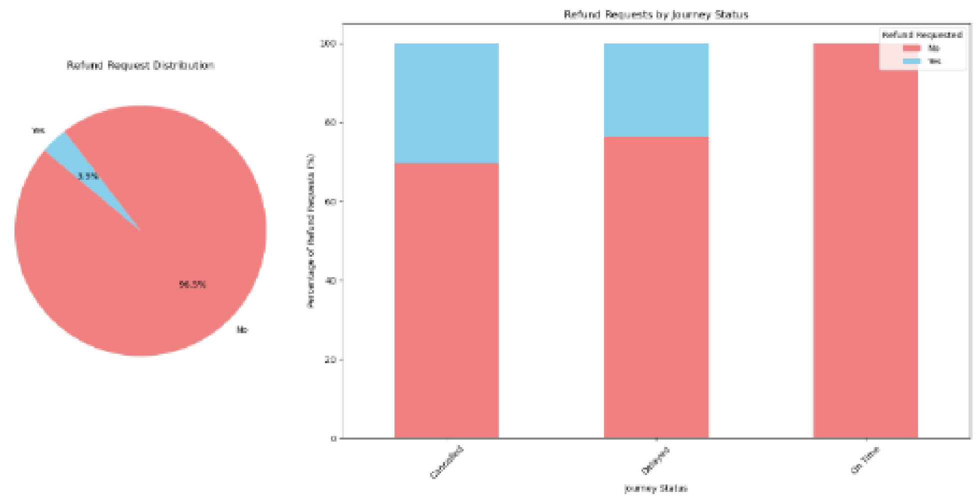
[22]:

```
df['Departure.Hour'] = pd.to_datetime(df['Departure'], errors='coerce').dt.hour

#Most frequent hours for departures based on stations
station_hour_counts = df.groupby(['Departure.Station', 'Departure.Hour']).size().reset_index(name='Count')

heatmap_data = station_hour_counts.pivot(index='Departure.Station', columns='Departure.Hour', values='Count').fillna(0)

plt.figure(figsize=(12, 8))
sns.heatmap(heatmap_data, cmap='YlOrBr', linewidths=0.5)
plt.title('Station Departure Counts by Hour')
plt.xlabel('Time (24 hour clock)')
plt.ylabel('Station')
plt.show()
```



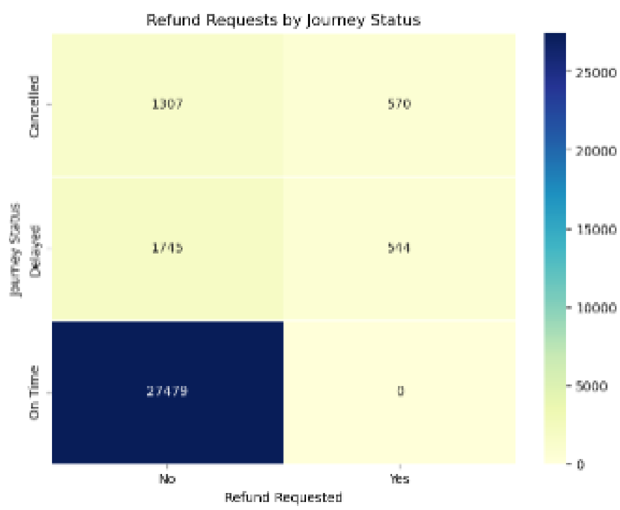
[23]:

```
fig, axes = plt.subplots(1, 2, figsize=(16, 8), gridspec_kw={'width_ratios': [1, 2]})

#Pie chart for percentage of refund requests
refund_requests = df['Refund.Request'].value_counts(normalize=True) * 100
axes[0].pie(refund_requests, labels=refund_requests.index, autopct='%1.1f%%', startangle=140, colors=['lightcoral', 'skyblue'])
axes[0].set_title('Refund Request Distribution')

#Bar chart for refund request - journey status relation
refund_status_relation = df.groupby(['Journey.Status', 'Refund.Request']).size().unstack().fillna(0)
refund_status_percentage = refund_status_relation.div(refund_status_relation.sum(axis=1), axis=0) * 100
refund_status_percentage.plot(kind='bar', stacked=True, color=['lightcoral', 'skyblue'], ax=axes[1])
axes[1].set_title('Refund Requests by Journey Status')
axes[1].set_xlabel('Journey Status')
axes[1].set_ylabel('Percentage of Refund Requests (%)')
axes[1].legend(title='Refund Requested', labels=['No', 'Yes'])
axes[1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```



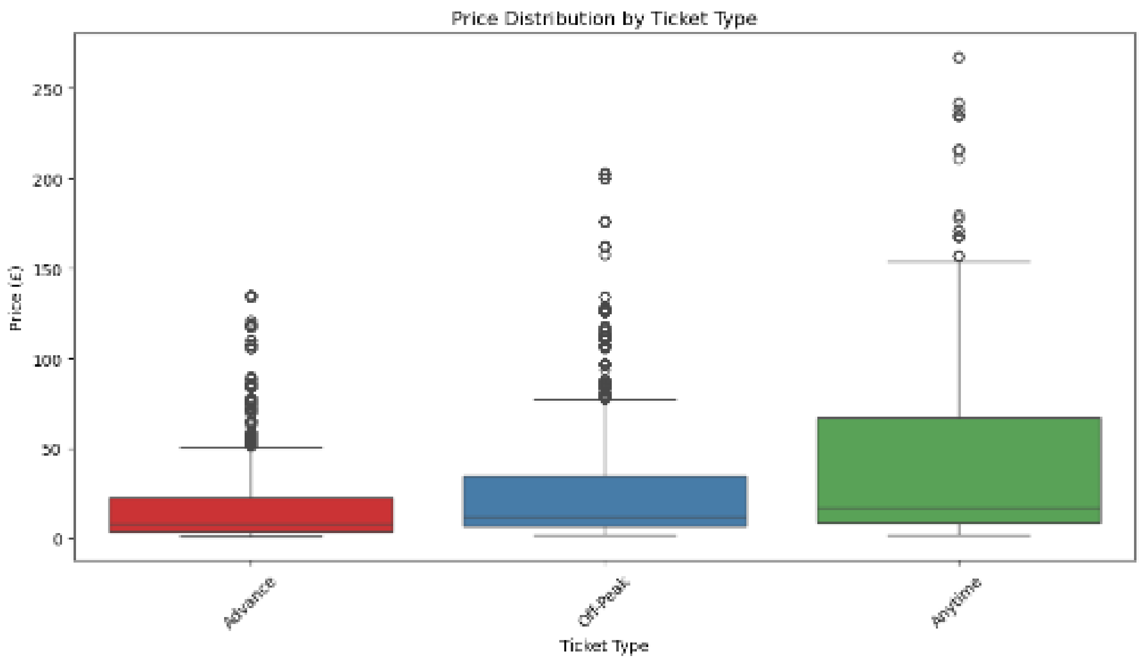
[24]:

```
#Heatmap for refund requests by journey status
plt.figure(figsize=(8, 6))
sns.heatmap(refund_status_relation, annot=True, cmap='YlGnBu', fmt=".0f", linewidths=0.5)
plt.title('Refund Requests by Journey Status')
plt.xlabel('Refund Requested')
plt.ylabel('Journey Status')
plt.show()
```



[26]:

```
#Price distribution according to ticket type
ticket_type_price = df[['Ticket.Type', 'Price']].dropna()

plt.figure(figsize=(12, 6))
sns.boxplot(data=ticket_type_price, x='Ticket.Type', y='Price',  palette='Set1')
plt.title('Price Distribution by Ticket Type')
plt.xlabel('Ticket Type')
plt.ylabel('Price (£)')
plt.xticks(rotation=45)
plt.show()
```



2. Delay In Minutes

[31]:
```python
df['DelayInMinutes'] = None

#Convert datees columns to datetime objects
df['Scheduled.Arrival'] = pd.to_datetime(df['Scheduled.Arrival'])
df['Actual.Arrival'] = pd.to_datetime(df['Actual.Arrival'])

#Calculate the delay in minutes
df['DelayInMinutes'] = (df['Actual.Arrival'] - df['Scheduled.Arrival']).dt.total_seconds() / 60

#Setting values to NA if it is not delayed
df.loc[df['DelayInMinutes'] <= 0, 'DelayInMinutes'] = pd.NA

df.head()
```

[31]:

	Payment.Method	Railcard	Ticket.Class	Ticket.Type	Price	Departure.Station	Arrival.Station	Departure	Scheduled.Arrival	Actual.Arrival	Journey.Statu
0	Contactless	Adult	Standard	Advance	43	London Paddington	Liverpool Lime Street	2024-01-01 11:00:00	2024-01-01 13:30:00	2024-01-01 13:30:00	On Time
1	Credit Card	Adult	Standard	Advance	23	London Kings Cross	York	2024-01-01 09:45:00	2024-01-01 11:35:00	2024-01-01 11:40:00	Delayed
2	Credit Card	NaN	Standard	Advance	3	Liverpool Lime Street	Manchester Piccadilly	2024-01-02 18:15:00	2024-01-02 18:45:00	2024-01-02 18:45:00	On Time
3	Credit Card	NaN	Standard	Advance	13	London Paddington	Reading	2024-01-01 21:30:00	2024-01-01 22:30:00	2024-01-01 22:30:00	On Time

3. Medium Price

[34]:
```
filtered = df[df['Journey.Status'] != 'On Time']
filtered.head()
```

[34]:

	Payment.Method	Railcard	Ticket.Class	Ticket.Type	Price	Departure.Station	Arrival.Station	Departure	Scheduled.Arrival	Actual.Arrival	Journey.Status	Reas
1	Credit Card	Adult	Standard	Advance	23	London Kings Cross	York	2024-01-01 09:45:00	2024-01-01 11:35:00	2024-01-01 11:40:00	Delayed	
8	Credit Card	NaN	Standard	Advance	37	London Euston	York	2024-01-01 00:00:00	2024-01-01 01:50:00	2024-01-01 02:07:00	Delayed	
20	Debit Card	Adult	Standard	Advance	7	Birmingham New Street	Manchester Piccadilly	2024-01-01 11:15:00	2024-01-01 12:35:00	2024-01-01 13:06:00	Delayed	Tı
26	Credit Card	Senior	First Class	Advance	34	Oxford	Bristol Temple Meads	2024-01-01 14:15:00	2024-01-01 15:30:00	2024-01-01 15:54:00	Delayed	
39	Credit Card	NaN	Standard	Advance	7	London Euston	Birmingham New Street	2024-01-02 02:15:00	2024-01-02 03:35:00	NaT	Cancelled	Tı

[35]:
```
filtered['MediumPrice'] =  (filtered['Price'] > 10) & (filtered['Price'] <= 30)

print(filtered[['Price', 'MediumPrice','Refund.Request']].head(5))
```

```
    Price  MediumPrice Refund.Request
1      23         True             No
8      37        False             No
20      7        False            Yes
26     34        False            Yes
39      7        False             No
```

```
C:\Users\emrec\AppData\Local\Temp\ipykernel_82612\3580612988.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered['MediumPrice'] =  (filtered['Price'] > 10) & (filtered['Price'] <= 30)
```

[37]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

[38]:
```
filtered['Refund.Request'] = filtered['Refund.Request'].map({'Yes': 1, 'No': 0})
```

```
C:\Users\emrec\AppData\Local\Temp\ipykernel_82612\2282492401.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  filtered['Refund.Request'] = filtered['Refund.Request'].map({'Yes': 1, 'No': 0})
```

[40]:
```
#Dependent and Independent variables
X = filtered[['MediumPrice']]
y = filtered['Refund.Request']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

#Model prediction and test
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      1.00      0.85       615
           1       0.00      0.00      0.00       219

    accuracy                           0.74       834
   macro avg       0.37      0.50      0.42       834
weighted avg       0.54      0.74      0.63       834
```

```
#Calculation for £5 and £25
test_data = pd.DataFrame({'MediumPrice': [(5 > 10) & (5 <= 30), (25 > 10) & (25 <= 30)]})


#Prediction of probabilities
refund_probabilities = model.predict_proba(test_data)


print(f"Probability of requesting a refund for £5 ticket: {refund_probabilities[0][1]:.2f}")
print(f"Probability of requesting a refund for £25 ticket: {refund_probabilities[1][1]:.2f}")
```

```
Probability of requesting a refund for £5 ticket: 0.26
Probability of requesting a refund for £25 ticket: 0.32
```

```
model.intercept_
```

```
array([-1.05979802])
```

```
model.coef_
```

```
array([[0.38087428]])
```

4. Regression

[46]:
```
predict_df = pd.read_csv("/Users/emrec/Desktop/SML Assignment/ToPredict.csv")
predict_df.head(8)
```

[46]:

	Payment.Method	Railcard	Ticket.Class	Ticket.Type	Price	Departure.Station	Arrival.Station	Departure	Scheduled.Arrival	Actual.Arrival	Journey.Status	Reaso
0	Debit Card	NaN	First Class	Advance	54	London St Pancras	Birmingham New Street	2024-01-04 17:45	2024-01-04 19:05	2024-01-04 19:05	On Time	
1	Credit Card	NaN	Standard	Advance	7	London Euston	Birmingham New Street	2024-01-05 08:15	2024-01-05 09:35	2024-01-05 09:35	On Time	
2	Debit Card	NaN	Standard	Off-Peak	113	Liverpool Lime Street	London Euston	2024-01-09 15:30	2024-01-09 17:45	2024-01-09 18:07	Delayed	
3	Contactless	Adult	Standard	Off-Peak	3	Liverpool Lime Street	Manchester Piccadilly	2024-01-31 05:45	2024-01-31 06:15	2024-01-31 06:49	Delayed	S
4	Credit Card	NaN	Standard	Off-Peak	4	Manchester Piccadilly	Liverpool Lime Street	2024-02-10 16:00	2024-02-10 16:30	NaN	Cancelled	Te
5	Contactless	NaN	Standard	Advance	3	Manchester Piccadilly	Liverpool Lime Street	2024-02-25 15:45	2024-02-25 16:15	NaN	Cancelled	
6	Debit Card	NaN	Standard	Off-Peak	126	Manchester Piccadilly	London Euston	2024-03-20 15:30	2024-03-20 17:20	2024-03-20 17:36	Delayed	
7	Credit Card	NaN	Standard	Advance	22	Birmingham New Street	London St Pancras	2024-04-16 04:30	2024-04-16 05:50	NaN	Cancelled	S

[48]:
```
#Having a dataframe for just refund request = yes values
yes_df = df[df['Refund.Request'] == 'Yes']
```

[60]:
```
total_yes = yes_df.shape[0]
```

[62]:
```
print(total_yes)
```

1114

[64]:
```
#Having a dataframe for just refund request = no values
no_df = df[df['Refund.Request'] == 'No']
```

[66]:
```
no_df.shape[0]
```

[66]: 30531

[68]:
```
#Having the first 1114 data to have equal number of refund request = yes and refund request = no cases
new_no = no_df.head(1114)
```

[70]:
```
new_no.shape[0]
```

[70]: 1114

[72]:
```
#Combining and shuffling the 2 dataframes so that we will have a new dataset to train our model
#to get rid of the bias, since original data has way more refund request = no cases
combined_df = pd.concat([yes_df, new_no])

shuffled_df = combined_df.sample(frac=1, random_state=42).reset_index(drop=True)
```

[74]:
```
shuffled_df.shape[0]
```

[74]: 2228

[76]:
```
X = pd.get_dummies(shuffled_df[['Price','Journey.Status']], drop_first=True)
y = shuffled_df['Refund.Request']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify = y, random_state=42)
```

[78]:
```
lm_model = LogisticRegression(max_iter=1000, random_state=42)
lm_model.fit(X_train, y_train)

y_pred = lm_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          No       1.00      0.91      0.96       223
         Yes       0.92      1.00      0.96       223

    accuracy                           0.96       446
   macro avg       0.96      0.96      0.96       446
weighted avg       0.96      0.96      0.96       446
```

[80]:
```python
predict_df = pd.get_dummies(predict_df, drop_first=True)

missing_cols = set(X_train.columns) - set(predict_df.columns)
for col in missing_cols:
    predict_df[col] = 0

predict_df = predict_df[X_train.columns]

refund_probabilities = lm_model.predict_proba(predict_df)[:, 1]  # İade talebi olasılığı

predict_df['Refund.Probability'] = refund_probabilities

print(predict_df[['Refund.Probability']].head(8))
```

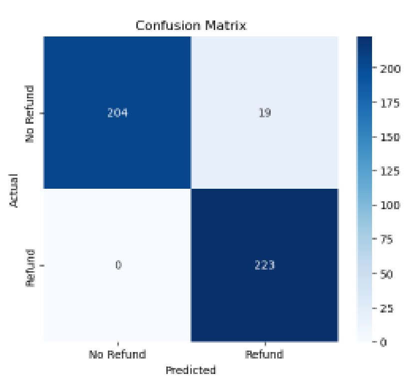
```
   Refund.Probability
0            0.006090
1            0.009763
2            0.857410
3            0.948283
4            0.920594
5            0.921330
6            0.840556
7            0.906214
```

[82]:
```python
threshold = 0.5
predict_df['Refund'] = predict_df['Refund.Probability'].apply(lambda prob: 'Yes' if prob >= threshold else 'No')
predict_df.sort_values(by='Refund.Probability', ascending=False)
```

[82]:

	Price	Journey.Status_Delayed	Journey.Status_On Time	Refund.Probability	Refund
3	3	True	False	0.948283	Yes
5	3	False	False	0.921330	Yes
4	4	False	False	0.920594	Yes
7	22	False	False	0.906214	Yes
2	113	True	False	0.857410	Yes
6	126	True	False	0.840556	Yes
1	7	False	True	0.009763	No
0	54	False	True	0.006090	No

[84]:
```python
from sklearn.metrics import confusion_matrix
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()
```



[92]:
```python
#End of the Notebook
```

7. References

- Maven Analytics. "*Maven Rail Challenge Dataset*." Accessed September 30, 2024. https://mavenanalytics.io/challenges/maven-rail-challenge/08941141-d23f-4cc9-93a3-4c25ed06e1c3.

- Python Software Foundation. "*Dataframe-image*." PyPI. Accessed November 12, 2024. https://pypi.org/project/dataframe-image/.

- Shah, D. "Logistic Regression: What It Is and How It Works." *V7.* Accessed November 15, 2024. https://www.v7labs.com/blog/logistic-regression.

- Şener, Y. “Veri Biliminde Kategorik Değişkenler, Dummy (Kukla) Variable ve Python Uygulaması.” Accessed November 15, 2024. https://yigitsener.medium.com/veri-biliminde-kategorik-de%C4%9Fi%C5%9Fkenler-dummy-kukla-variable-ve-python-uygulamas%C4%B1-e086cb9a4f71