

UK Train Rides Analysis

This notebook demonstrates the process of generating and analyzing mock train ticket data for the National Rail in the UK, covering journeys from January to April 2024. The data includes various details about each journey, such as the type of ticket, departure & arrival stations, ticket prices, and journey status.

Data Generation

The dataset consists of the following fields:

- **Transaction ID:** Unique identifier for each ticket purchase.
- **Date of Purchase:** The date the ticket was bought.
- **Time of Purchase:** The exact time the ticket was purchased.
- **Purchase Type:** Whether the ticket was bought online or at the station.
- **Payment Method:** Method used for payment (e.g., Credit Card, Contactless).
- **Railcard:** Type of railcard used (e.g., Adult, Senior, Disabled).
- **Ticket Class:** Class of the ticket (e.g., Standard, First Class).
- **Ticket Type:** Type of the ticket (e.g., Advance, Anytime).
- **Price:** The price of the ticket.
- **Departure Station:** The station where the journey starts.
- **Arrival Station:** The station where the journey ends.
- **Date of Journey:** The date of the journey.
- **Departure Time:** The scheduled departure time.
- **Arrival Time:** The scheduled arrival time.
- **Actual Arrival Time:** The actual arrival time (used to calculate delays).
- **Journey Status:** Status of the journey (e.g., On Time, Delayed).
- **Reason for Delay:** If the journey was delayed, this field describes the reason (e.g., Signal Failure, Weather).
- **Refund Request:** Whether a refund was requested (Yes/No).

Analysis

1. **Most Popular Routes:** We group the data by departure and arrival stations, counting the occurrences of each route to identify the most frequently traveled routes.
2. **Peak Travel Times:** By analyzing the `Departure Time`, we identify peak hours for train travel.
3. **Revenue Variation by Ticket Types and Classes:** We calculate the total revenue by grouping the data by `Ticket Type` and `Ticket Class` and summing the `Price` for each category.
4. **On-Time Performance (OTP):** The On-Time Performance is calculated as the percentage of journeys marked as "On Time". We also calculate the average delay time for delayed journeys and group the delays by reason.

And Many more

The dataset allows for insight into the operational performance of train journeys, including factors like ticket sales, popular routes, peak travel times, and delay reasons.

Import Libraries and Dataset

```
In [1]: 1 import pandas as pd
        2 import matplotlib.pyplot as plt
        3 import seaborn as sns
        4 import time
        5 import warnings
        6 warnings.filterwarnings('ignore')

In [2]: 1 data = pd.read_csv(r"D:\Maven Analytics\UK_Train_Rides\railway.csv")
```

Exploratory Data Analysis (EDA)

```
In [3]: 1 print("Dataset Preview:")
        2 data.head()
```

Dataset Preview:

Out[3]:

	Transaction ID	Date of Purchase	Time of Purchase	Purchase Type	Payment Method	Railcard	Ticket Class	Ticket Type	Price	Departure Station	Arrival Destination	Date of Journey	Departure Time
0	da8a6ba8-b3dc-4677-b176	2023-12-08	12:41:11	Online	Contactless	Adult	Standard	Advance	43	London Paddington	Liverpool Lime Street	2024-01-01	11:00:00
1	b0cdd1b0-f214-4197-be53	2023-12-16	11:23:01	Station	Credit Card	Adult	Standard	Advance	23	London Kings Cross	York	2024-01-01	09:45:00
2	f3ba7a96-f713-40d9-9629	2023-12-19	19:51:27	Online	Credit Card	None	Standard	Advance	3	Liverpool Lime Street	Manchester Piccadilly	2024-01-02	18:15:00
3	b2471f11-4fe7-4c87-8ab4	2023-12-20	23:00:36	Station	Credit Card	None	Standard	Advance	13	London Paddington	Reading	2024-01-01	21:30:00
4	2be00b45-0762-485e-a7a3	2023-12-27	18:22:56	Online	Contactless	None	Standard	Advance	76	Liverpool Lime Street	London Euston	2024-01-01	16:45:00

```
In [4]: 1 data.shape
```

Out[4]: (31653, 18)

```
In [5]: 1 print("\nMissing Values:")
        2 print(data.isnull().sum())
```

Missing Values:

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

```
In [6]: 1 print("\nSummary Statistics:")
        2 print(data.describe())
```

Summary Statistics:

	Price
count	31653.000000
mean	23.439200
std	29.997628
min	1.000000
25%	5.000000
50%	11.000000
75%	35.000000
max	267.000000

```
In [7]: 1 data.columns
```

```
Out[7]: Index(['Transaction ID', 'Date of Purchase', 'Time of Purchase',  
             'Purchase Type', 'Payment Method', 'Railcard', 'Ticket Class',  
             'Ticket Type', 'Price', 'Departure Station', 'Arrival Destination',  
             'Date of Journey', 'Departure Time', 'Arrival Time',  
             'Actual Arrival Time', 'Journey Status', 'Reason for Delay',  
             'Refund Request'],  
            dtype='object')
```

Data Cleaning

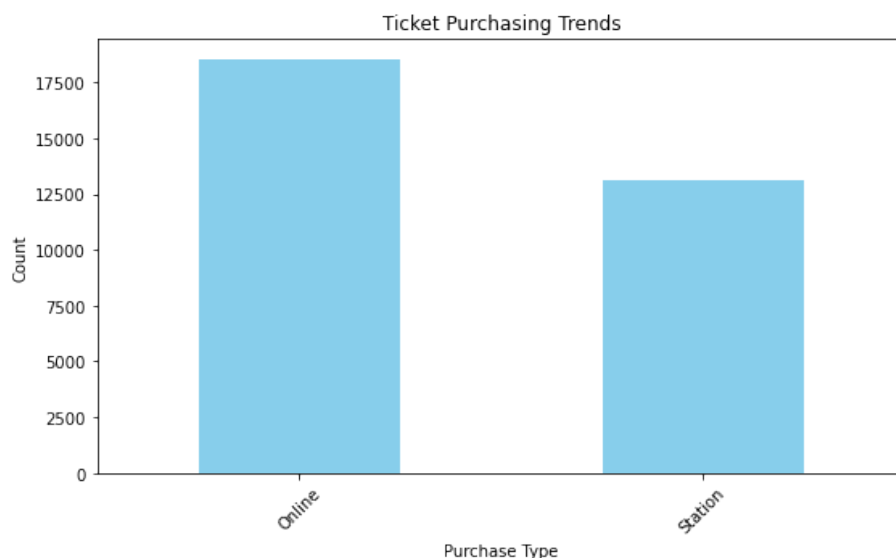
```
In [8]: 1 data['Date of Purchase'] = pd.to_datetime(data['Date of Purchase'], errors='coerce')  
2 data['Date of Journey'] = pd.to_datetime(data['Date of Journey'], errors='coerce')  
3 data['Time of Purchase'] = pd.to_datetime(data['Time of Purchase'], format='%H:%M:%S', errors='coerce')  
4 data['Departure Time'] = pd.to_datetime(data['Departure Time'], format='%H:%M:%S', errors='coerce')  
5 data['Arrival Time'] = pd.to_datetime(data['Arrival Time'], format='%H:%M:%S', errors='coerce')  
6 data['Actual Arrival Time'] = pd.to_datetime(data['Actual Arrival Time'], format='%H:%M:%S', errors='coerce')
```

```
In [9]: 1 data['Reason for Delay'] = data['Reason for Delay'].fillna('No delay')  
2 data['Reason for Delay'] = data['Reason for Delay'].replace('Signal failure', 'Signal Failure')
```

```
In [10]: 1 categorical_columns = ['Purchase Type', 'Payment Method', 'Railcard', 'Ticket Class', 'Ticket Type', 'Journey Status']  
2 for col in categorical_columns:  
3     data[col] = data[col].astype('category')  
4
```

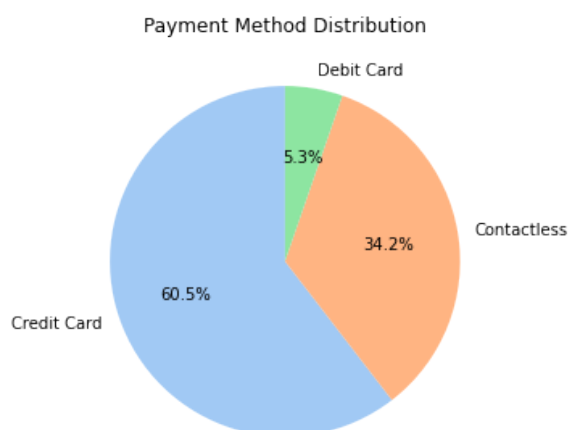
Data Visualization and Analysis

```
In [11]: 1 # Purchase Type  
2 purchase_trends = data['Purchase Type'].value_counts()  
3 plt.figure(figsize=(8, 5))  
4 purchase_trends.plot(kind='bar', color='skyblue')  
5 plt.title('Ticket Purchasing Trends')  
6 plt.xlabel('Purchase Type')  
7 plt.ylabel('Count')  
8 plt.xticks(rotation=45)  
9 plt.tight_layout()  
10 plt.show()
```



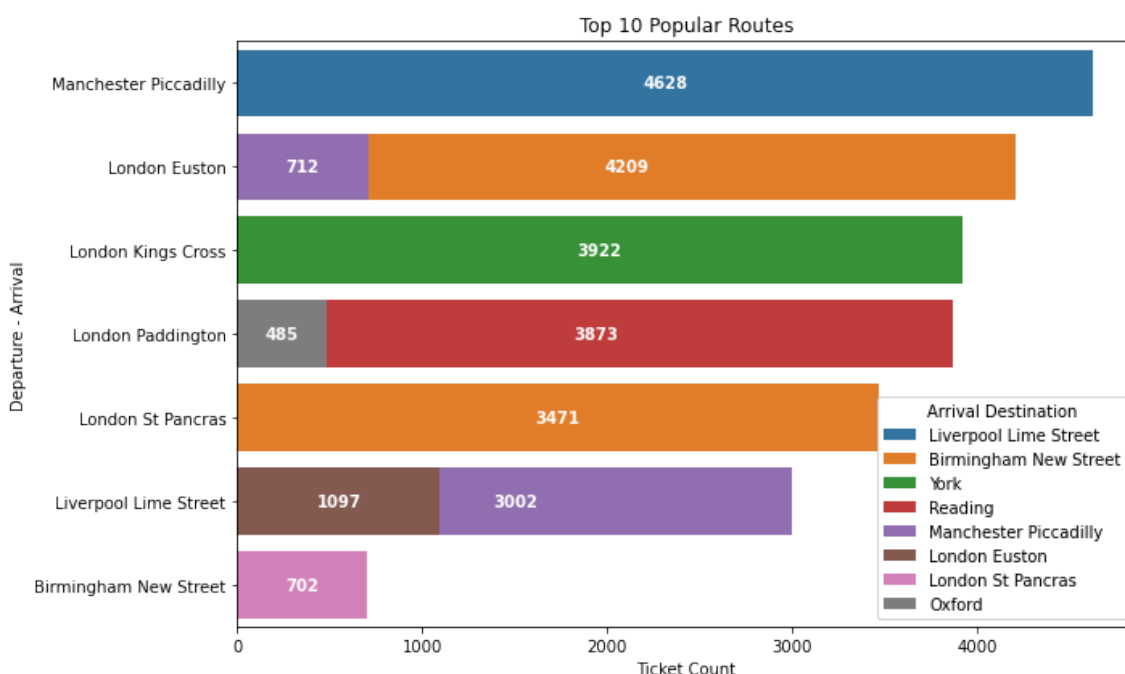
In [12]:

```
1 # Payment Methods
2 payment_methods = data['Payment Method'].value_counts()
3 plt.figure(figsize=(8, 5))
4 payment_methods.plot(kind='pie', autopct='%1.1f%%', startangle=90, colors=sns.color_palette('pastel'))
5 plt.title('Payment Method Distribution')
6 plt.ylabel('')
7 plt.show()
```



In [13]:

```
1 # Popular Routes
2 # Calculate popular routes
3 popular_routes = data.groupby(['Departure Station', 'Arrival Destination']).size().reset_index(name='Count')
4 popular_routes = popular_routes.sort_values('Count', ascending=False).head(10)
5
6 # Create the bar plot
7 plt.figure(figsize=(10, 6))
8 bar_plot = sns.barplot(data=popular_routes, x='Count', y='Departure Station', hue='Arrival Destination', dod=
9
10 # Add annotations to the bars (values in the middle)
11 for container in bar_plot.containers:
12     for bar in container:
13         bar_width = bar.get_width()
14         bar_height = bar.get_height()
15         if not pd.isna(bar_width): # Skip NaN values
16             plt.text(bar_width / 2, bar.get_y() + bar_height / 2,
17                      f'{int(bar_width)}', # Display the count
18                      va='center', ha='center', color='white', weight='bold')
19
20 # Customize plot
21 plt.title('Top 10 Popular Routes')
22 plt.xlabel('Ticket Count')
23 plt.ylabel('Departure - Arrival')
24 plt.tight_layout()
25 plt.show()
26
27
```

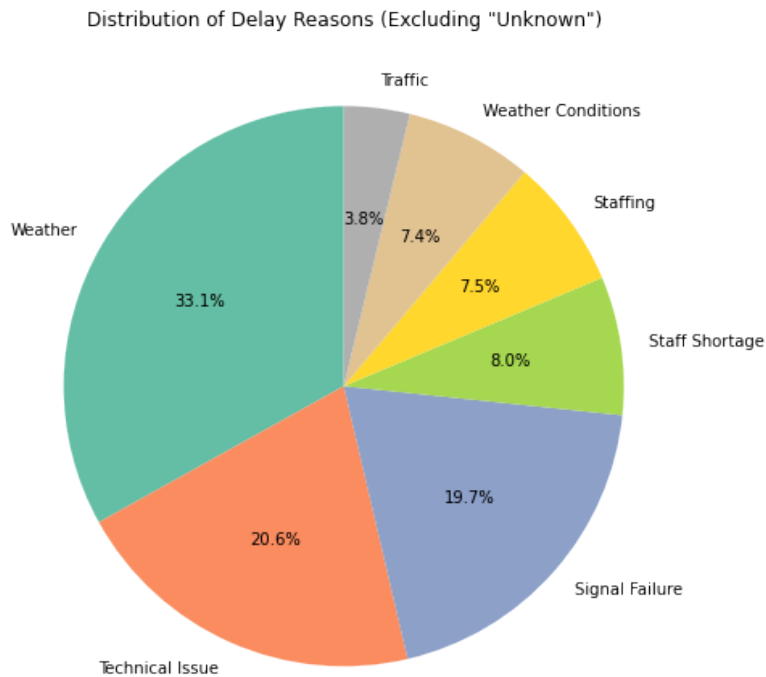


In [14]:

```
1 # Filter out rows where 'Reason for Delay' is 'Unknown' or blank
2 delayed_data_filtered = data[(data['Journey Status'] == 'Delayed') &
3                               ~data['Reason for Delay'].isin(['Unknown', ''])]
```

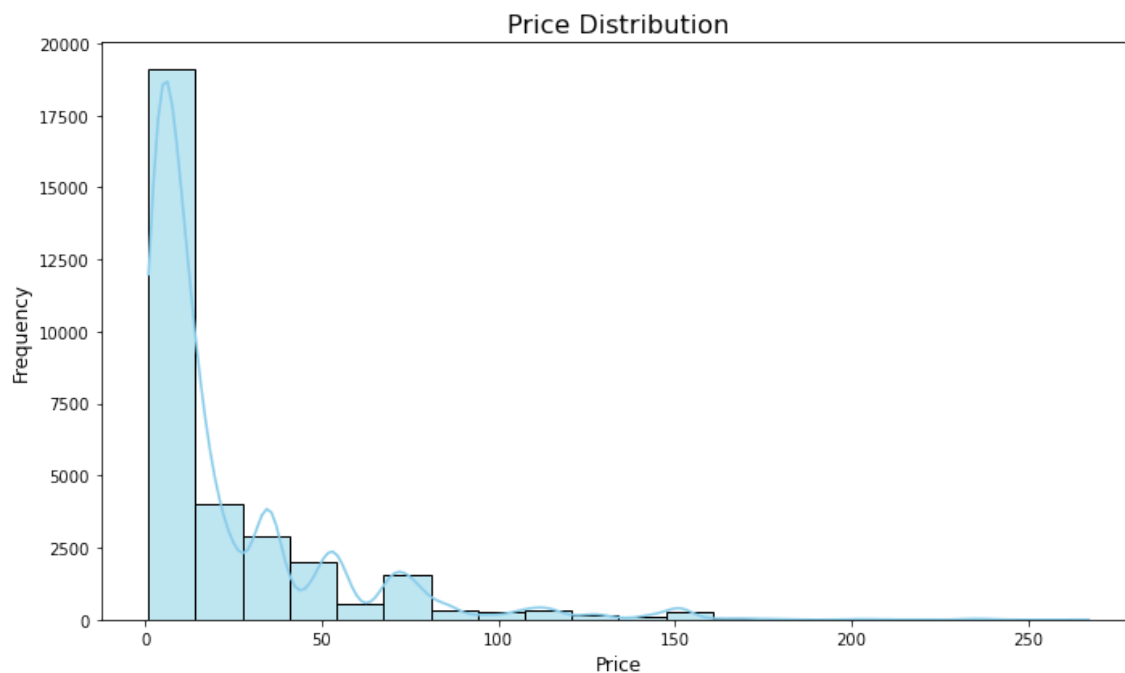
In [15]:

```
1 # Plot reasons for delays as a pie chart
2 plt.figure(figsize=(8, 8))
3 reason_counts = delayed_data_filtered['Reason for Delay'].value_counts()
4 reason_counts.plot.pie(autopct='%1.1f%%', startangle=90, legend=False, cmap='Set2')
5 plt.title('Distribution of Delay Reasons (Excluding "Unknown")')
6 plt.ylabel('')
7 plt.show()
```

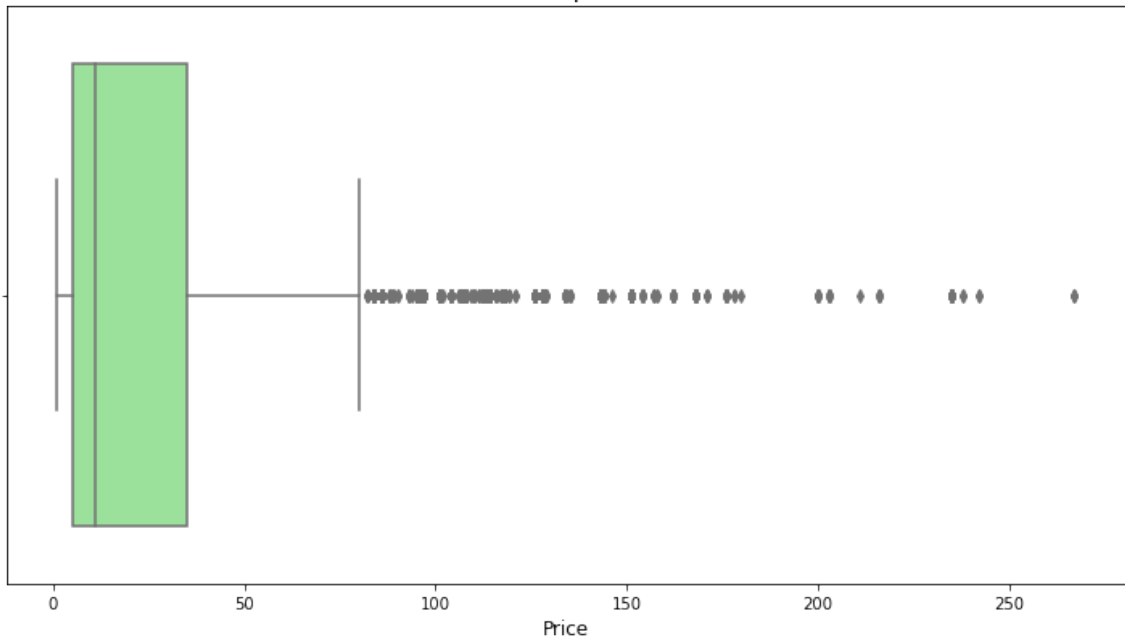


In [16]:

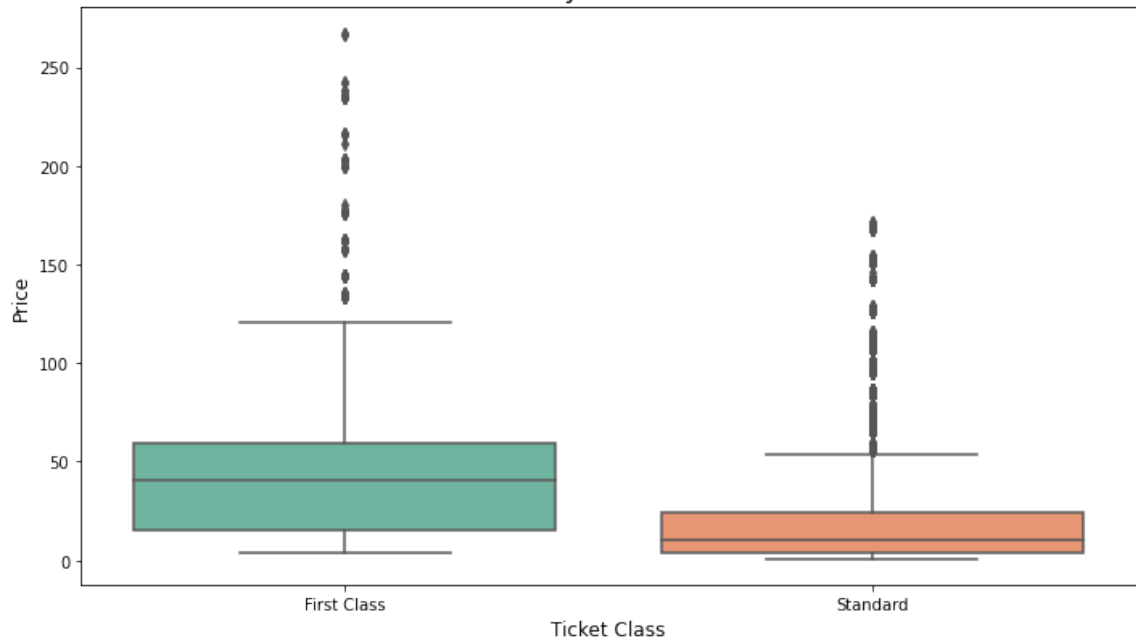
```
1 # 1. Histogram to show the distribution of prices
2 plt.figure(figsize=(10, 6))
3 sns.histplot(data['Price'], kde=True, color='skyblue', bins=20)
4 plt.title('Price Distribution', fontsize=16)
5 plt.xlabel('Price', fontsize=12)
6 plt.ylabel('Frequency', fontsize=12)
7 plt.tight_layout()
8 plt.show()
9
10
11 # 2. Box Plot to show the spread of prices
12 plt.figure(figsize=(10, 6))
13 sns.boxplot(data=data, x='Price', color='lightgreen')
14 plt.title('Price Spread', fontsize=16)
15 plt.xlabel('Price', fontsize=12)
16 plt.tight_layout()
17 plt.show()
18
19 # 3. Price by Ticket Class
20 plt.figure(figsize=(10, 6))
21 sns.boxplot(data=data, x='Ticket Class', y='Price', palette="Set2")
22 plt.title('Price by Ticket Class', fontsize=16)
23 plt.xlabel('Ticket Class', fontsize=12)
24 plt.ylabel('Price', fontsize=12)
25 plt.tight_layout()
26 plt.show()
27
28 # 4. Price by Railcard
29 plt.figure(figsize=(10, 6))
30 sns.boxplot(data=data, x='Railcard', y='Price', palette="Set1")
31 plt.title('Price by Railcard', fontsize=16)
32 plt.xlabel('Railcard', fontsize=12)
33 plt.ylabel('Price', fontsize=12)
34 plt.tight_layout()
35 plt.show()
36
```



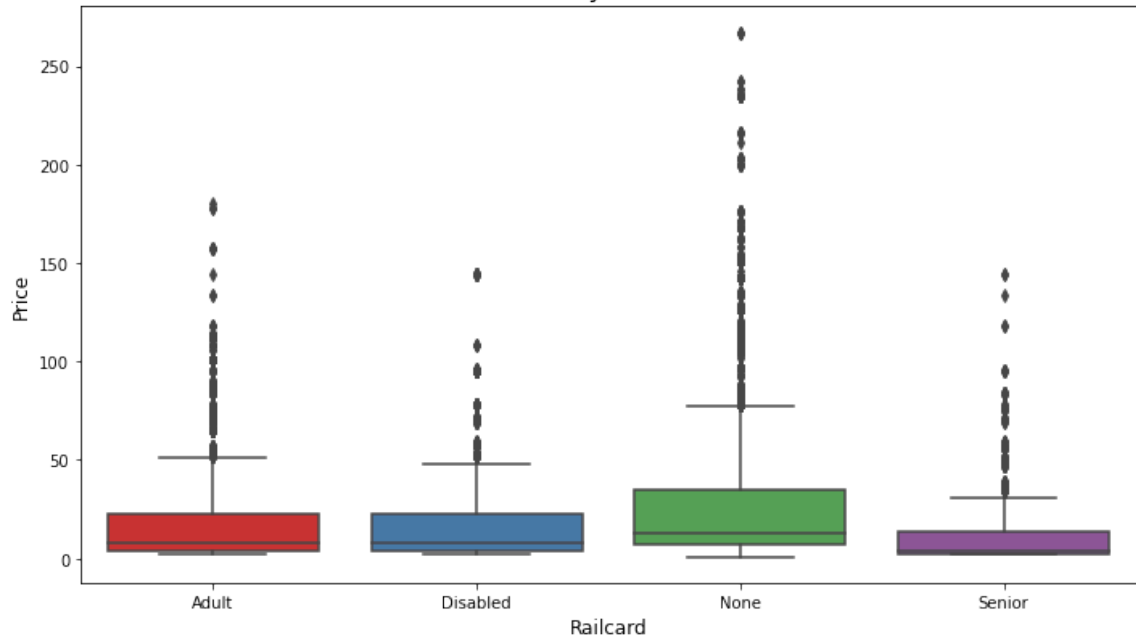
Price Spread



Price by Ticket Class



Price by Railcard



In [17]:

```
1 # Route with max delays
2
3 # Count the number of delays for each route (Departure Station to Arrival Destination)
4 route_delays = delayed_data_filtered.groupby(['Departure Station', 'Arrival Destination']).size().reset_index()
5
6 # Find the route with the most delays
7 max_delayed_route = route_delays.loc[route_delays['Delay Count'].idxmax()]
8
9 # Filter for rows that match the route with the most delays
10 max_delayed_route_data = delayed_data_filtered[
11     (delayed_data_filtered['Departure Station'] == max_delayed_route['Departure Station']) &
12     (delayed_data_filtered['Arrival Destination'] == max_delayed_route['Arrival Destination'])
13 ]
14
15 # Count the occurrences of each reason for delay for this route
16 delay_reason_counts = max_delayed_route_data['Reason for Delay'].value_counts()
17
18 # Print the route with the most delays and the delay reason counts
19 print("Route with the most delays:")
20 print(f"Route: {max_delayed_route['Departure Station']} -> {max_delayed_route['Arrival Destination']}")
21 print(f"Number of Delays: {max_delayed_route['Delay Count']}")
22 print("\nDelay Reasons for this Route:")
23 print(delay_reason_counts)
24
```

Route with the most delays:

Route: Liverpool Lime Street -> London Euston

Number of Delays: 780

Delay Reasons for this Route:

Weather 499

Technical Issue 139

Staffing 45

Traffic 40

Weather Conditions 29

Signal Failure 24

Staff Shortage 4

Name: Reason for Delay, dtype: int64

In [18]:

```
1 # Filter data for 'On Time' journeys
2 on_time_data = data[data['Journey Status'] == 'On Time']
3
4 # Combine departure and arrival stations to define unique routes
5 on_time_data['Route'] = on_time_data['Departure Station'] + " -> " + on_time_data['Arrival Destination']
6
7 # Identify all unique routes in the dataset
8 all_routes = data['Departure Station'] + " -> " + data['Arrival Destination']
9 unique_routes = all_routes.unique()
10
11 # Identify routes with delays
12 delayed_routes = delayed_data_filtered['Departure Station'] + " -> " + delayed_data_filtered['Arrival Destination']
13
14 # Find routes with zero delays by excluding delayed routes
15 routes_with_zero_delays = set(unique_routes) - set(delayed_routes)
16
17 # Display routes with zero delays
18 print("Routes with zero delays:")
19 for route in routes_with_zero_delays:
20     print(route)
21
```

Routes with zero delays:

Reading -> Didcot
Liverpool Lime Street -> London St Pancras
Birmingham New Street -> Coventry
London Paddington -> Manchester Piccadilly
London Paddington -> Oxford
London St Pancras -> Birmingham New Street
Liverpool Lime Street -> Leeds
Birmingham New Street -> Wolverhampton
York -> Liverpool Lime Street
Birmingham New Street -> London St Pancras
Bristol Temple Meads -> Cardiff Central
Birmingham New Street -> Reading
Manchester Piccadilly -> London Paddington
York -> Leeds
York -> Edinburgh Waverley
Birmingham New Street -> Nuneaton
Birmingham New Street -> London Kings Cross
Manchester Piccadilly -> York
Birmingham New Street -> Edinburgh
Reading -> Swindon
Manchester Piccadilly -> London Kings Cross
Birmingham New Street -> Tamworth
London St Pancras -> Leicester
York -> Birmingham New Street
Birmingham New Street -> York
Manchester Piccadilly -> Sheffield
Birmingham New Street -> Liverpool Lime Street
London Euston -> Oxford
London St Pancras -> Wolverhampton
Reading -> Birmingham New Street
Manchester Piccadilly -> London St Pancras
Liverpool Lime Street -> Crewe
London Kings Cross -> Liverpool Lime Street
Reading -> Oxford
Liverpool Lime Street -> Birmingham New Street
Manchester Piccadilly -> Warrington
Reading -> Liverpool Lime Street
Birmingham New Street -> Stafford
Reading -> London Paddington
London Kings Cross -> Edinburgh Waverley
Birmingham New Street -> London Paddington
London Euston -> Manchester Piccadilly
York -> Peterborough
London Paddington -> Liverpool Lime Street
London Paddington -> London Waterloo
Liverpool Lime Street -> Sheffield
York -> Edinburgh

In [19]:

```
1
2 # Ensure time columns are in datetime format
3 data['Arrival Time'] = pd.to_datetime(data['Arrival Time'])
4 data['Actual Arrival Time'] = pd.to_datetime(data['Actual Arrival Time'])
5
6 # Calculate delay in minutes
7 data['Delay (Minutes)'] = (data['Actual Arrival Time'] - data['Arrival Time']).dt.total_seconds() / 60
8
9 # Filter only delayed journeys
10 delayed_data = data[data['Delay (Minutes)'] > 0]
11
12 # Function to format delay in hours and minutes
13 def format_delay(minutes):
14     hours = int(minutes // 60)
15     remaining_minutes = int(minutes % 60)
16     if hours > 0:
17         if remaining_minutes > 0:
18             return f"{hours} hour{'s' if hours > 1 else ''} and {remaining_minutes} minute{'s' if remaining_
19             return f"{hours} hour{'s' if hours > 1 else ''}"
20         return f"{remaining_minutes} minute{'s' if remaining_minutes > 1 else ''}"
21
22 # Apply the formatting function to the 'Delay (Minutes)' column
23 delayed_data['Formatted Delay'] = delayed_data['Delay (Minutes)'].apply(format_delay)
24
25 # Find the route with the maximum delay
26 max_delayed_route = delayed_data.loc[delayed_data['Delay (Minutes)'].idxmax()]
27
28 # Display the result including the reason for delay
29 print(f"Route with Maximum Delay: {max_delayed_route['Departure Station']} -> {max_delayed_route['Arrival De
30 print(f"Maximum Delay: {max_delayed_route['Formatted Delay']}")
31 print(f"Reason for Delay: {max_delayed_route['Reason for Delay']}")
32 print(f>Date : {max_delayed_route['Date of Journey']}")
33
```

Route with Maximum Delay: Manchester Piccadilly -> Leeds

Maximum Delay: 3 hours

Reason for Delay: Signal Failure

Date : 2024-01-12 00:00:00

In [20]:

```
1
2 # Calculate delay in minutes
3 data['Delay (Minutes)'] = (data['Actual Arrival Time'] - data['Arrival Time']).dt.total_seconds() / 60
4
5 # Filter only delayed journeys
6 delayed_data = data[data['Delay (Minutes)'] > 0]
7
8 # Calculate the average delay
9 average_delay = delayed_data['Delay (Minutes)'].mean()
10
11 # Function to format delay in hours and minutes
12 def format_delay(minutes):
13     hours = int(minutes // 60)
14     remaining_minutes = int(minutes % 60)
15     if hours > 0:
16         if remaining_minutes > 0:
17             return f"{hours} hour{'s' if hours > 1 else ''} and {remaining_minutes} minute{'s' if remaining_
18             return f"{hours} hour{'s' if hours > 1 else ''}"
19         return f"{remaining_minutes} minute{'s' if remaining_minutes > 1 else ''}"
20
21 # Format the average delay
22 formatted_average_delay = format_delay(average_delay)
23
24 # Display the result
25 print(f"Average Delay: {formatted_average_delay}")
26
```

Average Delay: 42 minutes

```

In [21]: 1 delayed_data['Formatted Delay'] = delayed_data['Delay (Minutes)'].apply(format_delay)
2
3 # Find the route with the maximum delay
4 min_delayed_route = delayed_data.loc[delayed_data['Delay (Minutes)'].idxmin()]
5
6 # Display the result
7 print(f"Route with Minimum Delay: {min_delayed_route['Departure Station']} -> {min_delayed_route['Arrival De
8 print(f"Minimum Delay: {min_delayed_route['Formatted Delay']}")
9 print(f"Reason for Delay: {min_delayed_route['Reason for Delay']}")
10 print(f"Date : {min_delayed_route['Date of Journey']}")
11

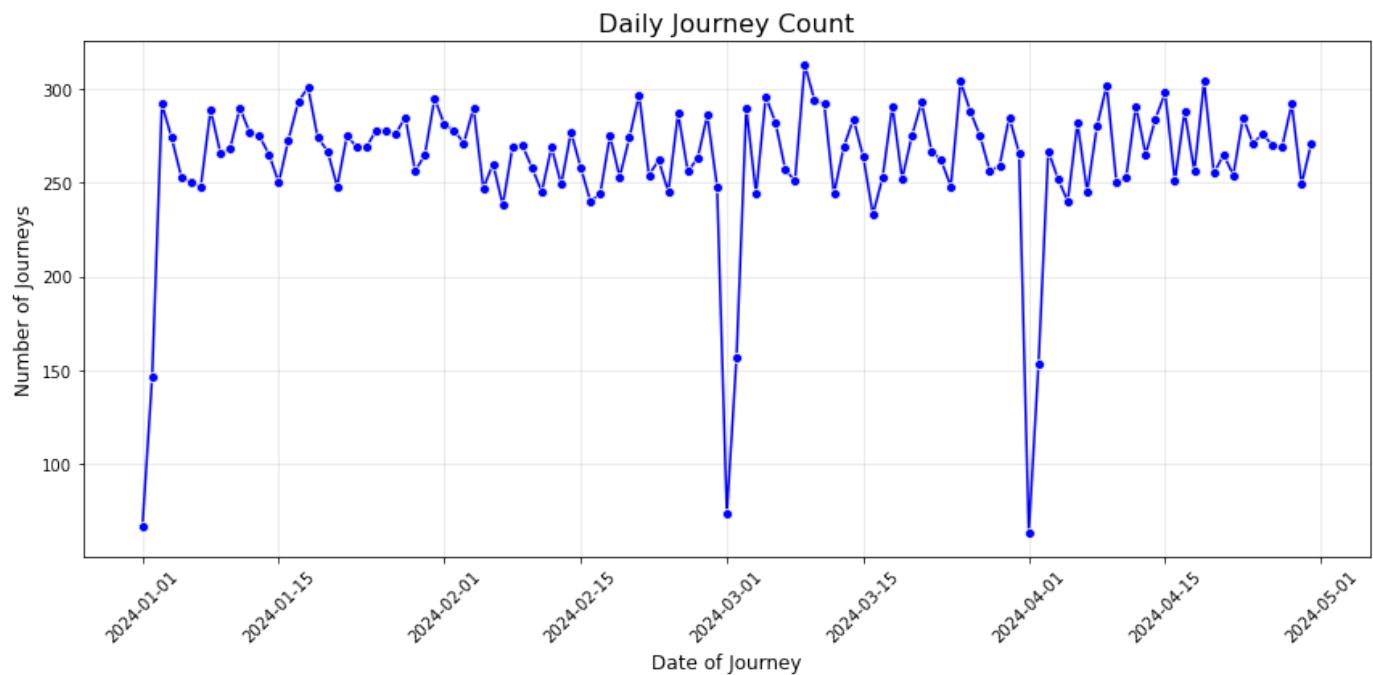
```

Route with Minimum Delay: Liverpool Lime Street -> Manchester Piccadilly
Minimum Delay: 1 minute
Reason for Delay: Technical Issue
Date : 2024-01-18 00:00:00

```

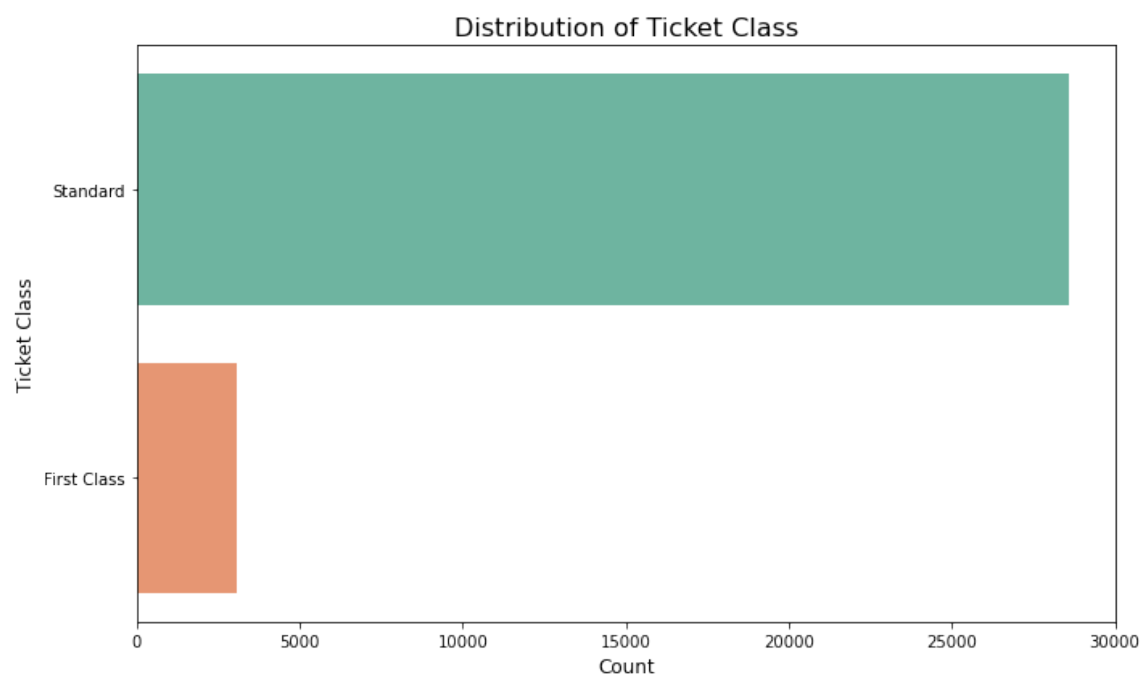
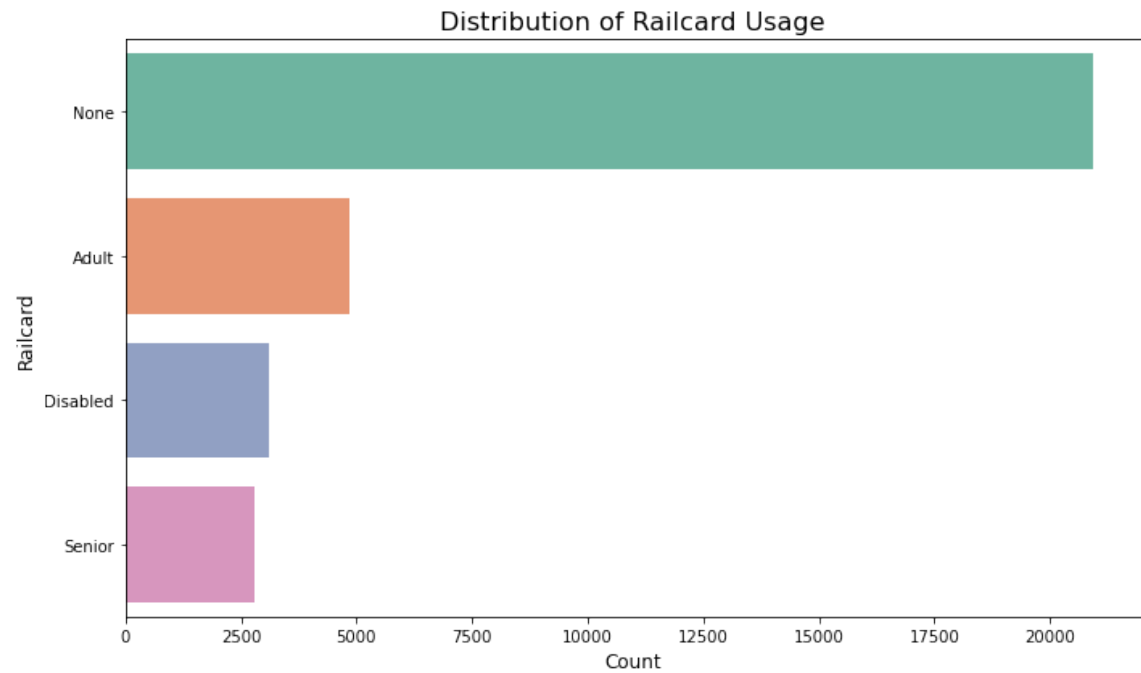
In [22]: 1 # Calculate daily journey count
2 daily_journey_count = data.groupby('Date of Journey').size().reset_index(name='Journey Count')
3
4 # Create the line plot
5 plt.figure(figsize=(12, 6))
6 sns.lineplot(data=daily_journey_count, x='Date of Journey', y='Journey Count', marker='o', color='b')
7
8 # Customize the plot
9 plt.title('Daily Journey Count', fontsize=16)
10 plt.xlabel('Date of Journey', fontsize=12)
11 plt.ylabel('Number of Journeys', fontsize=12)
12 plt.xticks(rotation=45)
13 plt.grid(alpha=0.3)
14 plt.tight_layout()
15
16 # Display the plot
17 plt.show()
18

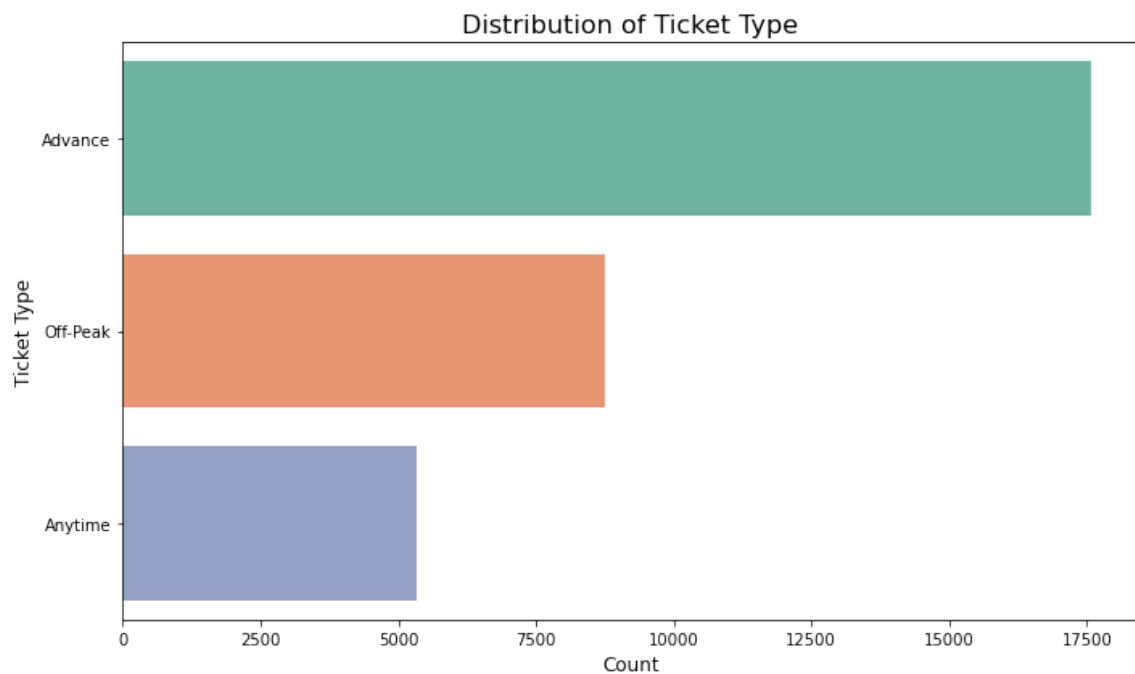
```



In [23]:

```
1 # Define a function to create bar plots
2 def plot_distribution(data, column, title):
3     plt.figure(figsize=(10, 6))
4     sns.countplot(data=data, y=column, order=data[column].value_counts().index, palette="Set2")
5     plt.title(title, fontsize=16)
6     plt.xlabel('Count', fontsize=12)
7     plt.ylabel(column, fontsize=12)
8     plt.tight_layout()
9     plt.show()
10
11 # Plot for Railcard
12 plot_distribution(data, 'Railcard', 'Distribution of Railcard Usage')
13
14 # Plot for Ticket Class
15 plot_distribution(data, 'Ticket Class', 'Distribution of Ticket Class')
16
17 # Plot for Ticket Type
18 plot_distribution(data, 'Ticket Type', 'Distribution of Ticket Type')
19
```





In [24]:

```

1 # Group data by 'Departure Station' and 'Arrival Destination' and sum the prices for each route
2 route_earnings = data.groupby(['Departure Station', 'Arrival Destination'])['Price'].sum().reset_index()
3
4 # Find the route with maximum earnings
5 max_earning_route = route_earnings.loc[route_earnings['Price'].idxmax()]
6
7 # Find the route with minimum earnings
8 min_earning_route = route_earnings.loc[route_earnings['Price'].idxmin()]
9
10 # Display the results
11 print(f"Route with Maximum Earnings: {max_earning_route['Departure Station']} -> {max_earning_route['Arrival Destination']}")
12 print(f"Maximum Earnings: £{max_earning_route['Price']:.2f}")
13
14 print(f"Route with Minimum Earnings: {min_earning_route['Departure Station']} -> {min_earning_route['Arrival Destination']}")
15 print(f"Minimum Earnings: £{min_earning_route['Price']:.2f}")
16

```

Route with Maximum Earnings: London Kings Cross -> York
Maximum Earnings: £183193.00
Route with Minimum Earnings: London Euston -> Oxford
Minimum Earnings: £41.00

In [25]:

```

1 # Ensure 'Date of Journey' is in datetime format
2
3 # Group data by 'Date of Journey' and sum the 'Price' for each date
4 daily_earnings = data.groupby('Date of Journey')['Price'].sum().reset_index()
5
6 # Find the date with maximum earnings
7 max_earning_date = daily_earnings.loc[daily_earnings['Price'].idxmax()]
8
9 # Find the date with minimum earnings
10 min_earning_date = daily_earnings.loc[daily_earnings['Price'].idxmin()]
11
12 # Display the results
13 print(f"Date with Maximum Earnings: {max_earning_date['Date of Journey'].strftime('%d-%m-%Y')}")
14 print(f"Maximum Earnings: £{max_earning_date['Price']:.2f}")
15
16 print(f"Date with Minimum Earnings: {min_earning_date['Date of Journey'].strftime('%d-%m-%Y')}")
17 print(f"Minimum Earnings: £{min_earning_date['Price']:.2f}")
18

```

Date with Maximum Earnings: 31-01-2024
Maximum Earnings: £9196.00
Date with Minimum Earnings: 01-04-2024
Minimum Earnings: £1562.00

In [26]:

```
1 # Calculate the difference between 'Date of Journey' and 'Date of Purchase'
2 data['Days Between'] = (data['Date of Journey'] - data['Date of Purchase']).dt.days
3
4 # Get the maximum and minimum gap in days
5 max_gap = data['Days Between'].max()
6 min_gap = data['Days Between'].min()
7
8 # Count tickets where 'Days Between' is 0 (purchased on the same day) and greater than 0
9 same_day_count = data[data['Days Between'] == 0].shape[0]
10 greater_than_zero_count = data[data['Days Between'] > 0].shape[0]
11
12 # Print the results
13 print(f"Maximum Gap: {max_gap} days")
14 print(f"Minimum Gap: {min_gap} days")
15 print(f"Tickets Purchased on the Same Day: {same_day_count}")
16 print(f"Tickets Purchased before Journey Day {greater_than_zero_count}")
17
```

Maximum Gap: 28 days

Minimum Gap: 0 days

Tickets Purchased on the Same Day: 14092

Tickets Purchased before Journey Day 17561

In [28]:

```
1 # Ensure delay time is calculated first
2 data['Delay (Minutes)'] = (pd.to_datetime(data['Actual Arrival Time']) - pd.to_datetime(data['Arrival Time']))
3
4 # Calculate On-Time Performance (OTP)
5 total_journeys = len(data)
6 on_time_journeys = len(data[data['Journey Status'] == 'On Time'])
7 otp = (on_time_journeys / total_journeys) * 100
8
9 # Identify the main contributing factors for delays and calculate average delay time
10 delay_reasons = data.groupby('Reason for Delay').agg(
11     delay_count=('Reason for Delay', 'size'),
12     avg_delay=('Delay (Minutes)', lambda x: round(x.mean()))
13 ).sort_values(by='delay_count', ascending=False)
14
15 # Display OTP and contributing factors with average delay time (rounded)
16 print(f"On-Time Performance (OTP): {otp:.2f}%")
17 print("\nMain Contributing Factors to Delays with Average Delay Time (rounded to nearest minute):")
18 print(delay_reasons)
```

On-Time Performance (OTP): 86.82%

Main Contributing Factors to Delays with Average Delay Time (rounded to nearest minute):

	delay_count	avg_delay
Reason for Delay		
No delay	27481	0
Weather	995	47
Signal Failure	970	52
Technical Issue	707	25
Staffing	410	26
Staff Shortage	399	75
Weather Conditions	377	31
Traffic	314	32