School of Information and Communication Technology

Griffith University

7821ICT

Green City Situation Awareness

User Manual – Delay Hours

Proof of Concept Demonstrator for Sustainable Transport

18/10/2024 T2 2024

Industry Partner: City of Gold Coast

Client: Phillip Karfs

Team members:

Omkar Dhananjay Kulkarni (s5325810)

Vlesetty Vishal Kumar (s5314434)

Viet An Nguyen (s5321461)

Zhanrui Liao (s5290972)

Table of Contents

1. INSTALLING ANACONDA AND STARTING JUPYTER

Step 1: Download and Install Anaconda

1. Visit the official Anaconda website at https://www.anaconda.com.
2. Click on the "Download" button and choose the appropriate version for your operating system (Windows, macOS, or Linux).
3. After downloading, run the installer and follow the on-screen instructions. It is recommended to install Anaconda for "All Users" (Windows) or choose the default settings (macOS/Linux).

Step 2: Verifying Anaconda Installation

1. After installation, open the terminal (macOS/Linux) or the Anaconda Prompt (Windows).
2. Type the following command to verify the installation：
   conda –version
   You should see the version number of Conda, confirming a successful installation.

Step 3: Starting Jupyter Notebook

1. Open the Anaconda Navigator from your Applications menu or by typing anaconda-navigator in the terminal/Anaconda Prompt.
2. In the Anaconda Navigator, find and click on "Launch" under the Jupyter Notebook section.
3. Alternatively, you can start Jupyter directly by typing the following command in the terminal or Anaconda Prompt:
   jupyter notebook
4. Your web browser will automatically open Jupyter's interface. You can now create or open .ipynb files to begin your work.

Step 4: Updating Anaconda and Jupyter

To ensure you're using the latest version of Anaconda and Jupyter, periodically update them with the following commands:

conda update conda
conda update anaconda
conda update jupyter

2. INTRODUCTION

This report provides an analysis of delay hours, with a specific focus on the months of July, August, and September over two years: 2023 and 2024. This analysis aims to explore the impact of the 50-cent fare initiative, which was introduced on August 6, 2024. By comparing data before and after the implementation of this initiative, the report seeks to assess its influence on travel patterns and delay trends.

3. DATASET

The data used in this analysis was sourced from the City of Gold Coast and accessed through Snowflake, a data warehousing platform. The dataset includes 12 key attributes essential for understanding travel patterns and delay analysis. These attributes are as follows:

- **Composite Key**: A unique identifier combining multiple fields to ensure data integrity.
- **Start Date Key**: The date associated with each data entry, indicating when the travel event occurred.
- **Weekday**: The day of the week for each entry, allowing for analysis based on weekday patterns.
- **Start Time Key**: The specific time when the travel event started.
- **Travel Time**: The total time taken for the trip, measured in minutes.
- **Delay**: The delay encountered during the trip, reflecting any unexpected increase in travel time.
- **Congestion**: A metric indicating the level of congestion experienced on the route.
- **Baseline Travel Time**: The expected travel time under normal, uncongested conditions.
- **Baseline Delay**: The typical delay expected based on historical data.
- **Baseline Speed**: The anticipated speed under regular conditions, facilitating comparison with actual travel speeds.
- **Baseline Excess Delay**: The additional time over the baseline delay, highlighting any significant deviations.
- **Baseline Congestion**: The expected level of congestion, used as a benchmark against actual congestion levels.

These attributes provide a comprehensive view of travel metrics before and after the implementation of the 50-cent fare initiative, offering insights into changes in public transport usage and delays over time.

4. ALGORITHMS / MACHINE LEARNING

4.1 Linear Regression

- **Description**: Linear Regression is a statistical algorithm used to model the relationship between a dependent variable and one or more independent variables. The goal is to find the line (or hyperplane in multiple dimensions) that best fits the data points. This is achieved by minimizing the sum of squared differences between observed and predicted values.
- **Applications**: It is widely used for predictive analysis in various fields, such as economics, biology, and social sciences. It's often used to understand trends, make forecasts, and evaluate relationships between variables.
- **Types**: There are two main types:
    - **Simple Linear Regression**: Involves a single independent variable.
    - **Multiple Linear Regression**: Involves multiple independent variables.

4.2 ARIMA (AutoRegressive Integrated Moving Average)

- **Description**: ARIMA is a time series forecasting algorithm that combines three components:
    - **AutoRegressive (AR)**: Uses the dependency between an observation and a certain number of lagged observations (previous time steps).
    - **Integrated (I)**: Involves differencing the observations to make the time series stationary, which means the statistical properties of the series do not depend on time.
    - **Moving Average (MA)**: Uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.
- **Applications**: ARIMA is often applied in fields such as economics, finance, and meteorology for tasks like sales forecasting, stock price prediction, and weather forecasting. It is particularly suited for data where trends, seasonality, and other temporal patterns are present.
- **Variants**:
    - **SARIMA (Seasonal ARIMA)**: An extension of ARIMA that includes a seasonal component.
    - **ARIMAX (ARIMA with eXogenous variables)**: An ARIMA model that includes other external variables to enhance forecasting.

5. PRE-PROCESSING DATA

5.1 Removing Null Value

```
[6]: df_1hour = df_1hour.dropna(subset=['TRAVEL_TIME', 'DELAY','CONGESTION','BASELINE_EXCESS_DELAY','BASELINE_CONGESTION'])
```

This code removes any rows from the df_1hour DataFrame that contain missing (NaN) values in the specified columns: 'TRAVEL_TIME', 'DELAY', 'CONGESTION', 'BASELINE_EXCESS_DELAY', and 'BASELINE_CONGESTION'. This ensures that subsequent analyses only include rows with complete data in these columns, preventing errors and inaccuracies caused by missing values.

5.2 Date And Time Formatting

```
]: # Function to convert seconds into HH:MM format
def convert_seconds_to_time(seconds):
    hours, remainder = divmod(seconds, 3600)
    minutes, _ = divmod(remainder, 60)
    return f"{int(hours):02}:{int(minutes):02}"

# Apply the function to convert START_TIME_KEY to readable time
df_1hour['START_TIME_FORM'] = df_1hour['START_TIME_KEY'].apply(convert_seconds_to_time)
# Convert the START_DATE_KEY to datetime format
df_1hour['START_DATE'] = pd.to_datetime(df_1hour['START_DATE_KEY'], format='%Y%m%d')
#Convert Travel Time to hours
df_1hour['TRAVEL_TIME_HOURS'] = df_1hour['TRAVEL_TIME'] / 3600
# Convert DELAY from seconds to hours
df_1hour['DELAY_HOURS'] = df_1hour['DELAY'] / 3600
```

This code performs several transformations on the df_1hour DataFrame to make the data more interpretable:

- **Convert Seconds to HH:** The convert_seconds_to_time function is defined to convert a given number of seconds into a HH:MM format string. This function is applied to the 'START_TIME_KEY' column to create a new column 'START_TIME_FORM', which represents the start time in a human-readable format.
- **Convert Date Key to Date Format:**
    - The 'START_DATE_KEY' column, which contains date values in YYYYMMDD format, is converted to a datetime format and saved as a new column 'START_DATE'.
- **Convert Travel Time to Hours:** The 'TRAVEL_TIME' column, originally in seconds, is converted to hours by dividing by 3600, resulting in a new column 'TRAVEL_TIME_HOURS'.
- **Convert Delay to Hours:** Similarly, the 'DELAY' column, initially in seconds, is converted to hours, resulting in a new column 'DELAY_HOURS'.

These transformations make the dataset easier to analyse and interpret, especially when visualizing trends or summarizing delays and travel times in terms of hours.

5.3 Splitting the dataset

```
data_2023 = df_1hour[(df_1hour['START_DATE'] >='2023-06-01') & (df_1hour['START_DATE'] <= '2023-09-17')]
data_2024 = df_1hour[(df_1hour['START_DATE'] >='2024-06-01') & (df_1hour['START_DATE'] <= '2024-09-17')]
```

This code filters the df_1hour DataFrame to create two new DataFrames for 2023 and 2024 from June to September:

- **data_2023:** Contains data from June 1 to September 17, 2023.
- **data_2024:** Contains data from the same period in 2024.

6. VISUALISATION AND INTERPRETATION

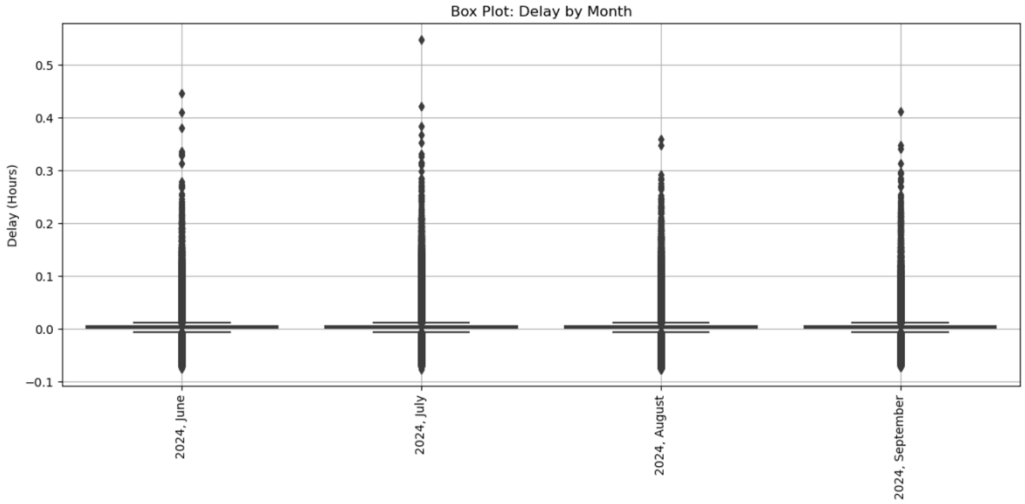
6.1 Comparing Delay by Month in 2024

```
# Extract month from START_DATE
data_2024['Month'] = data_2024['START_DATE'].dt.to_period('M').dt.strftime('%Y, %B')

# Define the correct order for months
month_order = ['2024, June', '2024, July', '2024, August', '2024, September']

# Create a box plot of delay by month
plt.figure(figsize=(12, 6))
sns.boxplot(x='Month', y='DELAY_HOURS', data=data_2024, order=month_order)
plt.title('Box Plot: Delay by Month')
plt.xlabel('')
plt.ylabel('Delay (Hours)')
plt.xticks(rotation=90)
plt.grid(True)
plt.tight_layout()
plt.show()
```

This code extracts the month and year from the 'START_DATE' column in the `data_2024` DataFrame, formatting it as "YYYY, Month" and saving it in a new 'Month' column. It then specifies a custom order for the months in `month_order`. A box plot is created using Seaborn (`sns.boxplot`), plotting 'DELAY_HOURS' against 'Month' while arranging the months based on the predefined order. The plot's title is set as "Box Plot: Delay by Month," with appropriate axis labels, gridlines, and rotated x-axis tick labels for better readability.

This code creates a box plot to visualize the delay hours by month for the year 2024:

- A box plot is generated with `Month` on the x-axis and `DELAY_HOURS` on the y-axis.
- It provides insights into the distribution of delay hours across each month, helping to identify patterns, such as the impact of the 50-cent initiative or seasonal variations.



Observation:

- While June and September show minimal disruptions, July has the highest delay hours and outliers. This could indicate specific factors during this period (such as increased road usage or construction) contributing to delays.
- August, following the 50-cent initiative, reflects some delays, but not as significant as July. It is possible that the initiative encouraged alternative travel times or transportation methods, reducing some of the delays

6.2 Hours Delay Trend in 2024

```
1]: # Handle the case where START_TIME_FORM is  "HH:MM:SS"
data_2024['START_TIME_FORM'] = pd.to_datetime(data_2024['START_TIME_FORM'], format='%H:%M:%S', errors='coerce').fillna(pd.to_datetime(data_2024['STAF

# Define the function for categorizing times based on business hours
def categorise_traffic_time(hour):
    if 8 <= hour < 10:
        return '8am-10am: Morning Peak'
    elif 15 <= hour < 18:
        return '3pm-6pm: Evening Peak'
    elif 10 <= hour < 15:
        return '10am-3pm: Midday Traffic'
    elif 18 <= hour < 20:
        return '6pm-8pm: Late Evening Traffic'
    elif 0 <= hour < 6:
        return '12am-6am: Night/Early Morning Traffic'
    elif 6 <= hour < 8:
        return '6am-8am: Early Morning Traffic'
    else:
        return '8pm-12am: Night Traffic'

data_2024['Categorise Traffic Time'] = data_2024['START_TIME_FORM'].apply(categorise_traffic_time)

# Plot
plt.figure(figsize=(12, 6))
sns.boxplot(x='Categorise Traffic Time', y='DELAY_HOURS', data=data_2024)
plt.title('Hourly Delay Trends')
plt.xlabel('')
plt.ylabel('Delay (Hours)')
plt.xticks(rotation=90)
plt.grid(True)
plt.tight_layout()
plt.show()
```

This code effectively categorizes each time segment in the data_2024 dataset into one of seven defined traffic time categories and visualizes the delay trends for each category. Here's a step-by-step explanation of what the code does:

- **Convert START_TIME_FORM to Hour Format**: The code first converts the START_TIME_FORM column into hours. It handles cases where the time might be formatted as either "HH:MM:SS" or "HH:MM".

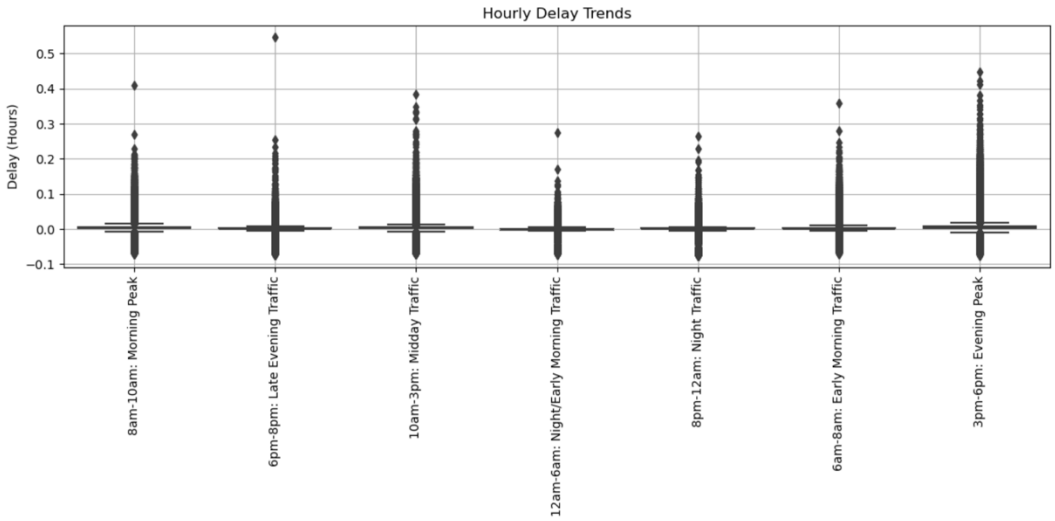
- **Categorize Time into Traffic Segments**: The categorise_traffic_time function is defined to map each hour into one of the seven traffic time categories:

    - 6 AM - 8 AM: Early Morning Traffic
    - 8 AM - 10 AM: Morning Peak
    - 10 AM - 3 PM: Midday Traffic
    - 3 PM - 6 PM: Evening Peak
    - 6 PM - 8 PM: Late Evening Traffic
    - 8 PM - 12 AM: Night Traffic
    - 12 AM - 6 AM: Night/Early Morning Traffic

- Finally, the code generates a box plot using Seaborn (sns.boxplot) to visualize the distribution of 'DELAY_HOURS' across the categorized traffic times. The plot is titled "Hourly Delay Trends" with rotated x-axis labels for readability, a grid for clarity, and a clean layout.

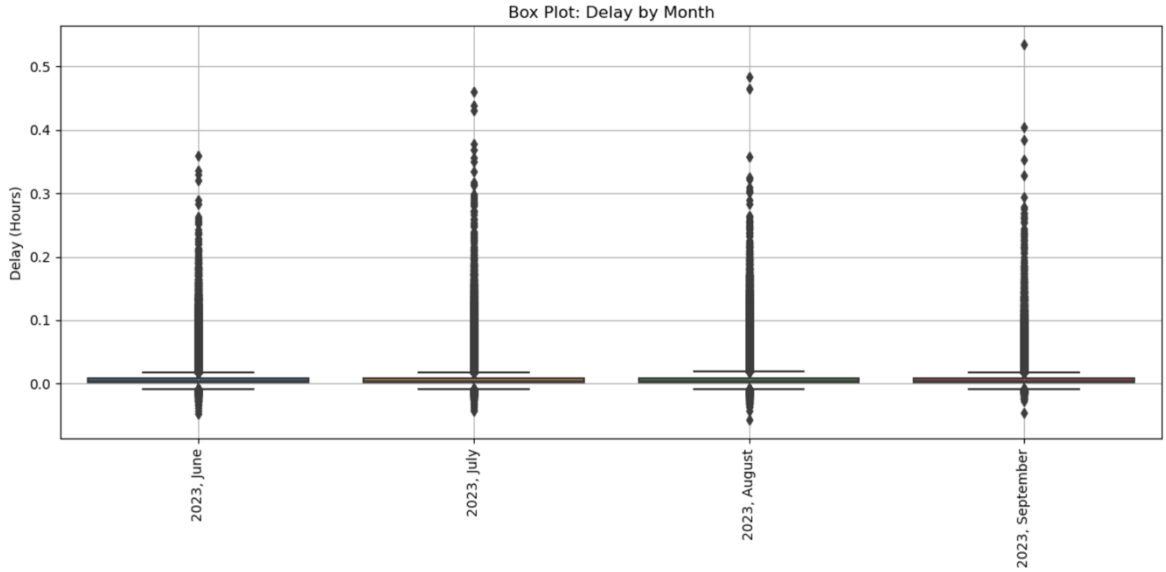
- **Visualize Hourly Delay Trends**:

    - A box plot is created to display the distribution of delay hours for each traffic time category.
    - The x-axis shows the different traffic time categories, and the y-axis displays the delay in hours.
    - This plot provides insight into how delay times vary across different times of the day, highlighting which time segments experience the most or least delays.



- **Morning and Evening Peaks**: These segments, especially 8am-10am and 3pm-6pm, show increased delay variability and higher frequencies of outliers, aligning with typical rush hour patterns.
- **Stable Periods**: Night/Early Morning and Midday traffic times (12am-6am and 10am-3pm) show stable travel times with minimal delays, highlighting them as the least congested periods.

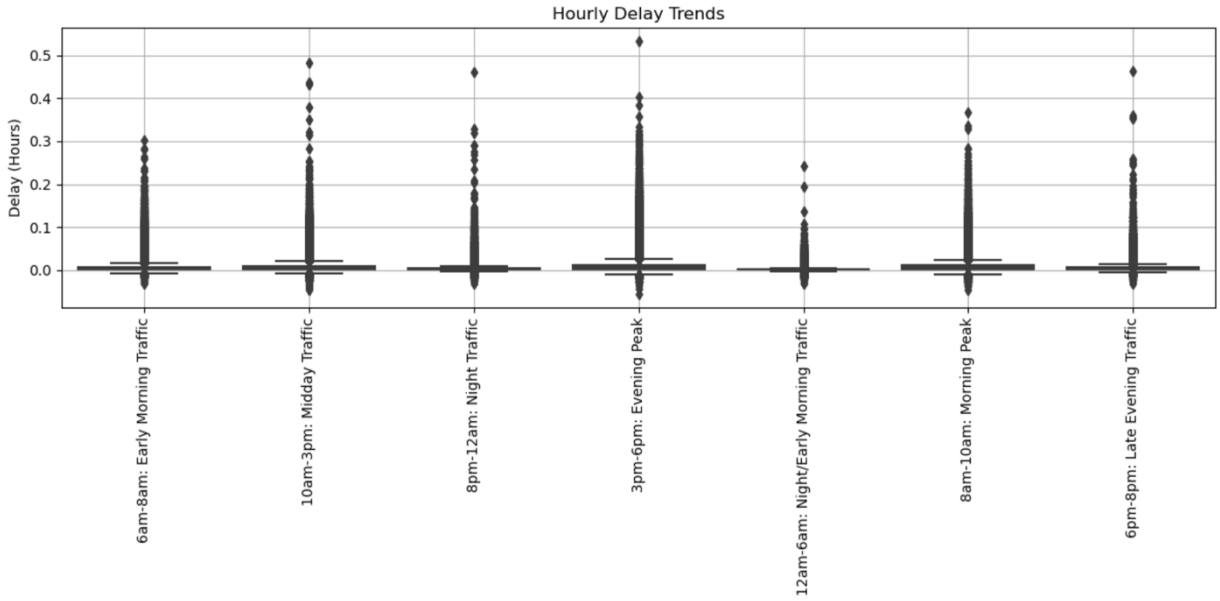
6.3 Comparing Delay by Month in 2023



Observation:

- **Higher Delays in July**: The most significant delays occurred in July, likely due to increased traffic congestion or external factors causing unexpected delays.
- **Stable Traffic Conditions in June and September**: These months exhibit lower variability in delays, suggesting more stable traffic conditions.
- **Outliers and Variability**: The presence of outliers, especially in July and August, indicates occasional severe delays that might be linked to specific days or events.

6.4 Hours Delay Trend in 2023



Observation:

- **Peak Hours (8am-10am, 3pm-6pm)**: The morning and evening peaks show the most delay variability, with the evening peak (3pm-6pm) having the most significant outliers. This suggests higher congestion during these times due to typical commuting patterns.
- **Non-Peak Hours (6pm-8pm, 8pm-12am, 12am-6am)**: The later evening and early morning hours show relatively low delays, reflecting smoother traffic conditions outside of peak hours.
- **Midday Variability**: The midday segment (10am-3pm) also exhibits notable delay variability, potentially due to random events like deliveries, construction, or lunchtime activities.

Comparing 2024 with 2023:

- **Median Delays:**
  - **In 2023**, there was a clear upward trend in the median delays from June to September, indicating a progressive increase in congestion as the year advanced.
  - **In 2024**, the median delays remained relatively consistent across all four months, showing only minor fluctuations. This suggests more stable traffic patterns and potentially better traffic flow management compared to 2023.
- **Variability in Delays**:
  - **In 2023,** the variability in delays, as shown by the interquartile range (IQR), increased from June to September, with September having the largest spread. This indicates that September experienced the highest traffic inconsistency and congestion.
  - **In 2024,** the variability was more balanced across all months, with smaller fluctuations in both the IQR and the number of outliers. This suggests a more controlled and consistent traffic flow throughout the year.
- **Outliers (Extreme Delays):**
  - **In 2023** saw a sharp increase in outliers, particularly in September, indicating more frequent extreme delays.

- **In 2024**, there was a noticeable reduction in the number of outliers, especially in the later months, suggesting fewer instances of severe traffic delays and a more predictable travel experience overall.

- **Negative Delays (Faster Travel Times):**
  - The occurrence of **negative delays** (indicating faster-than-expected travel times) was higher in **August and September 2024** compared to the same months in 2023. This points to reduced travel times and improved traffic conditions in 2024.
- **Hourly Trends:**
  - Similarly, in the hourly delay trends for 2024, the proportion of negative delays was higher compared to 2023, especially during peak times. This implies that congestion during peak hours was less intense in 2024, with some routes even experiencing faster travel times than anticipated.

Conclusion:

- Both years experienced traffic delays, but the nature of these delays differed significantly. In **2023**, delays were more variable and pronounced, especially during the month of September, which had the highest levels of congestion and outliers. In contrast, **2024** showed more evenly distributed delays across all months, with fewer extreme cases and more frequent instances of faster travel times.

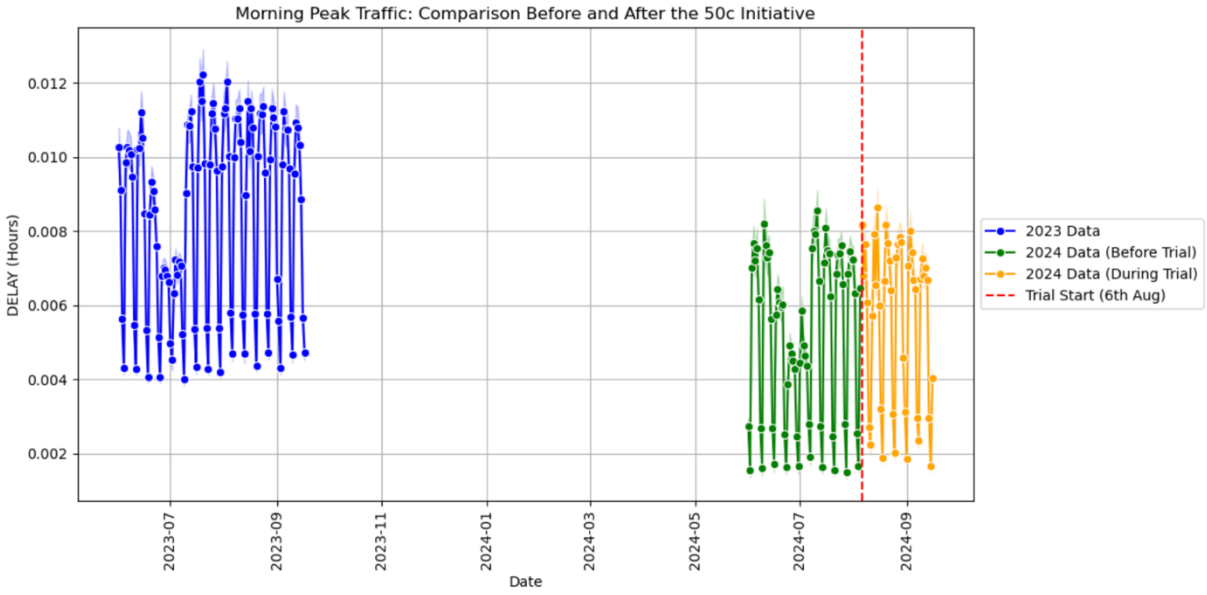
- The increase in **negative delays** during peak times and the months of **August and September 2024** suggests an overall reduction in congestion, possibly influenced by initiatives like the **50c Public Transport Fare**. This change could have encouraged a shift to public transport, thereby easing traffic on the roads and reducing delays.

6.4.1 Actual Morning Peak

[18]:
```
# Filter the dataset to include only '8am-10am: Morning Peak' for 2023 and 2024
df_2023_morning = data_2023[data_2023['Categorise Traffic Time'] == '8am-10am: Morning Peak']
df_2024_morning = data_2024[data_2024['Categorise Traffic Time'] == '8am-10am: Morning Peak']

# Define the dates for trial start (6th Aug 2024) and pre-trial data range
trial_start_date = '2024-08-06'

# Split the 2024 data into "before" and "during" the 50c initiative
df_2024_before_trial = df_2024_morning[df_2024_morning['START_DATE'] < trial_start_date]
df_2024_during_trial = df_2024_morning[df_2024_morning['START_DATE'] >= trial_start_date]

# Plot the full Morning Peak data for 2023 and 2024
plt.figure(figsize=(12, 6))

# Plot for 2023 Morning Peak
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_2023_morning, marker="o", label='2023 Data', color='blue')

# Plot for 2024 Morning Peak (before trial)
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_2024_before_trial, marker="o", label='2024 Data (Before Trial)', color='green')

# Plot for 2024 Morning Peak (during trial)
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_2024_during_trial, marker="o", label='2024 Data (During Trial)', color='orange')

# Add a vertical red dashed line to mark the start of the trial
highlight_date = pd.to_datetime('2024-08-06')
plt.axvline(x=highlight_date, color='red', linestyle='--', label='Trial Start (6th Aug)')

# Customise the plot
plt.title('Morning Peak Traffic: Comparison Before and After the 50c Initiative')
plt.xlabel('Date')
plt.ylabel('DELAY (Hours)')
plt.xticks(rotation=90)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)

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

This code compares traffic delays during the morning peak hours (8am-10am) for 2023 and 2024, particularly focusing on the impact of the 50-cent public transport initiative that started on August 6, 2024. Here's how it works:

- **Filtering Data for Morning Peak**: It filters the data_2023 and data_2024 DataFrames to include only records where the 'Categorise Traffic Time' is '8am-10am: Morning Peak'. This results in df_2023_morning and df_2024_morning datasets, respectively.
- **Splitting 2024 Data**: The code defines trial_start_date as August 6, 2024, marking the beginning of the 50-cent initiative. It then splits the 2024 morning data into two parts:
    - **Before Trial**: Data before August 6, 2024 (df_2024_before_trial).
    - **During Trial**: Data from August 6, 2024, onward (df_2024_during_trial).
- **Plotting the Data**: It creates a line plot comparing the morning peak delays in 2023 and 2024:
    - **2023 Morning Peak**: Plotted in blue, showing delay data for 2023.
    - **2024 Morning Peak (Before Trial)**: Plotted in green, showing delays before the initiative in 2024.
    - **2024 Morning Peak (During Trial)**: Plotted in orange, showing delays during the initiative period in 2024.
- **Highlighting the Trial Start**: A vertical dashed red line is added at August 6, 2024, to visually mark the start of the 50-cent trial.
- **Customization**: The plot includes a title ("Morning Peak Traffic: Comparison Before and After the 50c Initiative"), rotated x-axis date labels, and a legend placed outside the plot for better clarity. Gridlines and a tight layout are applied to make the plot visually clean.



Observation:

- **Pronounced Peaks in 2023:** The 2023 data shows significantly more frequent and higher peaks in traffic delays, especially in July and August. Delays often rise to values between 0.010 to 0.012 hours, indicating heavier congestion during several days in the morning peak hours.

- **Lower Delays in 2024:** In 2024, the delays are generally lower, with most values staying below 0.010 hours. There is a notable reduction in fluctuation compared to 2023, and the peaks are much less pronounced, suggesting an improvement in traffic conditions.
- **Consistency in 2024:** The delays in 2024 appear more consistent and smoother, with fewer significant spikes. This stability indicates that traffic conditions were more predictable and stable during the Morning Peak hours in 2024 compared to 2023.
- **Pre-Trial Period (June 1 - August 5, 2024):** Traffic delays before the 50c initiative (June to early August 2024) were already lower compared to the same period in 2023, showing improved traffic flow during this period.
- **Effect of the 50c Initiative (Post-August 6, 2024):** After the introduction of the 50c fare trial, the delays stabilized even further, with no significant spikes. The pattern of delays becomes even more consistent, with a slight reduction in overall delay levels compared to both the pre-trial period in 2024 and the corresponding period in 2023.
- **Weekday vs Weekend Influence:** The fluctuating peaks that are still visible in both 2023 and 2024 may reflect differences between weekday and weekend traffic patterns. However, these peaks are more pronounced in 2023.

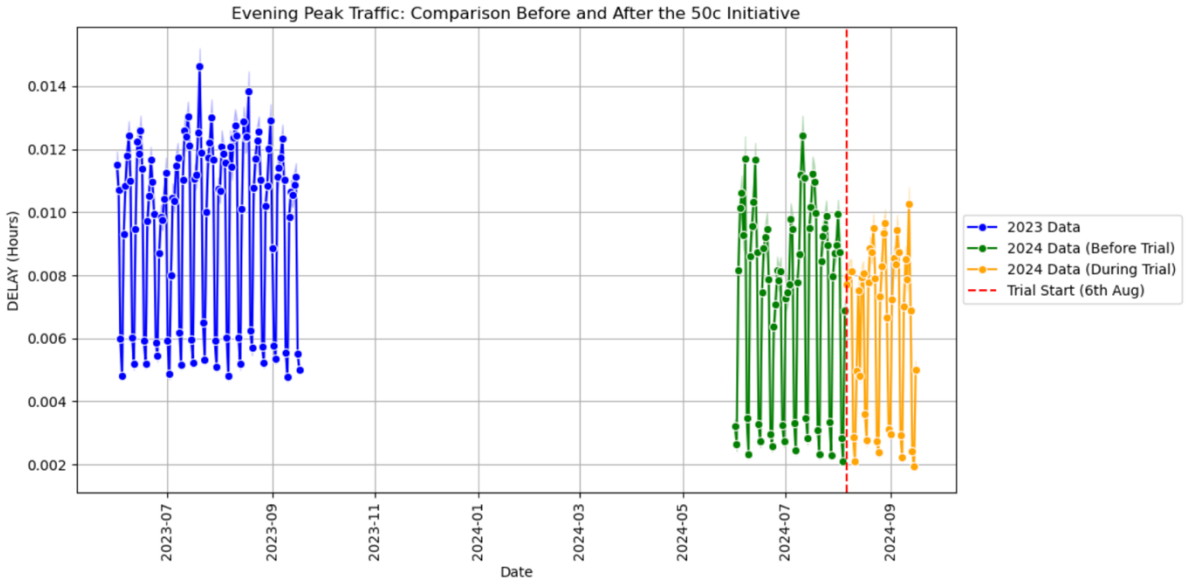
Conclusion:
While both 2023 and 2024 follow similar patterns in terms of overall delay trends, 2024 consistently exhibits lower traffic delays. The introduction of the **50c fare initiative** seems to have played a role in further reducing and stabilizing traffic delays, suggesting that the initiative positively impacted reducing congestion, particularly during the Morning Peak hours.

6.4.2 Actual Evening Peak

[9]:
```
# Assuming 'data_2023' and 'data_2024' are your dataframes with START_DATE and DELAY_HOURS columns
# And 'Categorise Traffic Time' column contains '3pm–6pm: Evening Peak'

# Filter the dataset for '3pm–6pm: Evening Peak'
df_2023_evening = data_2023[data_2023['Categorise Traffic Time'] == '3pm–6pm: Evening Peak']
df_2024_evening = data_2024[data_2024['Categorise Traffic Time'] == '3pm–6pm: Evening Peak']

# Split 2024 data into before and during the trial periods
df_2024_before_trial = df_2024_evening[(df_2024_evening['START_DATE'] >= '2024-06-01') & (df_2024_evening['START_DATE'] <= '2024-08-05')]
df_2024_during_trial = df_2024_evening[(df_2024_evening['START_DATE'] >= '2024-08-06') & (df_2024_evening['START_DATE'] <= '2024-09-16')]

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

# 2023 data
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_2023_evening, marker="o", label='2023 Data', color='blue')

# 2024 before the trial
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_2024_before_trial, marker="o", label='2024 Data (Before Trial)', color='green')

# 2024 during the trial
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_2024_during_trial, marker="o", label='2024 Data (During Trial)', color='orange')

# Add a vertical red line at the trial start date (6th August)
highlight_date = pd.to_datetime('2024-08-06')
plt.axvline(x=highlight_date, color='red', linestyle='--', label='Trial Start (6th Aug)')

# Customise the plot
plt.title('Evening Peak Traffic: Comparison Before and After the 50c Initiative')
plt.xlabel('Date')
plt.ylabel('DELAY (Hours)')
plt.xticks(rotation=90)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)

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

This code compares traffic delays during the evening peak hours (3pm-6pm) for 2023 and 2024, focusing on the impact of the 50-cent public transport initiative, which began on August 6, 2024. Here's how it works:

- **Filtering Data for Evening Peak**: It filters the `data_2023` and `data_2024` DataFrames to include only records where the 'Categorise Traffic Time' is '3pm-6pm: Evening Peak'. This results in `df_2023_evening` and `df_2024_evening` datasets, respectively.
- **Splitting 2024 Data**: The code defines `trial_start_date` as August 6, 2024, marking the beginning of the 50-cent initiative. It splits the 2024 evening data into two parts:
    - **Before Trial**: Data from June 1 to August 5, 2024 (`df_2024_before_trial`).
    - **During Trial**: Data from August 6 to September 16, 2024 (`df_2024_during_trial`).
- **Plotting the Data**:
    - **2023 Evening Peak**: Plotted in blue to show delay data for 2023.
    - **2024 Evening Peak (Before Trial)**: Plotted in green, showing delays before the initiative in 2024.
    - **2024 Evening Peak (During Trial)**: Plotted in orange, showing delays during the initiative period in 2024.
- **Highlighting the Trial Start**: A vertical dashed red line marks August 6, 2024, to indicate when the 50-cent initiative started.
- **Customization**: The plot includes a title ("Evening Peak Traffic: Comparison Before and After the 50c Initiative"), rotated x-axis labels for dates, a legend positioned outside the plot for clarity, gridlines, and a tight layout for a clean visual presentation.



Observation:

- **2023 Data:** Delays during the Evening Peak in 2023 show frequent peaks, with consistent delays of around 0.012 hours throughout July and August, indicating more pronounced traffic congestion.
- **2024 Data (Before Trial):** In the pre-trial period (June to August 5, 2024), delays are notably lower compared to the same period in 2023. The peaks are less frequent and less severe, suggesting improved traffic conditions before the 50c fare initiative.

- **2024 Data (During Trial):** Following the start of the trial on August 6, there is a clear reduction in the magnitude of delays. The overall trend from August to September shows fewer pronounced peaks and more consistent, stabilized delays, indicating further improvements in traffic conditions during the trial period.

Conclusion:

The 50c fare trial appears to have contributed significantly to reducing traffic delays during the Evening Peak. Both the frequency and magnitude of delays have decreased post-trial, with traffic patterns becoming more stable and consistent compared to both the pre-trial months in 2024 and the same period in 2023. This suggests that the trial has successfully encouraged greater public transport use, helping alleviate road congestion and improving overall traffic flow.

7. RESULT OF ALGORITHMS

7.1 Prediction using Linear Regression

- Filtering the dataset to include only the Peak traffic times for the years 2023 and 2024, then breaking this down into training (pre-August 2024) and test (post-August 2024) sets.
- In this case, since prediction is over time, our main feature will be time (likely as ordinal date values) and the target variable will be delayed in hours.
- Ensuring there are no missing values or outliers in the dataset, as these can affect the accuracy of the model.
- Using Simple Linear Regression to model the relationship between time (as the independent variable) and delay (hours) (as the dependent variable).
- Using the trained model to predict the delays for August to December 2024. This will give us insight into whether delays are expected to increase or decrease, and how they trend after the 50c fare initiative.
- Visualisation of both the actual data (June-September) and the predicted data (October-December) in the same plot, highlighting any noticeable trends.
- Observation whether the delay is expected to decrease due to the 50c initiative (which might lead to reduced road traffic and less congestion). Then, evaluation if the predicted delay will eventually stagnate after a certain point, indicating the limit of the 50c fare's impact on congestion reduction.

Approach	Training Data	Testing Data	Pros	Cons
1. Use 2023 data only	All of 2023	August-September 2024	-Model learns full seasonality of an entire year. - Good baseline from previous year.	-Model may not capture new traffic dynamics in 2024 - Could underperform if 2024 traffic patterns differ significantly from 2023.
2. Use June-July 2024 only	June-July 2024	August-September 2024	- Recent and relevant data, reflective of current traffic trends. - Focused on pre-trial 2024 behaviour.	- Misses out on yearly seasonality trends from 2023. - Model may overfit to two months of limited data.

3. Use 2023 + June-July 2024 (proposed)	All of 2023 + June-July 2024	August-September 2024	- Captures full year seasonality from 2023 - Includes recent pre-trial 2024 data. - Balanced and likely to generalize well	- Potential challenge if post-trial behaviour is very different. - Limited 2024 data for training (only two months).
4. Use all of 2024 (June-September)	June-July + August-September 2024	None	- Focused entirely on recent data. - No need for testing; predictions are real-time.	- Model may not learn broader patterns or seasonality. - Predictions cannot be verified since there's no test set.
5. Mixed year approach	June-August 2023 + June-July 2024	August-September 2024	- Combines data from two years (2023 + 2024) in similar time periods. - Provides a balanced comparison of before and after patterns.	- Mixing two different years may confuse the model if the traffic behaviour changes year-to-year. - Less exposure to full seasonality.

7.1.1 Linear Regression on Morning Peak

```
11]: # Filter the dataset to include only '8am-10am: Morning Peak' for 2023 and 2024
df_filtered = df[df['Categorise Traffic Time'] == '8am-10am: Morning Peak']

# Split the data into training and testing sets
# Training set: All 2023 data + 2024 data up to July
df_train = df_filtered[df_filtered['START_DATE'] <= '2024-07-31']

# Testing set: August and September 2024 (for comparison purposes)
df_test = df_filtered[(df_filtered['START_DATE'] >= '2024-08-01') & (df_filtered['START_DATE'] <= '2024-09-30')]

# Convert dates to ordinal for regression
df_train['START_DATE_ORD'] = df_train['START_DATE'].map(pd.Timestamp.toordinal)
df_test['START_DATE_ORD'] = df_test['START_DATE'].map(pd.Timestamp.toordinal)

# Prepare the training data
X_train = df_train[['START_DATE_ORD']]  # Independent variable (ordinal dates)
y_train = df_train['DELAY_HOURS']       # Dependent variable (delay hours)

# Initialize and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Prepare the dates for prediction (August to December 2024)
future_dates = pd.date_range(start='2024-08-01', end='2024-12-31')
future_dates_ordinal = future_dates.map(pd.Timestamp.toordinal)

# Predict the delay hours for August to December
X_future = np.array(future_dates_ordinal).reshape(-1, 1)
y_pred = model.predict(X_future)

# Create a DataFrame for the predicted values
df_predictions = pd.DataFrame({
    'START_DATE': future_dates,
    'DELAY_HOURS': y_pred
})

# Set the style for better aesthetics
sns.set(style='whitegrid')

# Plot the actual data (2023-2024) from the training set
plt.figure(figsize=(12, 6))
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_train, marker="o", label='Actual Data (2023-July 2024)')

# Plot the actual data from August and September 2024 (testing set)
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_test, marker="o", color='green', label='Actual Data (Aug-Sep 2024)')

# Plot the predicted values (August to December 2024)
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_predictions, marker="o", color='orange', label='Predicted (Aug-Dec 2024)')

# Add a vertical red line at 01/08/2024
highlight_date = pd.to_datetime('2024-08-01')
plt.axvline(x=highlight_date, color='red', linestyle='--', label='Trial Start (1st Aug 2024)')

# Customise the plot
plt.title('Morning Peak Traffic: Actual vs Predicted (August to December 2024)')
plt.xlabel('Date')
plt.ylabel('DELAY (Hours)')
plt.xticks(rotation=90)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```

This code performs a linear regression analysis to predict traffic delays during the morning peak hours (8am-10am) for the period of August to December 2024, based on historical data from 2023 and early 2024. Here's a step-by-step explanation of the code:

Filtering Data for Morning Peak:

- The code filters the dataset (`df`) to include only the records where the 'Categorise Traffic Time' is '8am-10am: Morning Peak'. This filtered data is stored in `df_filtered`.

Splitting the Data into Training and Testing Sets:

- **Training Set**: Contains all data from 2023 and the first seven months of 2024. This is achieved by filtering `df_filtered` for dates on or before July 31, 2024 (`df_train`).
- **Testing Set**: Contains data specifically for August and September 2024, which is used for comparison after the predictions are made (`df_test`).

Converting Dates to Ordinal Format:

- The code converts the 'START_DATE' column into ordinal numbers (the number of days since a fixed date) for both training and testing datasets.

Preparing the Training Data:

  - The independent variable (`X_train`) is set as the ordinal dates from the training set, and the dependent variable (`y_train`) is the corresponding delay hours.

Initializing and Fitting the Linear Regression Model:

- A `Linear Regression` model is created and fitted using the training data (`X_train` and `y_train`):

```
model = LinearRegression()

model.fit(X_train, y_train)
```

Preparing Future Dates for Prediction: The code generates a range of future dates from August 1 to December 31, 2024, and converts these dates to ordinal format for predictions:

```
future_dates = pd.date_range(start='2024-08-01', end='2024-12-31')
future_dates_ordinal = future_dates.map(pd.Timestamp.toordinal)
```

Making Predictions: The model predicts the delay hours for the future dates using the fitted regression mode

```
X_future = np.array(future_dates_ordinal).reshape(-1,1)
y_pred = model.predict(X_future)
```

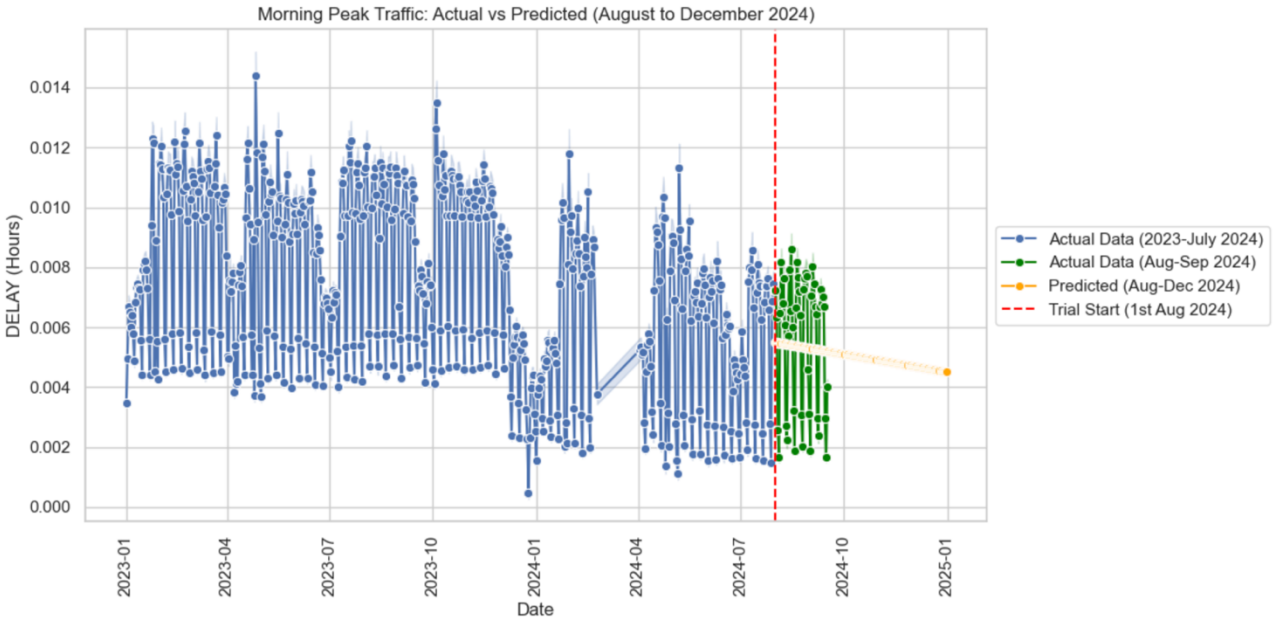
Creating a DataFrame for Predicted Values: A new DataFrame (`df_predictions`) is created that contains the future dates along with the predicted delay hours.

Setting Plot Aesthetics: The code sets the style for the plot using seaborn's whitegrid style.

Plotting the Data:

- It creates a line plot showing:
  - Actual delay data from the training set (2023 to July 2024) in blue.
  - Actual delay data from the testing set (August and September 2024) in green.
  - Predicted delay data from August to December 2024 in orange.
  - A vertical dashed red line is added to mark the start of the trial on August 1, 2024.
- **Customizing the Plot**:
  - The plot includes a title, x and y-axis labels, rotated x-axis date labels, a legend positioned outside the plot for clarity, and gridlines.
- **Displaying the Plot**:
  - Finally, the plot is rendered with a tight layout to ensure no clipping of labels.

This code provides a visual comparison of actual traffic delays and predicted delays, helping to assess the potential impact of the 50-cent public transport initiative on morning peak traffic.



Observation:

- **Initial Decline in Predictions (August - September 2024):** The prediction starts relatively close to the actual values recorded in August and September 2024. This suggests that the model has captured the immediate post-trial trend reasonably well. The slight decline in predicted delay hours aligns with the actual data, justifying the model's use of historical data patterns before the trial.

- **Steady Decrease in Delay (October - December 2024):** The linear regression model predicts a continuous decrease in traffic delay for the remainder of the year. This steady decline could be attributed to the trend observed in August and September, where delays were lower than in previous months. The model extrapolates this downward trend, likely because the data prior to the trial suggests a significant reduction in delays due to the public transport initiative.

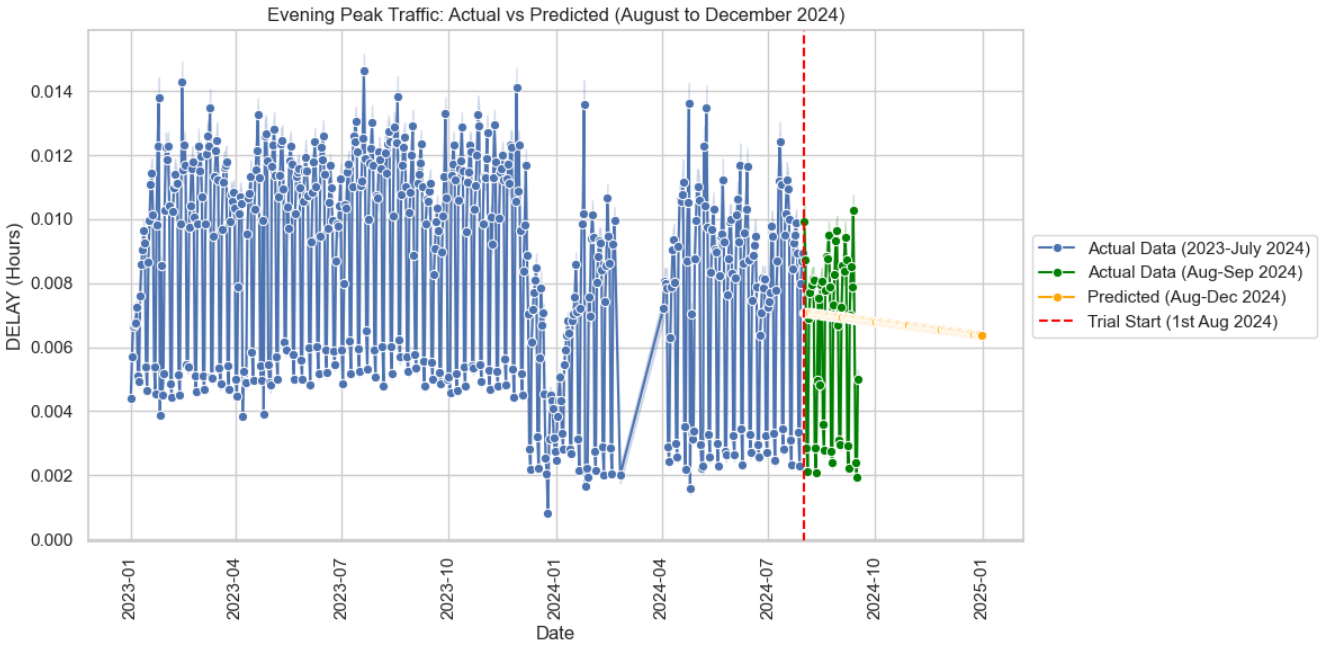
Limitation:

- **Sharp Gap between Actual and Predicted (From Mid-September Onward):** The sharp drop in predicted delays compared to the actual values indicates that the model is forecasting a rapid improvement in traffic conditions. This might be due to the model overfitting the post-trial data, where delays were relatively low, leading to overly optimistic predictions. The continuous downward trend suggests the model expects a significant ongoing impact from the 50c trial on reducing delays, though this could be a limitation of the linear approach.
- **Limitations of the Prediction:** The absence of any rebound or stabilization in the delay predictions could indicate a potential flaw in the model's ability to handle real-world variability. Given that traffic patterns can fluctuate due to factors such as holidays, weather, and other external conditions, the linear regression model may not be fully accounting for these dynamics, resulting in predictions that might be overly simplistic.

7.1.2 Linear Regression on Evening Peak

Doing the same code from Morning peak, however in the categorise traffic time change to be '3pm-6pm: Evening Peak'.

[22]:
```
# Filter the dataset to include only '3pm-6pm: Evening Peak' for 2023 and 2024
df_filtered = df[df['Categorise Traffic Time'] == '3pm-6pm: Evening Peak']

# Prepare data for linear regression (Convert dates to ordinal for regression)
df_filtered['START_DATE_ORD'] = df_filtered['START_DATE'].map(pd.Timestamp.toordinal)

# Split the data into training (up to July 2024) and testing (August to September 2024)
df_train = df_filtered[df_filtered['START_DATE'] <= '2024-07-31']
df_test = df_filtered[(df_filtered['START_DATE'] >= '2024-08-01') & (df_filtered['START_DATE'] <= '2024-09-30')]

# Train the linear regression model on training data (up to July 2024)
X_train = df_train[['START_DATE_ORD']]  # Independent variable (ordinal dates)
y_train = df_train['DELAY_HOURS']       # Dependent variable (delay hours)

# Initialize and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Prepare the future dates for prediction (August to December 2024)
future_dates = pd.date_range(start='2024-08-01', end='2024-12-31')
future_dates_ordinal = future_dates.map(pd.Timestamp.toordinal)

# Predict the delay hours for August to December
X_future = np.array(future_dates_ordinal).reshape(-1, 1)
y_pred = model.predict(X_future)

# Create a DataFrame for the predicted values
df_predictions = pd.DataFrame({
    'START_DATE': future_dates,
    'DELAY_HOURS': y_pred
})

# Set the style for better aesthetics
sns.set(style='whitegrid')

# Plot the actual data (2023-July 2024)
plt.figure(figsize=(12, 6))
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_filtered[df_filtered['START_DATE'] <= '2024-07-31'], marker="o", label='Actual Data (2023-July 2024)')

# Plot the actual data for August-September 2024
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_test, marker="o", color='green', label='Actual Data (Aug-Sep 2024)')

# Plot the predicted values (August to December 2024)
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_predictions, marker="o", color='orange', label='Predicted (Aug-Dec 2024)')

# Add a vertical red line at 01/08/2024
highlight_date = pd.to_datetime('2024-08-01')
plt.axvline(x=highlight_date, color='red', linestyle='--', label='Trial Start (1st Aug 2024)')

# Customise the plot
plt.title('Evening Peak Traffic: Actual vs Predicted (August to December 2024)')
plt.xlabel('Date')
plt.ylabel('DELAY (Hours)')
plt.xticks(rotation=90)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)

# Show the plot
plt.tight_layout()
plt.show()
```



Observation:

- Decreasing Trend: The predicted delay hours for the period from August to December 2024 (orange line) show a continuous downward trend, indicating that the model expects traffic delays to reduce steadily over time.

Limitations:

- Unlike the actual data (both 2023 and early 2024) which exhibits significant fluctuations and frequent peaks, the predicted values appear smoother. This suggests that the linear regression model is capturing a general trend but might not be as responsive to daily variability in delays.
- The actual data from August to September 2024 (green line) shows some peaks and fluctuations, which the model has not fully captured in the predictions. The predicted values are consistently lower than the actual delays for this period, suggesting that the model may be underestimating the delays.
- The model seems to predict an overly optimistic reduction, suggesting that further tuning of the model (or a more complex model) might be necessary to capture the variability better while still forecasting the delay reduction trend.

7.2 Prediction using ARIMA

7.2.1 On Morning Peak

[28]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.arima.model import ARIMA

# Step 1: Filter the dataset for '8am-10am: Morning Peak'
df_filtered = df[df['Categorise Traffic Time'] == '8am-10am: Morning Peak'].copy()

# Step 2: Filter the data for training: all data from 2023 to July 2024 (Pre-Trial Data)
df_train = df_filtered[df_filtered['START_DATE'] < '2024-08-01'].copy()

# Convert 'START_DATE' to datetime and set it as the index
df_train['START_DATE'] = pd.to_datetime(df_train['START_DATE'])
df_train.set_index('START_DATE', inplace=True)

# Ensure 'DELAY_HOURS' is numeric and drop NaN values
df_train['DELAY_HOURS'] = pd.to_numeric(df_train['DELAY_HOURS'], errors='coerce')
df_train.dropna(subset=['DELAY_HOURS'], inplace=True)

# Step 3: Fit the ARIMA model on the training data (Pre-Trial Data)
model = ARIMA(df_train['DELAY_HOURS'], order=(2, 1, 2))  # Simpler ARIMA(2,1,2) model
model_fit = model.fit()

# Step 4: Forecast the delay hours for August and September (for testing purposes)
forecast_steps_test = 61  # Forecast for 61 days (August + September)
forecast_test = model_fit.get_forecast(steps=forecast_steps_test)
forecast_values_test = forecast_test.predicted_mean

# Step 5: Prepare the dates for testing (August and September 2024)
test_dates = pd.date_range(start='2024-08-01', periods=forecast_steps_test)

# Create a DataFrame for the predicted values during the test period
df_predictions_test = pd.DataFrame({
    'START_DATE': test_dates,
    'DELAY_HOURS': forecast_values_test
})

# Step 6: Filter the actual data for the test period (August to September 2024)
df_test = df_filtered[(df_filtered['START_DATE'] >= '2024-08-01') & (df_filtered['START_DATE'] <= '2024-09-30')].copy()
df_test['START_DATE'] = pd.to_datetime(df_test['START_DATE'])
df_test.set_index('START_DATE', inplace=True)

# Step 7: Plot the results
sns.set(style='whitegrid')

# Plot actual data for training period (2023 to July 2024)
plt.figure(figsize=(12, 6))
sns.lineplot(x=df_train.index, y='DELAY_HOURS', data=df_train, marker="o", label='Actual Data (2023 - Jul 2024)')

# Plot actual data for August to September (test period)
sns.lineplot(x=df_test.index, y='DELAY_HOURS', data=df_test, marker="o", color='green', label='Actual Data (Aug-Sep)')

# Plot the ARIMA predictions for the test period (August to September 2024)
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_predictions_test, marker="o", color='orange', label='Predicted (Aug-Sep)')

# Add a vertical red line at 01/08/2024 (start of predictions)
highlight_date = pd.to_datetime('2024-08-01')
plt.axvline(x=highlight_date, color='red', linestyle='--', label='Start of Test Period (1st Aug 2024)')

# Customise the plot
plt.title('Morning Peak Traffic: ARIMA Predictions vs Actual Data (Testing Aug-Sep 2024)')
plt.xlabel('Date')
plt.ylabel('DELAY (Hours)')
plt.xticks(rotation=90)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)

# Ensure a tight layout
plt.tight_layout()
plt.show()
```

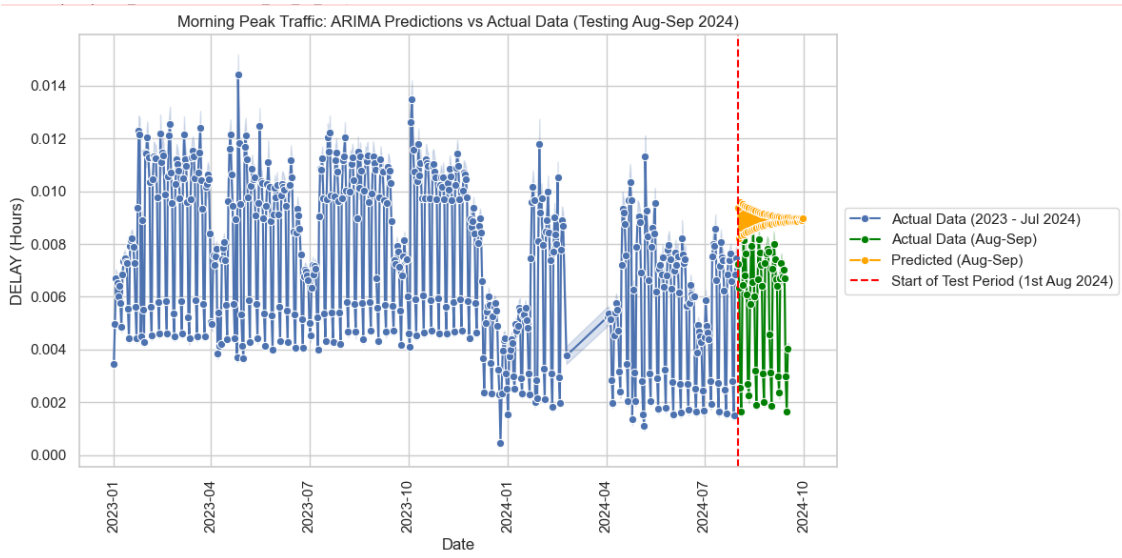
This code applies an ARIMA model to forecast Morning Peak delay hours for August and September 2024, based on historical data from 2023 through July 2024:

- **Filtering the Dataset**: The dataset (`df`) is filtered to include only records categorized under '8am-10am: Morning Peak' and stored in `df_filtered`. The `copy()` method is used to avoid modifying the original DataFrame.
- **Creating the Training Dataset**: The code filters the data to include all records from 2023 and until July 31, 2024 (pre-trial data), and stores it in `df_train`.
- **Date Handling**: The 'START_DATE' column is converted to a datetime format, and it's set as the index of `df_train`. This step is crucial for time series analysis as it allows for proper handling of date-related indexing.
- **Ensuring Numeric Delay Hours**: The 'DELAY_HOURS' column is converted to a numeric format, ensuring any non-numeric entries are coerced into NaN. The code then drops any rows with NaN values in 'DELAY_HOURS' to clean the dataset.
- **Fitting the ARIMA Model**:
    - An ARIMA model with parameters (2, 1, 2) is initialized and fitted to the training data. These parameters indicate:

        - 2 autoregressive terms
        - 1 differencing term (to make the data stationary)
        - 2 moving average terms
    - The model is then fitted using `model.fit()`, which estimates the model parameters.

- **Forecasting for Testing Period**: The code sets forecast_steps_test to 61, representing the number of days to forecast (August and September). It uses model_fit.get_forecast(steps=forecast_steps_test) to obtain the forecasted values for these days, stored in forecast_values_test.
- **Preparing Dates for Predictions**: A date range is created for the test period (August 1 to September 30, 2024) and stored in test_dates. This range corresponds to the forecasted values.
- **Creating a DataFrame for Predictions**: A new DataFrame (df_predictions_test) is created containing the forecasted values for delay hours, indexed by their corresponding dates.
- **Filtering Actual Data for Testing Period**: The actual delay data for the testing period (August to September 2024) is extracted from df_filtered, and the 'START_DATE' column is converted to datetime format and set as the index.
- **Plotting Results**:
    - The plot is styled using seaborn's whitegrid aesthetic.
    - It visualizes:
        - Actual delay data from the training period (2023 to July 2024) in blue.
        - Actual delay data for August and September 2024 (test period) in green.
        - ARIMA model predictions for the same period in orange.
    - A vertical dashed red line indicates the start of the test period on August 1, 2024.
- **Customizing the Plot**: The plot is customized with a title, x and y-axis labels, and a legend. The x-axis date labels are rotated for better readability. Gridlines are added for clarity.
- **Displaying the Plot**: Finally, `plt.tight_layout()` ensures that the layout is tight and non-overlapping, followed by `plt.show()` to render the plot.

This code effectively demonstrates how to filter, preprocess, and analyse time series data for traffic delays using an ARIMA model, followed by visualizing both actual and predicted delay hours. The results will help in understanding traffic patterns and informing decision-making for traffic management during the morning peak hours.



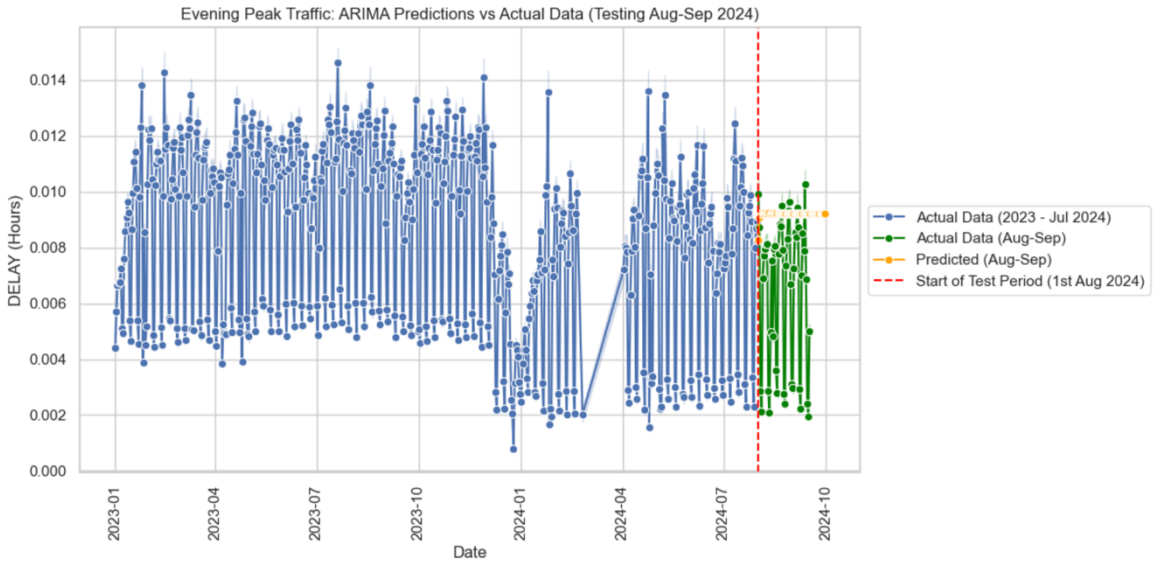
Observation and Limitation:

The predicted values are somewhat constant, fluctuating in a small range. Given the variability in the actual data, it seems the model may be underfitting the data. The ARIMA order used might not be complex enough to capture the nuances or seasonality in the dataset.

7.2.2 On Evening Peak

Doing the same code from Morning peak, however in the categorise traffic time change to be '3pm-6pm: Evening Peak'.

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.arima.model import ARIMA

# Step 1: Filter the dataset for '3pm-6pm: Evening Peak'
df_filtered = df[df['Categorise Traffic Time'] == '3pm-6pm: Evening Peak'].copy()

# Step 2: Filter the data for training: all data from 2023 to July 2024 (Pre-Trial Data)
df_train = df_filtered[df_filtered['START_DATE'] < '2024-08-01'].copy()

# Convert 'START_DATE' to datetime and set it as the index
df_train['START_DATE'] = pd.to_datetime(df_train['START_DATE'])
df_train.set_index('START_DATE', inplace=True)

# Ensure 'DELAY_HOURS' is numeric and drop NaN values
df_train['DELAY_HOURS'] = pd.to_numeric(df_train['DELAY_HOURS'], errors='coerce')
df_train.dropna(subset=['DELAY_HOURS'], inplace=True)

# Step 3: Fit the ARIMA model on the training data (Pre-Trial Data)
model = ARIMA(df_train['DELAY_HOURS'], order=(2, 1, 2))  # ARIMA(2,1,2) model
model_fit = model.fit()

# Step 4: Forecast the delay hours for August and September (for testing purposes)
forecast_steps_test = 61  # Forecast for 61 days (August + September)
forecast_test = model_fit.get_forecast(steps=forecast_steps_test)
forecast_values_test = forecast_test.predicted_mean

# Step 5: Prepare the dates for testing (August and September 2024)
test_dates = pd.date_range(start='2024-08-01', periods=forecast_steps_test)

# Create a DataFrame for the predicted values during the test period
df_predictions_test = pd.DataFrame({
    'START_DATE': test_dates,
    'DELAY_HOURS': forecast_values_test
})

# Step 6: Filter the actual data for the test period (August to September 2024)
df_test = df_filtered[(df_filtered['START_DATE'] >= '2024-08-01') & (df_filtered['START_DATE'] <= '2024-09-30')].copy()
df_test['START_DATE'] = pd.to_datetime(df_test['START_DATE'])
df_test.set_index('START_DATE', inplace=True)

# Step 7: Plot the results
sns.set(style='whitegrid')

# Plot actual data for training period (2023 to July 2024)
plt.figure(figsize=(12, 6))
sns.lineplot(x=df_train.index, y='DELAY_HOURS', data=df_train, marker="o", label='Actual Data (2023 - Jul 2024)')

# Plot actual data for August to September (test period)
sns.lineplot(x=df_test.index, y='DELAY_HOURS', data=df_test, marker="o", color='green', label='Actual Data (Aug-Sep)')

# Plot the ARIMA predictions for the test period (August to September 2024)
sns.lineplot(x='START_DATE', y='DELAY_HOURS', data=df_predictions_test, marker="o", color='orange', label='Predicted (Aug-Sep)')

# Add a vertical red line at 01/08/2024 (start of predictions)
highlight_date = pd.to_datetime('2024-08-01')
plt.axvline(x=highlight_date, color='red', linestyle='--', label='Start of Test Period (1st Aug 2024)')

# Customise the plot
plt.title('Evening Peak Traffic: ARIMA Predictions vs Actual Data (Testing Aug-Sep 2024)')
plt.xlabel('Date')
plt.ylabel('DELAY (Hours)')
plt.xticks(rotation=90)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.grid(True)

# Ensure a tight layout
plt.tight_layout()
plt.show()
```



Observations and Limitations:

- Similar to the morning peak observation, the predicted values for the evening peak are much smoother than the actual data. This is an indication that the model may be underfitting the data, as it cannot capture the short-term variations in the actual traffic delays.
- The ARIMA model's predictions stay relatively constant, while the actual evening peak delay data (in green) has considerable variance, with many peaks and valleys. The model fails to react to these abrupt changes in traffic delay, suggesting that it's not capturing some inherent patterns, such as seasonality or external factors causing traffic delays.
- Given the observations of smoothing and underfitting, we might need to try more sophisticated models, such as SARIMA (Seasonal ARIMA).