In [3]:

```python
# Importing essential libraries

# Data manipulation and analysis
import pandas as pd
import numpy as np

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

# Statistical modeling
import statsmodels.api as sm

# Data preprocessing
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Missing value handling
from sklearn.impute import SimpleImputer

# Encoding categorical variables
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Visualization for categorical variables
import plotly.express as px

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

importing the dataset and displaying the first 5 rows

In [6]:

```python
import pandas as pd

# Replace 'my_data.csv' with the actual name of your CSV file
rail = pd.read_csv('railway.csv')

# Display the first 5 rows of the DataFrame
```

In [8]:

```python
rail.head()
```

Out[8]:

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	Ticket Type
0	da8a6ba8-b3dc-4677-b176	2023-12-08	12:41:11	Online	Contactless	Adult	Standard	Advance
1	b0cdd1b0-f214-4197-be53	2023-12-16	11:23:01	Station	Credit Card	Adult	Standard	Advance
2	f3ba7a96-f713-40d9-9629	2023-12-19	19:51:27	Online	Credit Card	NaN	Standard	Advance
3	b2471f11-4fe7-4c87-8ab4	2023-12-20	23:00:36	Station	Credit Card	NaN	Standard	Advance
4	2be00b45-0762-485e-a7a3	2023-12-27	18:22:56	Online	Contactless	NaN	Standard	Advance

In [12]:

```
# last 5 rows
rail.tail()
```

Out[12]:

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	Tick Typ
31648	1304623d-b8b7-4999-8e9c	2024-04-30	18:42:58	Online	Credit Card	NaN	Standard	O Pe
31649	7da22246-f480-417c-bc2f	2024-04-30	18:46:10	Online	Contactless	NaN	Standard	O Pe
31650	add9debf-46c1-4c75-b52d	2024-04-30	18:56:41	Station	Credit Card	NaN	Standard	O Pe
31651	b92b047c-21fd-4859-966a	2024-04-30	19:51:47	Station	Credit Card	NaN	Standard	O Pe
31652	1d5d89a2-bde5-410f-8f91	2024-04-30	20:05:39	Station	Credit Card	Adult	Standard	O Pe

checking for duplicates, (there's none)

In [15]:

```
counts = rail['Transaction ID'].value_counts()

duplicate_values = counts[counts > 1].index

print(duplicate_values)
```

```
Index([], dtype='object', name='Transaction ID')
```

In [17]:

```
rail.duplicated().sum()  # Count of duplicate rows
```

Out[17]: 0

First : understanding the structure of the dataset and fixing potential problems

In [20]:

```
rail.describe(include='all')
```

Out[20]:

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	Tick Typ
count	31653	31653	31653	31653	31653	10735	31653	3165
unique	31653	128	24351	2	3	3	2	
top	da8a6ba8-b3dc-4677-b176	2024-02-02	8:16:53	Online	Credit Card	Adult	Standard	Advanc
freq	1	513	6	18521	19136	4846	28595	1756
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

In [22]:

```
rail.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31653 entries, 0 to 31652
Data columns (total 18 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Transaction ID       31653 non-null  object
 1   Date of Purchase     31653 non-null  object
 2   Time of Purchase     31653 non-null  object
 3   Purchase Type        31653 non-null  object
 4   Payment Method       31653 non-null  object
 5   Railcard             10735 non-null  object
 6   Ticket Class         31653 non-null  object
 7   Ticket Type          31653 non-null  object
 8   Price                31653 non-null  int64 
 9   Departure Station    31653 non-null  object
 10  Arrival Destination  31653 non-null  object
 11  Date of Journey      31653 non-null  object
 12  Departure Time       31653 non-null  object
 13  Arrival Time         31653 non-null  object
 14  Actual Arrival Time  29773 non-null  object
 15  Journey Status       31653 non-null  object
 16  Reason for Delay     4172 non-null   object
 17  Refund Request       31653 non-null  object
dtypes: int64(1), object(17)
memory usage: 4.3+ MB
```

the Railcard column has null values because pandas interpret "None" as Null so we will fix it to "No Railcard

```
In [31]: rail['Railcard'].fillna('No Railcard', inplace=True)
```

```
In [33]: # Quick check
         rail.isnull().sum()
```

```
Out[33]: Transaction ID             0
         Date of Purchase           0
         Time of Purchase           0
         Purchase Type              0
         Payment Method             0
         Railcard                   0
         Ticket Class               0
         Ticket Type                0
         Price                      0
         Departure Station          0
         Arrival Destination        0
         Date of Journey            0
         Departure Time             0
         Arrival Time               0
         Actual Arrival Time     1880
         Journey Status             0
         Reason for Delay       27481
         Refund Request             0
         dtype: int64
```

the null values in 'Reason for Delay' was only recorded when the train arrived on time, to check this

In [36]:

```
def analyze_on_time_journeys(df):
    """
    Analyzes records where Journey Status is "On Time" and
    returns the count and unique values of Reason for Delay.
    """

    # Filter for "On Time" journeys
    on_time_df = rail[rail['Journey Status'] == 'On Time']

    # Count the number of "On Time" journeys
    on_time_count = len(on_time_df)

    # Get unique values of "Reason for Delay" for "On Time" journeys
    reasons_for_delay = on_time_df['Reason for Delay'].unique()

    return on_time_count, reasons_for_delay

# Example Usage
on_time_count, reasons_for_delay = analyze_on_time_journeys(rail)

print(f"Number of 'On Time' journeys: {on_time_count}")
print(f"Unique 'Reason for Delay' values for 'On Time' journeys: {reasons_
```

```
Number of 'On Time' journeys: 27481
Unique 'Reason for Delay' values for 'On Time' journeys: [nan]
```

to fix this we'll just set them to 'No Delay'

In [39]:

```
rail['Reason for Delay'].fillna('No Delay', inplace=True)
```

now let's check the unique values for multiple columns

In [69]:

```
def display_unique_values_filtered(rail, columns_to_check):
    """Displays unique values, filtering out rows with all NaNs."""

    unique_values_dict = {}
    for column in columns_to_check:
        unique_vals = rail[column].unique()
        unique_values_dict[column] = unique_vals

    data = {}
    for column in columns_to_check:
        data[column] = pd.Series(unique_values_dict[column])

    unique_df = pd.DataFrame(data)

    # Filter out rows where all specified columns are NaN
    filtered_df = unique_df.dropna(how='all')

    # Replace NaN values with an empty string
    filtered_df = filtered_df.fillna('')

    print(filtered_df)

# Example Usage
columns_to_check = ['Purchase Type', 'Payment Method', 'Railcard', 'Ticke
display_unique_values_filtered(rail, columns_to_check)
```

```
  Purchase Type Payment Method     Railcard Ticket Class Ticket Type  \
0        Online    Contactless        Adult     Standard     Advance   
1       Station    Credit Card  No Railcard  First Class    Off-Peak   
2                   Debit Card     Disabled                  Anytime   
3                                    Senior                            
4                                                                      
5                                                                      
6                                                                      
7                                                                      
8                                                                      

  Journey Status    Reason for Delay Refund Request  
0        On Time            No Delay             No  
1        Delayed      Signal Failure            Yes  
2      Cancelled     Technical Issue                 
3                 Weather Conditions                 
4                            Weather                 
5                           Staffing                 
6                     Staff Shortage                 
7                     Signal failure                 
8                            Traffic                 
```

we noticed that 'Reason for Delay' still needs to be standardized

In [72]:

```python
rail['Reason for Delay'].value_counts()   # before
```

Out[72]:

```
Reason for Delay
No Delay              27481
Weather                 995
Technical Issue         707
Signal Failure          523
Signal failure          447
Staffing                410
Staff Shortage          399
Weather Conditions      377
Traffic                 314
Name: count, dtype: int64
```

In [74]:

```python
rail['Reason for Delay'].replace({
    'Signal failure': 'Signal Failure',
    'Staffing': 'Staff Shortage',
    'Weather': 'Weather Conditions'
}, inplace=True)

rail['Reason for Delay'].value_counts()  # After
```

Out[74]:

```
Reason for Delay
No Delay              27481
Weather Conditions     1372
Signal Failure          970
Staff Shortage          809
Technical Issue         707
Traffic                 314
Name: count, dtype: int64
```

converting the Time and date columns to their actual data type for better analysis

In [77]:

```
time_columns = ['Time of Purchase', 'Departure Time', 'Arrival Time', 'Ac

for column in time_columns:
    rail[column] = pd.to_datetime(rail[column], format='%H:%M:%S').dt.tim
```

In [79]:

```
date_columns = ['Date of Purchase', 'Date of Journey']

for column in date_columns:
    rail[column] = pd.to_datetime(rail[column], format='%Y-%m-%d').dt.dat
```

Second : Looking for Inconsistencies or Discrepancies across the columns

instances when the train arrived on time (Arrival Time = Actual Arrival Time) but was recorded as "Delayed"

In [87]:

```
mis_enteries = rail[
    (rail['Arrival Time'] == rail['Actual Arrival Time']) &
    (rail['Journey Status'] == 'Delayed')
]

mis_enteries
```

Out[87]:

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	T
10633	f10dc9f2-80c3-4b9f-8b72	2024-02-06	05:01:05	Station	Credit Card	No Railcard	Standard	Adv
13933	add29bde-e183-426a-adca	2024-02-15	15:01:47	Station	Debit Card	No Railcard	Standard	
15130	3d6c240e-5c33-4665-9144	2024-02-21	11:54:54	Station	Debit Card	Adult	First Class	Adv
16274	2b2bf794-2111-44bf-8758	2024-03-03	10:45:53	Online	Debit Card	Adult	Standard	Adv
16483	bd082832-41f9-4364-a8d2	2024-03-04	07:46:54	Online	Debit Card	Senior	Standard	
16488	73bc8893-5e5f-47c6-951b	2024-03-04	07:56:08	Online	Contactless	Senior	Standard	
16868	97203c12-be97-4199-8ac0	2024-03-05	16:11:29	Station	Contactless	Adult	Standard	Any
16879	3d6779a3-1206-4b3b-872f	2024-03-05	17:07:35	Station	Debit Card	Adult	Standard	Any
18927	9fe75f16-a67a-4d45-9c92	2024-03-13	04:19:37	Station	Debit Card	Senior	Standard	
22975	9479bec9-2e01-4aac-be28	2024-03-28	05:09:54	Station	Credit Card	No Railcard	Standard	Adv
23128	1923b77a-c469-41e7-98ea	2024-03-28	17:14:18	Station	Debit Card	Adult	Standard	Any
25003	c6a831e2-45a2-4089-8161	2024-04-06	02:01:10	Station	Credit Card	No Railcard	Standard	
25740	bfea5b54-7877-4ab1-9fed	2024-04-08	17:13:59	Station	Debit Card	Adult	Standard	Any
27923	441924c9-c008-4102-8b1d	2024-04-16	17:11:47	Station	Debit Card	Adult	Standard	Any
30495	cacaaff8-cede-4f77-9ae1	2024-04-26	06:05:42	Online	Credit Card	Disabled	Standard	Any
30739	1f6f2747-3b49-40f4-a159	2024-04-27	04:58:52	Station	Contactless	No Railcard	Standard	

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	T
30740	8a62b6cd-d298-420c-a4fa	2024-04-27	04:59:38	Online	Debit Card	No Railcard	Standard	
30866	67488422-ff65-46f9-b35b	2024-04-27	15:13:00	Station	Debit Card	No Railcard	Standard	

Two options : Delete Or Update, remove the # to execute

In [ ]:

```
#rail = rail[
    ~((rail['Arrival Time'] == rail['Actual Arrival Time']) &
      (rail['Journey Status'] == 'Delayed'))
]
```

In [ ]:

```
#railway.loc[
    (rail['Arrival Time'] == rail['Actual Arrival Time']) &
    (rail['Journey Status'] == 'Delayed'),
    ['Journey Status', 'Reason for Delay', 'Refund Request']
] = ['On Time', 'No Delay', 'No']
```

people with Senior, Disabled and Adult Railcards should pay the same fare for the same journey (same Departure Station, Arrival Destination, Date of Journey, Departure Time) 2/3 of the fare that someone with no railcards pays, holding Ticket Class and Ticket Type constant so we will check that

In [93]:

```
def find_price_discrepancies(rail):
    """
    Finds records where 'Price' is different for the same train (same
    Ticket Class, Ticket Type, Departure Station, Arrival Destination,
    Date of Journey, Departure Time) when Railcard is 'Senior', 'Disabled
    or 'Adult'.
    """

    # Filter for relevant Railcards
    relevant_railcards = ['Senior', 'Disabled', 'Adult']
    filtered_rail = rail[rail['Railcard'].isin(relevant_railcards)]

    # Group by train details
    train_groups = filtered_rail.groupby(['Ticket Class', 'Ticket Type',

    discrepancies = []

    for name, group in train_groups:
        # Check if there are multiple prices in the group
        if group['Price'].nunique() > 1:
            discrepancies.append(group)

    if discrepancies:
        return pd.concat(discrepancies)
    else:
        return "No price discrepancies found."
```

```
# Example Usage (assuming your DataFrame is named 'rail')
discrepancy_records = find_price_discrepancies(rail)

print(discrepancy_records)
```

```
No price discrepancies found.
```

Due to rounding, there are some entries where a first class ticket was recorded the same as a standard one for the same route (holding other factors constant)

In [98]:

```
# Create a new DataFrame to hold the results
results = []

# Group by the relevant columns
grouped = rail.groupby(['Departure Station', 'Arrival Destination', 'Date

# Iterate through each group
for name, group in grouped:
    # Check if there are both 'Standard' and 'First Class' in the group
    if 'Standard' in group['Ticket Class'].values and 'First Class' in gr
        # Get the prices for both classes
        standard_price = group.loc[group['Ticket Class'] == 'Standard', '
        first_class_price = group.loc[group['Ticket Class'] == 'First Cla

        # Check the price condition
        if standard_price >= first_class_price:
            results.append(group)

# Concatenate the results into a single DataFrame
filtered_df = pd.concat(results) if results else pd.DataFrame()

filtered_df
```

Out[98]:

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	Ticke Typ
3529	82070b5c-65ee-4c91-b4ac	2024-01-14	17:21:42	Online	Credit Card	Senior	First Class	Of Pea
3535	53e24dbc-525a-4519-8f83	2024-01-14	17:28:16	Online	Credit Card	Senior	Standard	Of Pea
17737	1d1cbdd7-1219-4dba-aae9	2024-03-08	20:18:47	Online	Credit Card	Senior	Standard	Advanc
17744	7dd7b086-4a99-4232-8b68	2024-03-08	20:23:37	Online	Credit Card	Senior	First Class	Advanc
18036	8b7b2a95-7c75-43dc-9029	2024-03-09	20:19:26	Online	Credit Card	Senior	Standard	Advanc
18039	49f083ed-2cf0-4204-aaae	2024-03-09	20:22:04	Online	Credit Card	Senior	Standard	Advanc
18040	6a1afbae-6451-40e1-b7fa	2024-03-09	20:24:10	Online	Credit Card	Senior	First Class	Advanc
19634	a05042e3-bf81-45ef-ba72	2024-03-15	17:17:01	Online	Credit Card	Senior	Standard	Anytim
19641	8b173675-4cc8-4ce8-ac5e	2024-03-15	17:25:26	Online	Credit Card	Senior	First Class	Anytim
19642	d3b704b8-3710-4da2-a704	2024-03-15	17:29:25	Online	Credit Card	Senior	Standard	Anytim
19757	d9037dc2-892e-4fdf-ba6f	2024-03-16	06:03:41	Online	Credit Card	Senior	First Class	Of Pea
19761	2fa511be-c790-4b0c-8fb2	2024-03-16	06:12:59	Online	Credit Card	Senior	Standard	Of Pea
29285	4a8d305e-5206-4a7e-aa9e	2024-04-21	17:16:58	Online	Credit Card	Senior	First Class	Of Pea
29290	89571e67-4144-48e5-b8c2	2024-04-21	17:22:02	Online	Credit Card	Senior	Standard	Of Pea

to delete them

In [ ]:

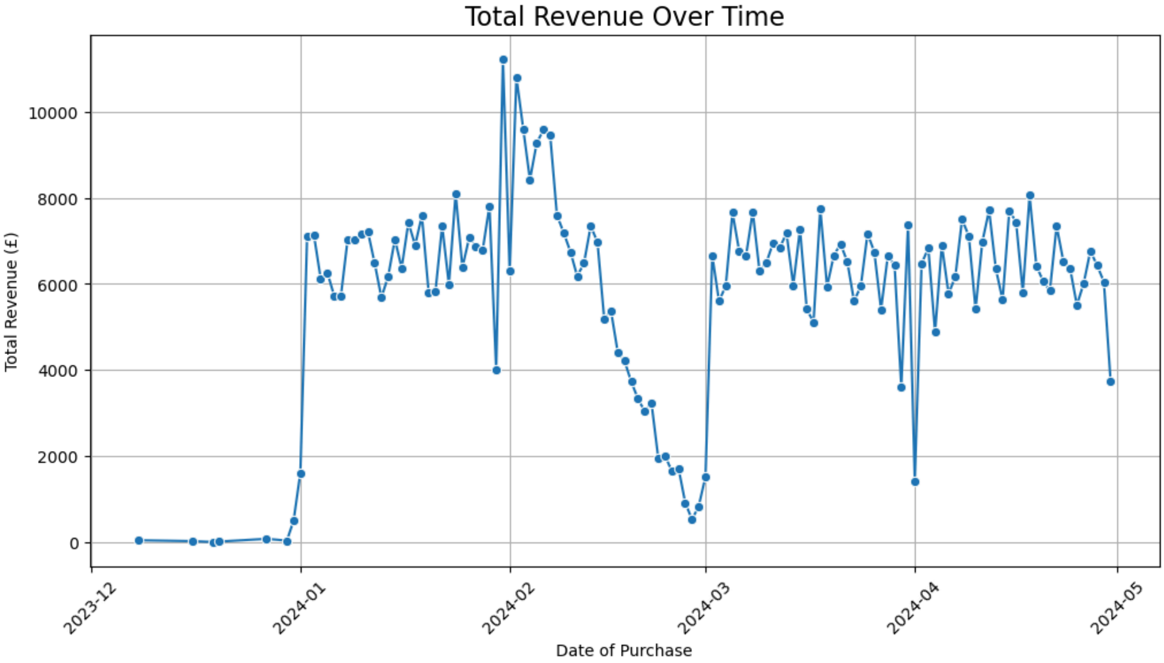
```
#inconsistent_indices = filtered_df.index.get_level_values(-1)
rail.drop(inconsistent_indices, inplace=True)
print("Remaining rows after deletion:", rail.shape[0])
```

In [ ]:

```

```

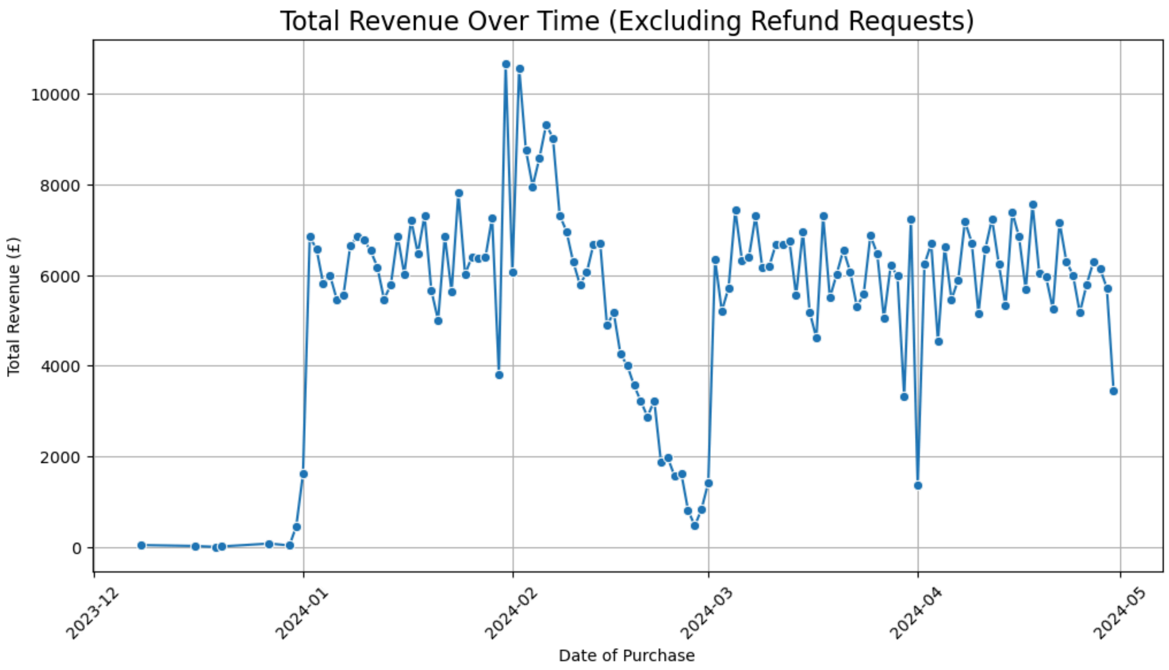
In [105…

```
rail['Date of Purchase'] = pd.to_datetime(rail['Date of Purchase'])
sales_over_time = rail.groupby('Date of Purchase')['Price'].sum().reset_i

plt.figure(figsize=(12,6))
sns.lineplot(data=sales_over_time, x='Date of Purchase', y='Price', marke
plt.title('Total Revenue Over Time', fontsize=16)
plt.xlabel('Date of Purchase')
plt.ylabel('Total Revenue (£)')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



In [109…

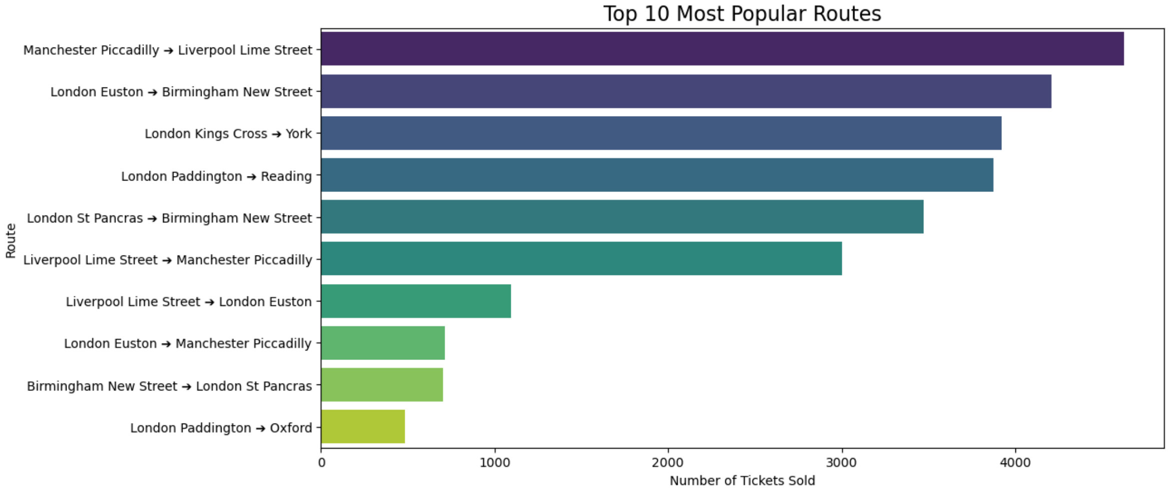
```
filtered_rail = rail[rail['Refund Request'] != 'Yes']

# Group by 'Date of Purchase' and calculate total revenue
sales_over_time = filtered_rail.groupby('Date of Purchase')['Price'].sum(

# Plot the results
plt.figure(figsize=(12,6))
sns.lineplot(data=sales_over_time, x='Date of Purchase', y='Price', marke
plt.title('Total Revenue Over Time (Excluding Refund Requests)', fontsize
plt.xlabel('Date of Purchase')
plt.ylabel('Total Revenue (£)')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



In [ ]:

In [111…

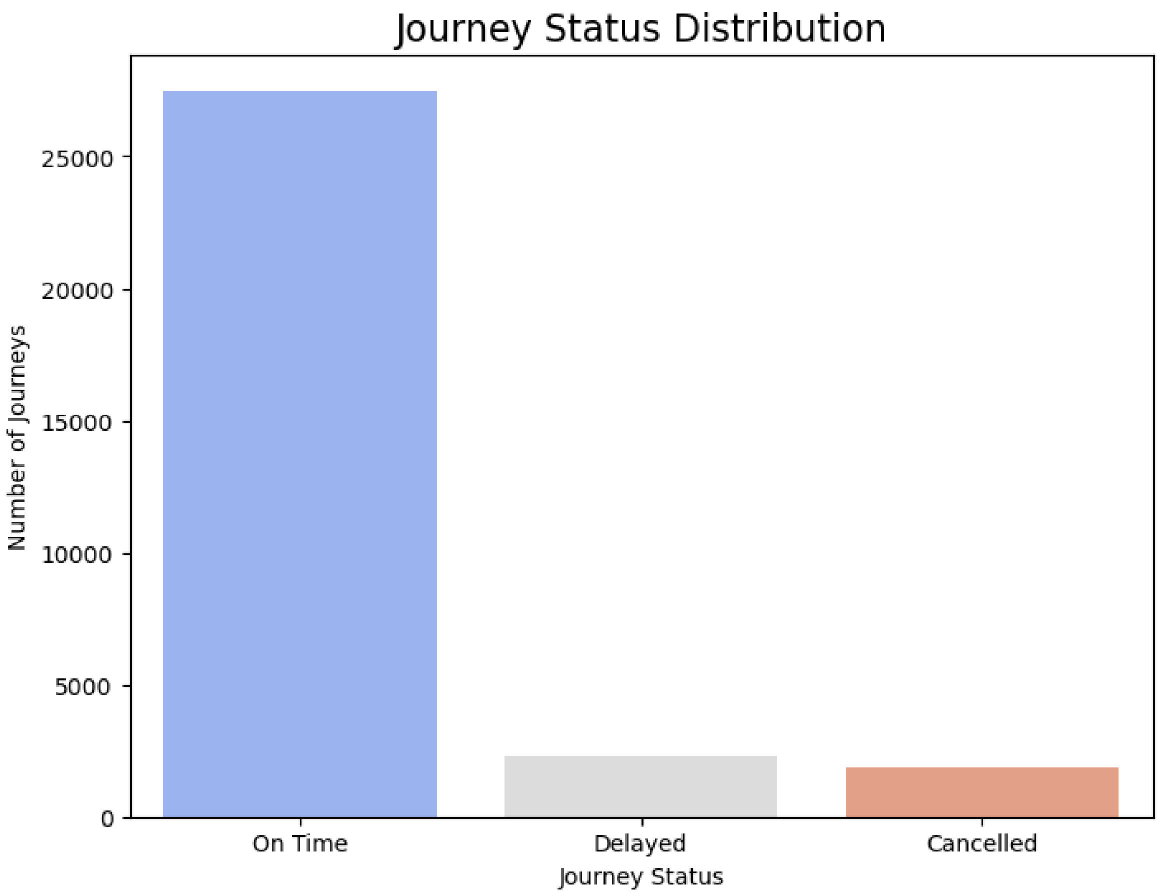
```
# Most frequent routes
rail['Route'] = rail['Departure Station'] + " ➔ " + rail['Arrival Destina
popular_routes = rail['Route'].value_counts().head(10)

plt.figure(figsize=(12,6))
sns.barplot(x=popular_routes.values, y=popular_routes.index, palette='vir
plt.title('Top 10 Most Popular Routes', fontsize=16)
plt.xlabel('Number of Tickets Sold')
plt.ylabel('Route')
plt.show()
```



In [ ]:

In [113…

```
# On-Time vs Delayed Journeys
status_counts = rail['Journey Status'].value_counts()

plt.figure(figsize=(8,6))
sns.barplot(x=status_counts.index, y=status_counts.values, palette='coolw
plt.title('Journey Status Distribution', fontsize=16)
plt.xlabel('Journey Status')
plt.ylabel('Number of Journeys')
plt.show()
```

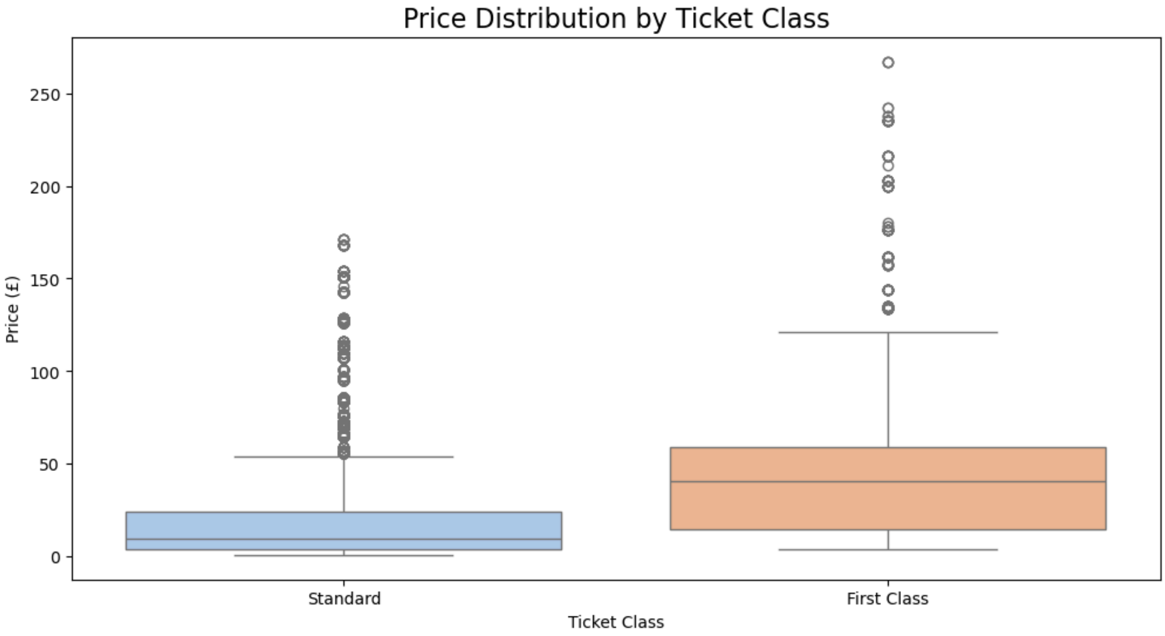


In [ ]:

```
```

In [132…

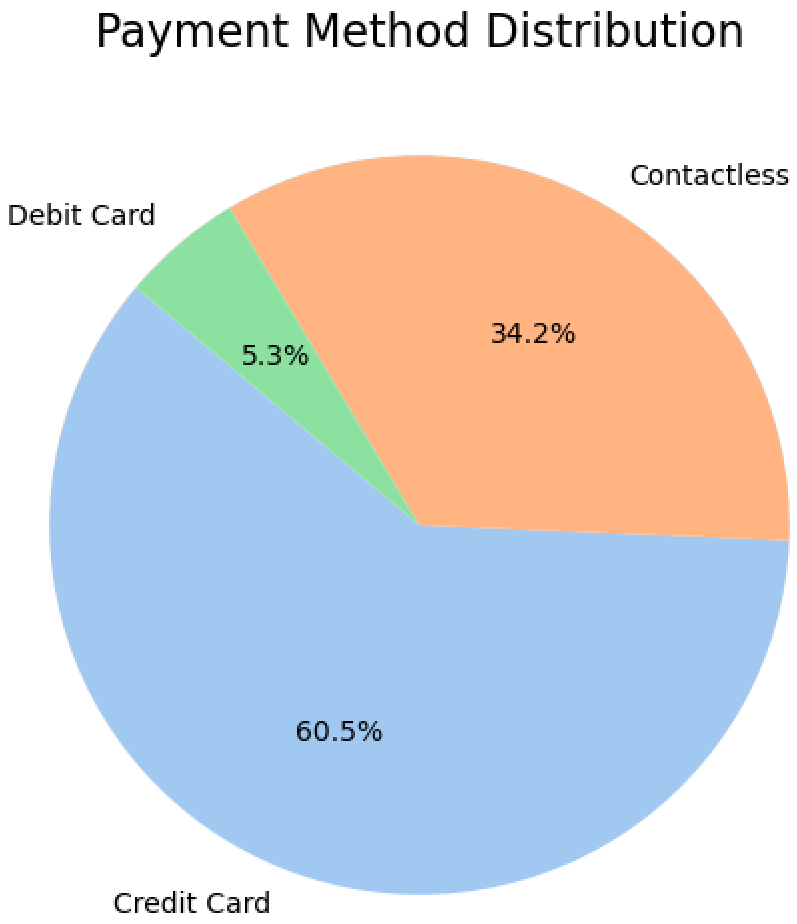
```
# Price distribution by Ticket Class
plt.figure(figsize=(12,6))
sns.boxplot(data=rail, x='Ticket Class', y='Price', palette='pastel')
plt.title('Price Distribution by Ticket Class', fontsize=16)
plt.xlabel('Ticket Class')
plt.ylabel('Price (£)')
plt.show()
```



In [ ]:

In [136…

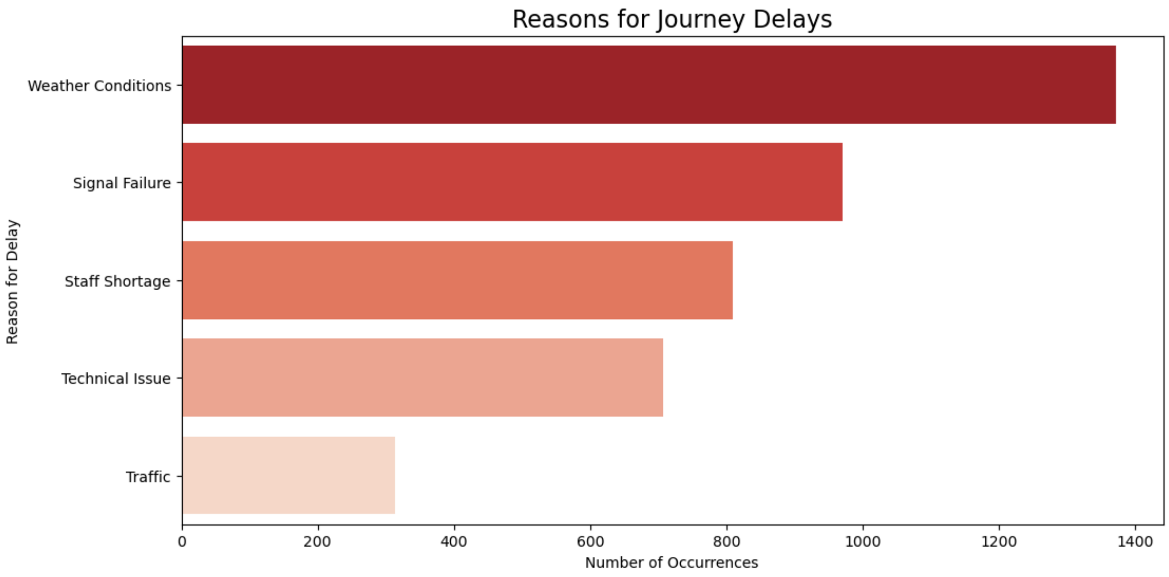
```
# Payment method breakdown
payment_counts = rail['Payment Method'].value_counts()

plt.figure(figsize=(8,6))
payment_counts.plot(kind='pie', autopct='%1.1f%%', startangle=140, colors
plt.title('Payment Method Distribution', fontsize=16)
plt.ylabel('')
plt.show()
```



In [ ]:

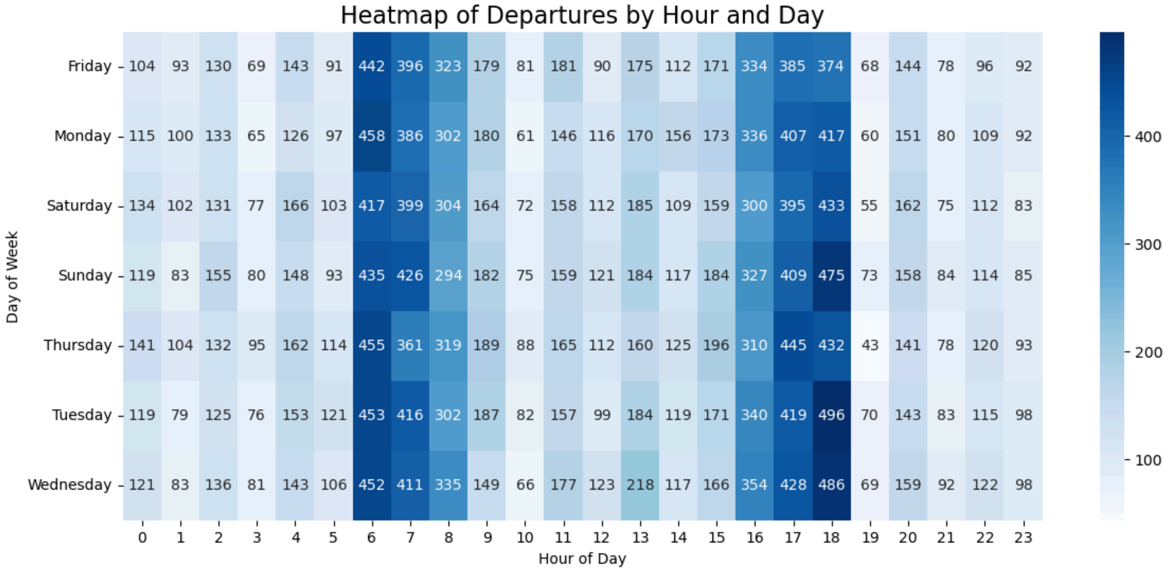
In [148…

```
filtered_delay_reasons = rail[rail['Reason for Delay'] != 'No Delay']['Re

# Count the occurrences of each delay reason
delay_reasons = filtered_delay_reasons.value_counts()

# Plot the results
plt.figure(figsize=(12,6))
sns.barplot(x=delay_reasons.values, y=delay_reasons.index, palette='Reds_
plt.title('Reasons for Journey Delays', fontsize=16)
plt.xlabel('Number of Occurrences')
plt.ylabel('Reason for Delay')
plt.show()
```



```
In [ ]:
```

```
In [154…
rail['Journey Day'] = pd.to_datetime(rail['Date of Journey']).dt.day_name

# Convert 'Departure Time' to a proper datetime string and extract the ho
rail['Departure Hour'] = pd.to_datetime(rail['Departure Time'].astype(str

# Create a crosstab of 'Journey Day' and 'Departure Hour'
heatmap_data = pd.crosstab(rail['Journey Day'], rail['Departure Hour'])

# Plot the heatmap
plt.figure(figsize=(14,6))
sns.heatmap(heatmap_data, cmap='Blues', annot=True, fmt='d')
plt.title('Heatmap of Departures by Hour and Day', fontsize=16)
plt.xlabel('Hour of Day')
plt.ylabel('Day of Week')
plt.show()
```



```
In [ ]:
```

```
In [156…
rail.head()
```

Out[156…

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	Ticket Type
0	da8a6ba8-b3dc-4677-b176	2023-12-08	12:41:11	Online	Contactless	Adult	Standard	Advance
1	b0cdd1b0-f214-4197-be53	2023-12-16	11:23:01	Station	Credit Card	Adult	Standard	Advance
2	f3ba7a96-f713-40d9-9629	2023-12-19	19:51:27	Online	Credit Card	No Railcard	Standard	Advance
3	b2471f11-4fe7-4c87-8ab4	2023-12-20	23:00:36	Station	Credit Card	No Railcard	Standard	Advance
4	2be00b45-0762-485e-a7a3	2023-12-27	18:22:56	Online	Contactless	No Railcard	Standard	Advance

5 rows × 21 columns

In [ ]:

In [158…

```
# Assuming 'rail' is your DataFrame

# Convert relevant columns to datetime (if not already done)
rail['Date of Purchase'] = pd.to_datetime(rail['Date of Purchase'])
rail['Date of Journey'] = pd.to_datetime(rail['Date of Journey'])
rail['Arrival Time'] = pd.to_datetime(rail['Arrival Time'].astype(str))
rail['Actual Arrival Time'] = pd.to_datetime(rail['Actual Arrival Time'].

# Add 'Booking Lead Days' column
rail['Booking Lead Days'] = (rail['Date of Journey'] - rail['Date of Purc

# Add 'Delay Time' column (in minutes)
rail['Delay Time'] = (rail['Actual Arrival Time'] - rail['Arrival Time'])

# Convert the timedelta to hh:mm:ss format
rail['Delay Time'] = rail['Delay Time'].apply(lambda x: str(x).split()[-1

# Display the updated DataFrame
print(rail[['Date of Purchase', 'Date of Journey', 'Booking Lead Days', '
  Date of Purchase Date of Journey  Booking Lead Days        Arrival Time  
\
0       2023-12-08      2024-01-01                 24 2025-02-22 13:30:00   
1       2023-12-16      2024-01-01                 16 2025-02-22 11:35:00   
2       2023-12-19      2024-01-02                 14 2025-02-22 18:45:00   
3       2023-12-20      2024-01-01                 12 2025-02-22 22:30:00   
4       2023-12-27      2024-01-01                  5 2025-02-22 19:00:00   

  Actual Arrival Time Delay Time  
0 2025-02-22 13:30:00   00:00:00  
1 2025-02-22 11:40:00   00:05:00  
2 2025-02-22 18:45:00   00:00:00  
3 2025-02-22 22:30:00   00:00:00  
4 2025-02-22 19:00:00   00:00:00  
```

In [162…

```
rail['Departure Time'] = pd.to_datetime(rail['Departure Time'], format='%
rail['Arrival Time'] = pd.to_datetime(rail['Arrival Time'], format='%H:%M
rail['Actual Arrival Time'] = pd.to_datetime(rail['Actual Arrival Time'],
rail['Time of Purchase'] = pd.to_datetime(rail['Time of Purchase'], forma
```

In [164…

```
rail.head()
```

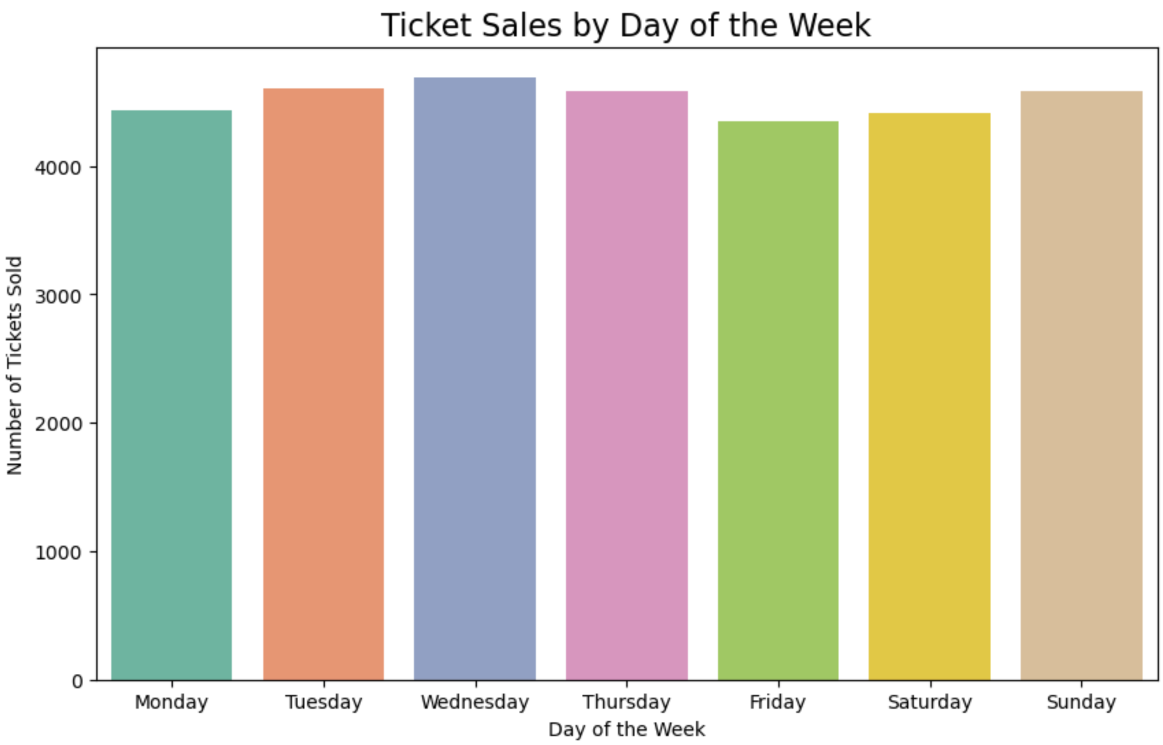
Out[164…

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	Ticket Type
0	da8a6ba8-b3dc-4677-b176	2023-12-08	12:41:11	Online	Contactless	Adult	Standard	Advance
1	b0cdd1b0-f214-4197-be53	2023-12-16	11:23:01	Station	Credit Card	Adult	Standard	Advance
2	f3ba7a96-f713-40d9-9629	2023-12-19	19:51:27	Online	Credit Card	No Railcard	Standard	Advance
3	b2471f11-4fe7-4c87-8ab4	2023-12-20	23:00:36	Station	Credit Card	No Railcard	Standard	Advance
4	2be00b45-0762-485e-a7a3	2023-12-27	18:22:56	Online	Contactless	No Railcard	Standard	Advance

5 rows × 23 columns

In [166…

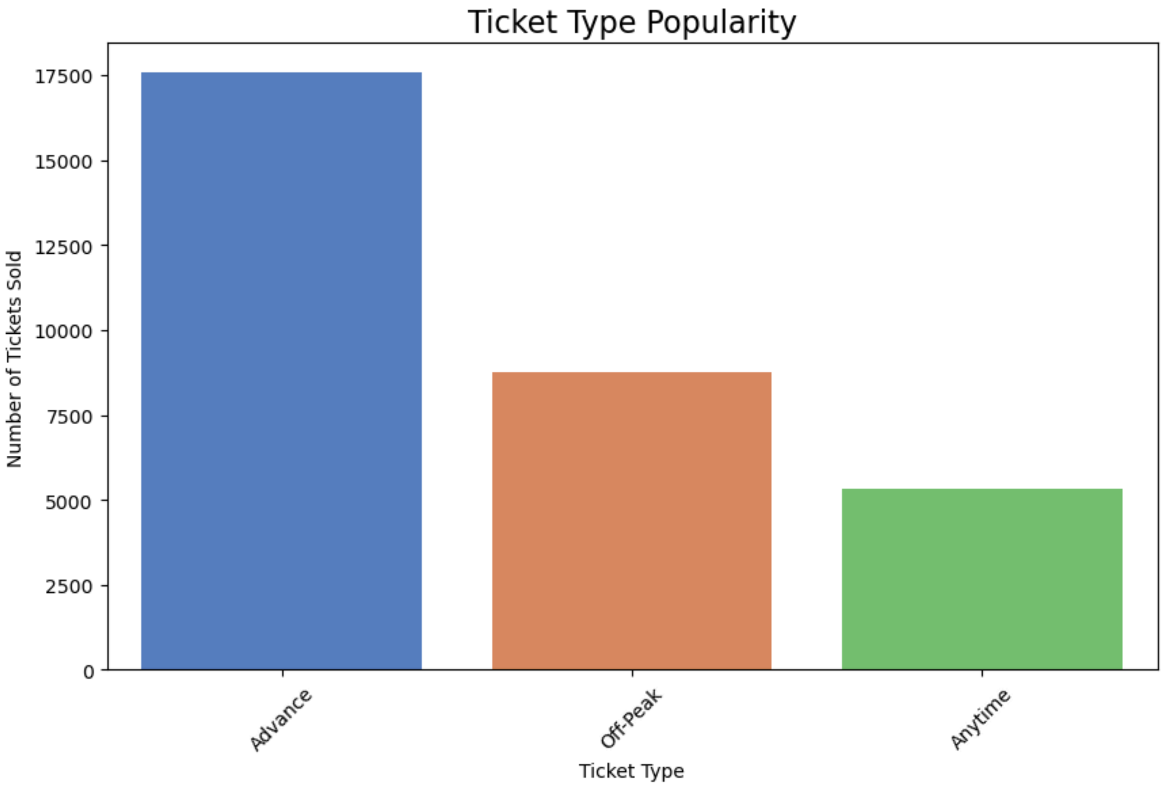
```
# Ticket sales per day
plt.figure(figsize=(10,6))
sns.countplot(data=rail, x='Journey Day', order=['Monday', 'Tuesday', 'We
plt.title('Ticket Sales by Day of the Week', fontsize=16)
plt.xlabel('Day of the Week')
plt.ylabel('Number of Tickets Sold')
plt.show()
```



In [ ]:

In [168…

```
# Popular ticket types
ticket_type_counts = rail['Ticket Type'].value_counts()

plt.figure(figsize=(10,6))
sns.barplot(x=ticket_type_counts.index, y=ticket_type_counts.values, pale
plt.title('Ticket Type Popularity', fontsize=16)
plt.xlabel('Ticket Type')
plt.ylabel('Number of Tickets Sold')
plt.xticks(rotation=45)
plt.show()
```

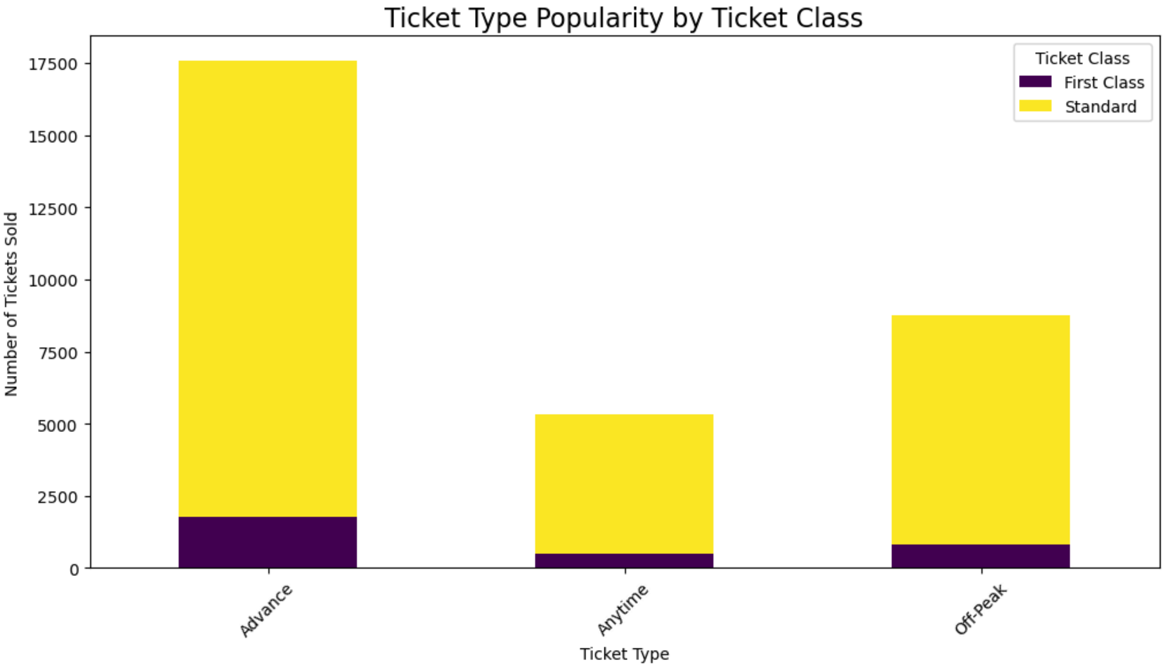


In [ ]:

In [170…

```
ticket_type_class_counts = rail.groupby(['Ticket Type', 'Ticket Class']).

# Plot the results
plt.figure(figsize=(12,6))
ticket_type_class_counts.plot(kind='bar', stacked=True, colormap='viridis
plt.title('Ticket Type Popularity by Ticket Class', fontsize=16)
plt.xlabel('Ticket Type')
plt.ylabel('Number of Tickets Sold')
plt.xticks(rotation=45)
plt.legend(title='Ticket Class')
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```

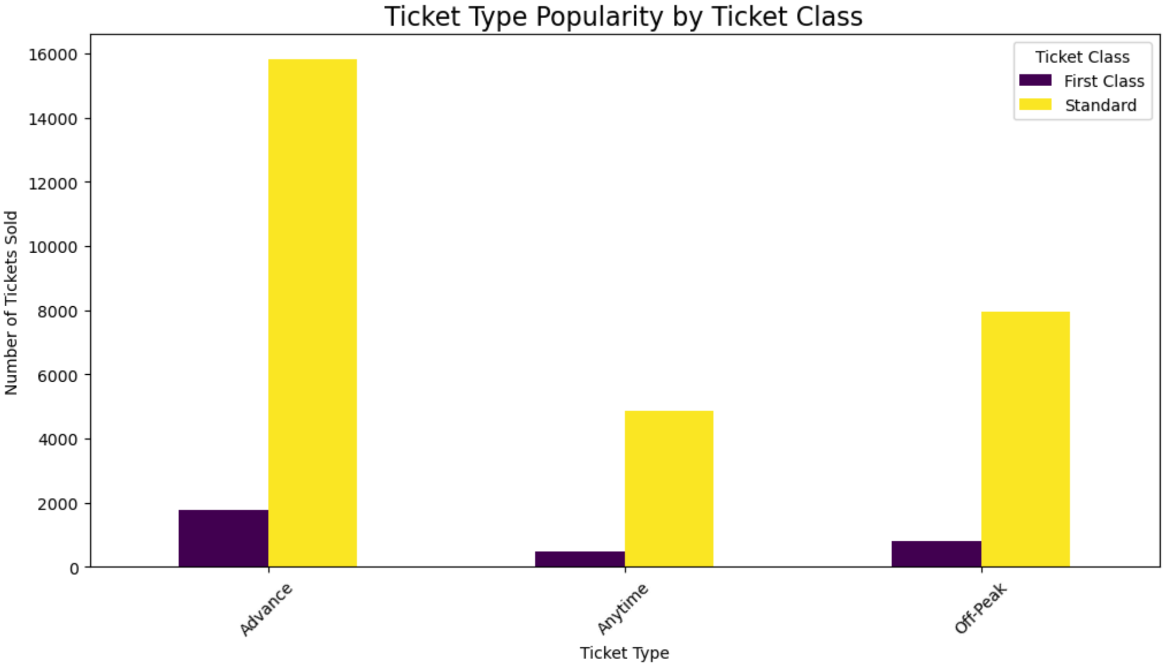


In [ ]:

In [172…

```
plt.figure(figsize=(12,6))
ticket_type_class_counts.plot(kind='bar', colormap='viridis', figsize=(12
plt.title('Ticket Type Popularity by Ticket Class', fontsize=16)
plt.xlabel('Ticket Type')
plt.ylabel('Number of Tickets Sold')
plt.xticks(rotation=45)
plt.legend(title='Ticket Class')
plt.show()
```

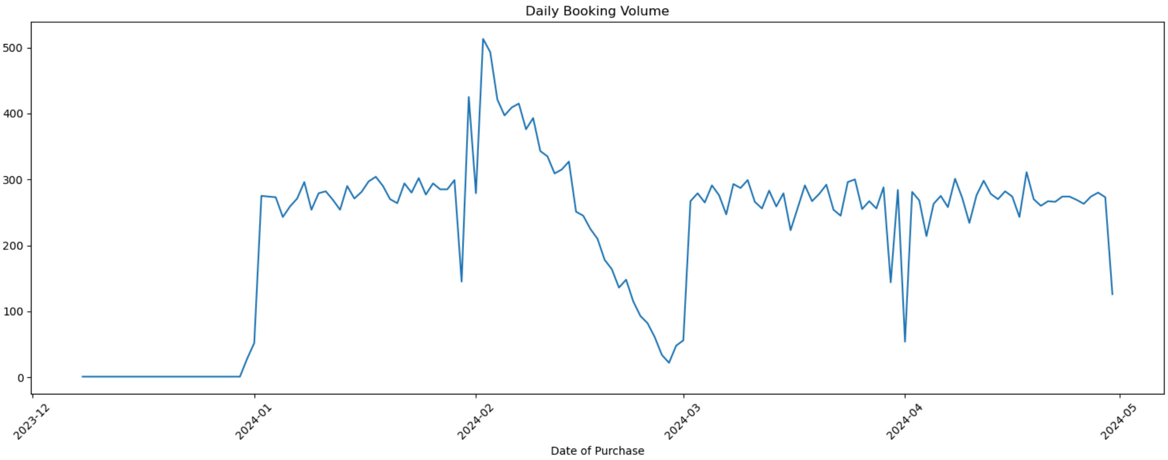
```
<Figure size 1200x600 with 0 Axes>
```

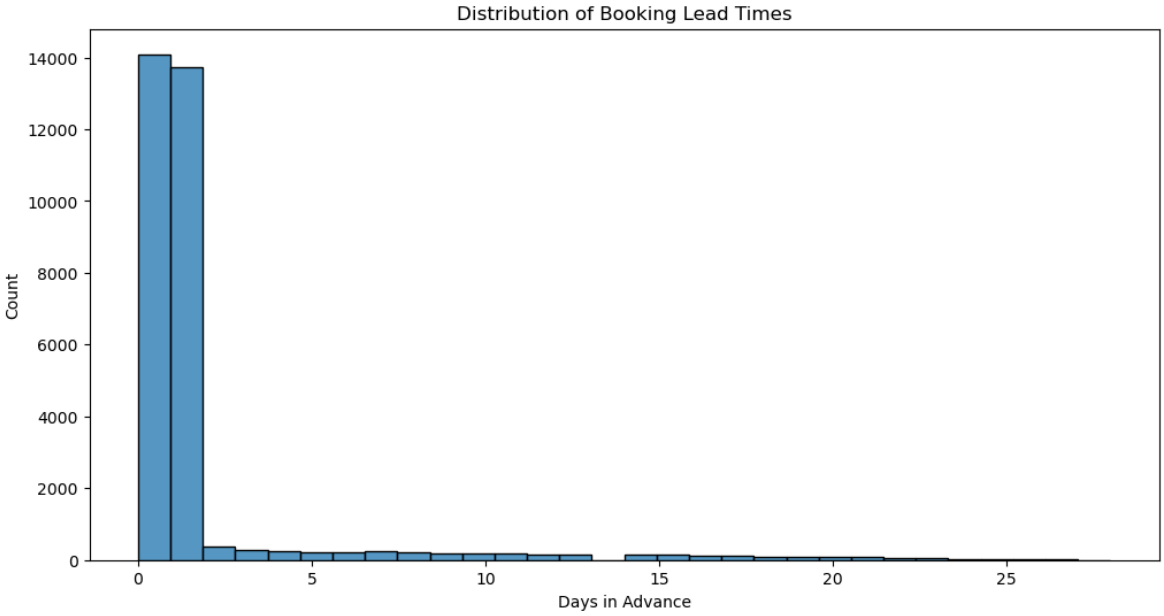


In [ ]:

In [186…

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
from datetime import datetime

# Convert date columns to datetime
date_columns = ['Date of Purchase', 'Date of Journey']
for col in date_columns:
    rail[col] = pd.to_datetime(rail[col])

# 1. Daily Booking Trends
plt.figure(figsize=(15, 6))
daily_bookings = rail['Date of Purchase'].value_counts().sort_index()
sns.lineplot(x=daily_bookings.index, y=daily_bookings.values)
plt.title('Daily Booking Volume')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 2. Booking Lead Time Distribution
plt.figure(figsize=(12, 6))
sns.histplot(data=rail, x='Booking Lead Days', bins=30)
plt.title('Distribution of Booking Lead Times')
plt.xlabel('Days in Advance')
plt.show()
```





In [184…

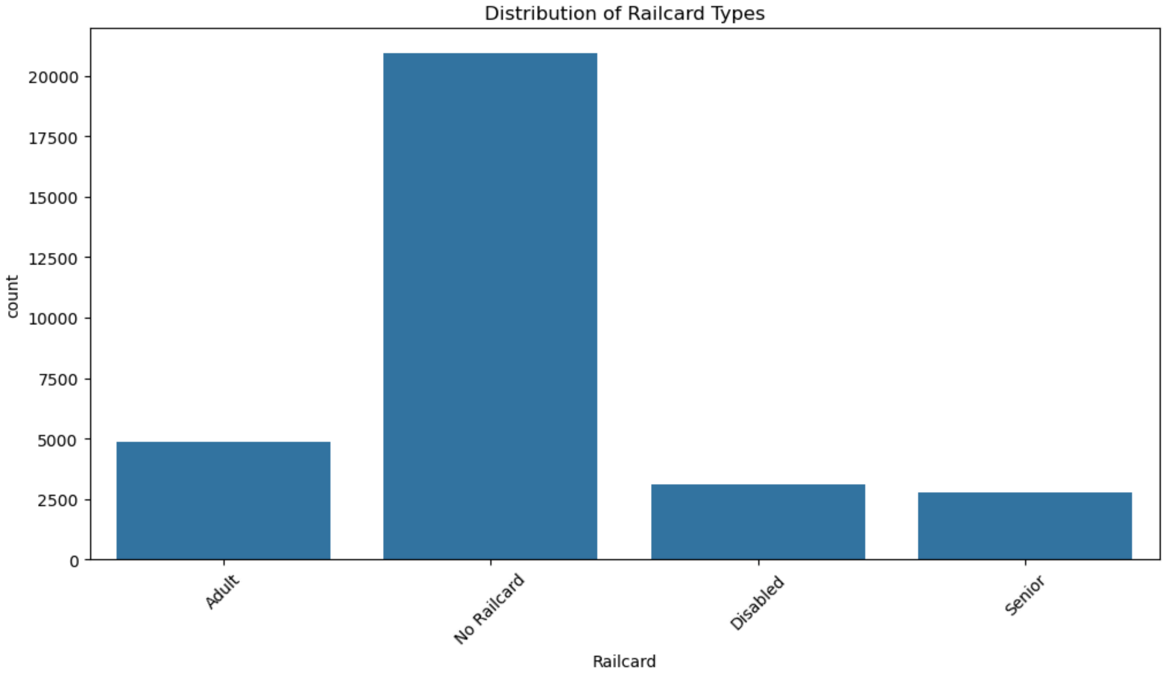
```
# For the hourly booking pattern analysis, we can directly use the hour f
rail['Hour'] = rail['Time of Purchase'].apply(lambda x: x.hour)
hourly_bookings = rail['Hour'].value_counts().sort_index()

fig = go.Figure(data=[
    go.Bar(x=hourly_bookings.index, y=hourly_bookings.values)
])
fig.update_layout(
    title='Hourly Booking Distribution',
    xaxis_title='Hour of Day',
    yaxis_title='Number of Bookings'
)
fig.show()
```

In [ ]:

```
```

In [196…

```
# 2. Railcard Usage
plt.figure(figsize=(12, 6))
sns.countplot(data=rail, x='Railcard')
plt.title('Distribution of Railcard Types')
plt.xticks(rotation=45)
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```