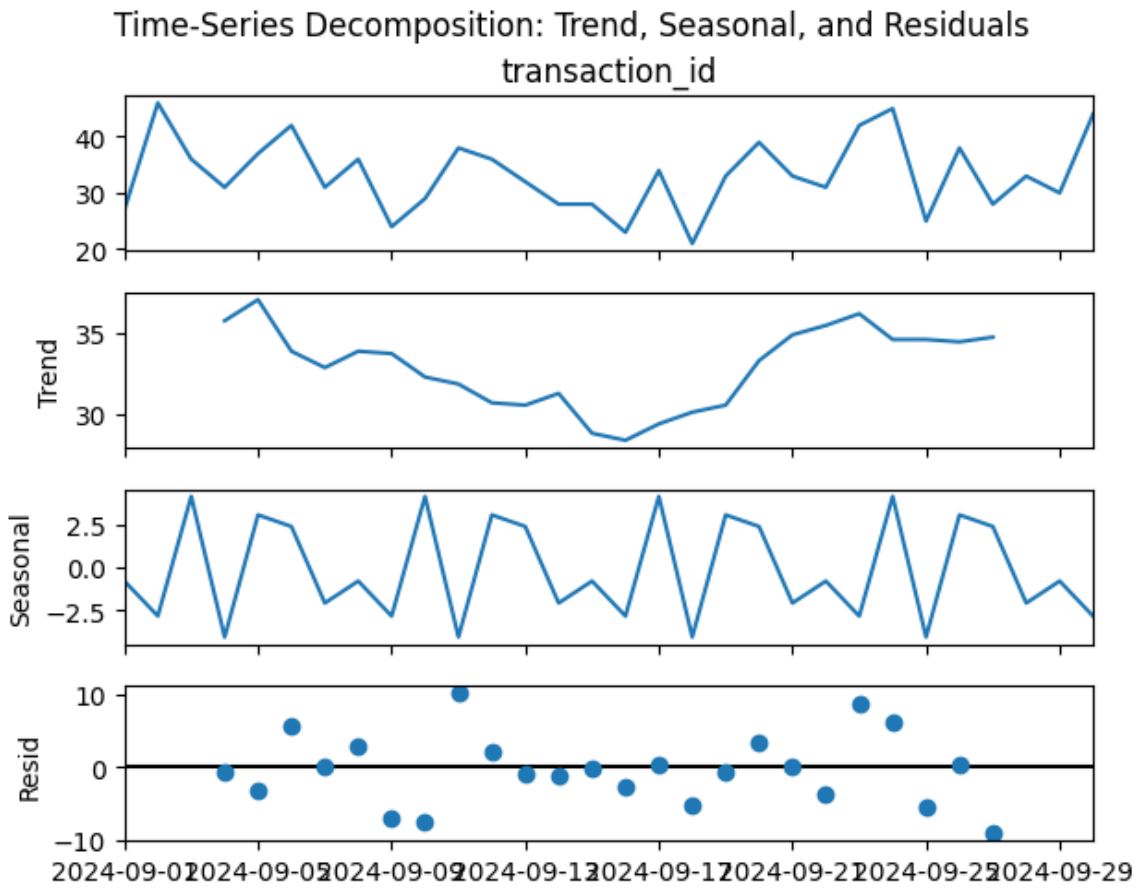
In [4]:

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from pmdarima import auto_arima
from prophet import Prophet
import numpy as np
import os

# Load Dataset
df = pd.read_csv('transaction_updated.csv')  # Replace with your dataset path

# Data Preprocessing
# Handling the date format explicitly
df['timestamp'] = pd.to_datetime(df['timestamp'], format='%Y-%m-%dT%H-%M-%S.%fZ', errors
='coerce')

# Check for any rows where parsing failed
invalid_dates = df[df['timestamp'].isnull()]
if not invalid_dates.empty:
    print("Some dates could not be parsed:")
    print(invalid_dates)

# Drop rows with invalid dates if needed
df = df.dropna(subset=['timestamp'])

# Continue with setting the index
df.set_index('timestamp', inplace=True)

# Filtering based on user input for stn no and EqN
stn_no = input("Enter station number (or 'all' for all stations): ")
eqn_no = input("Enter equipment number (or 'all' for all equipment): ")

# Apply filters
if stn_no.lower() != 'all':
    df = df[df['stn_no'] == int(stn_no)]
if eqn_no.lower() != 'all':
    df = df[df['EqN'] == int(eqn_no)]

# Aggregating the Footfall at Different Levels
daily_footfall = df.resample('D').count()  # Daily footfall

# Time-Series Decomposition
decomposition = seasonal_decompose(daily_footfall['transaction_id'], model='additive')
fig = decomposition.plot()
fig.suptitle('Time-Series Decomposition: Trend, Seasonal, and Residuals', y=1.02)
plt.show()

# Specify Forecast Period
start_date = pd.to_datetime(input("Enter forecast start date (YYYY-MM-DD): "))
end_date = pd.to_datetime(input("Enter forecast end date (YYYY-MM-DD): "))
forecast_period = (end_date - start_date).days

# File Path Base
base_path = r'C:\Users\admin\Desktop\airline\sensor-file-ridership\output of sensors'

# ARIMA Forecasting
try:
    # Automatically determine ARIMA order
    auto_model = auto_arima(daily_footfall['transaction_id'].dropna(), seasonal=False, s
tepwise=True)
    print("Auto ARIMA Model Summary:")
    print(auto_model.summary())

    # Fit ARIMA model with the best parameters
    arima_model = ARIMA(daily_footfall['transaction_id'], order=auto_model.order)
    arima_model_fit = arima_model.fit()
```

```
    # Adjust the forecast range based on the start and end dates provided
    forecast_start_index = (start_date - daily_footfall.index[-1]).days  # Days from las
t data point to start date
    arima_forecast = arima_model_fit.forecast(steps=(forecast_period + forecast_start_ind
ex))

    # Creating a DataFrame for ARIMA Forecast
    arima_forecast_dates = pd.date_range(daily_footfall.index[-1] + pd.Timedelta(days=1)
, periods=len(arima_forecast))
    arima_forecast_df = pd.DataFrame({'timestamp': arima_forecast_dates, 'ARIMA_Forecast
': arima_forecast})

    # Filter the forecast to start exactly from the user-specified start date
    arima_forecast_df = arima_forecast_df[arima_forecast_df['timestamp'] >= start_date]

    # Display ARIMA Forecasted Data
    print(f"ARIMA Model - Forecasted Footfall from {start_date.date()} to {end_date.date(
)}:")
    print(arima_forecast_df)

    # Aggregation of ARIMA Forecast
    daily_forecast = arima_forecast_df.copy()
    daily_forecast.set_index('timestamp', inplace=True)
    weekly_forecast = daily_forecast.resample('W').sum()
    monthly_forecast = daily_forecast.resample('M').sum()
    yearly_forecast = daily_forecast.resample('Y').sum()
    weekend_forecast = daily_forecast[daily_forecast.index.dayofweek >= 5].resample('D')
.sum()
    weekday_forecast = daily_forecast[daily_forecast.index.dayofweek < 5].resample('D').
sum()

    # Add week of year and day of week to weekly forecast
    weekly_forecast['week_of_year'] = weekly_forecast.index.isocalendar().week
    weekly_forecast['day_of_week'] = weekly_forecast.index.day_name()

    # Add month name to monthly forecast
    monthly_forecast['month_name'] = monthly_forecast.index.month_name()

    # Add day of week to weekend and weekday forecasts
    weekend_forecast['day_of_week'] = weekend_forecast.index.day_name()
    weekday_forecast['day_of_week'] = weekday_forecast.index.day_name()

    # Save ARIMA Forecast to CSV
    arima_forecast_df.to_csv(os.path.join(base_path, f'arima_forecast_{stn_no}_{eqn_no}.
csv'), index=False)

    # Save Aggregated ARIMA Forecast Data to CSV
    weekly_forecast.to_csv(os.path.join(base_path, f'weekly_forecast_arima_{stn_no}_{eqn_
no}.csv'))
    monthly_forecast.to_csv(os.path.join(base_path, f'monthly_forecast_arima_{stn_no}_{eq
n_no}.csv'))
    yearly_forecast.to_csv(os.path.join(base_path, f'yearly_forecast_arima_{stn_no}_{eqn_
no}.csv'))
    weekend_forecast.to_csv(os.path.join(base_path, f'weekend_forecast_arima_{stn_no}_{eq
n_no}.csv'))
    weekday_forecast.to_csv(os.path.join(base_path, f'weekday_forecast_arima_{stn_no}_{eq
n_no}.csv'))

    # Plot ARIMA Forecast
    plt.figure(figsize=(10, 6))
    plt.plot(daily_footfall.index, daily_footfall['transaction_id'], label='Observed', c
olor='blue')
    plt.plot(arima_forecast_df['timestamp'], arima_forecast_df['ARIMA_Forecast'], label=
'ARIMA Forecast', color='orange')
    plt.title(f'ARIMA Model: Observed vs Forecasted Footfall ({start_date.date()} to {end
_date.date()})')
    plt.xlabel('Date')
    plt.ylabel('Footfall')
    plt.legend()
    plt.show()
```

```
except Exception as e:
    print(f"An error occurred with ARIMA model: {e}")

# Prophet Model - Alternative Time-Series Forecasting
daily_footfall_prophet = daily_footfall.reset_index()

# Check the columns in the DataFrame
print(daily_footfall_prophet.columns)  # This will help to understand the columns before renaming

# Ensure the DataFrame only has 'timestamp' and 'transaction_id' after resetting the index
if 'timestamp' in daily_footfall_prophet.columns and 'transaction_id' in daily_footfall_prophet.columns:
    daily_footfall_prophet = daily_footfall_prophet[['timestamp', 'transaction_id']]  # Keep only necessary columns
    daily_footfall_prophet.columns = ['ds', 'y']  # Rename columns for Prophet
else:
    raise ValueError("Expected columns 'timestamp' and 'transaction_id' not found in the DataFrame after resetting index")

# Convert 'ds' to datetime and 'y' to numeric
daily_footfall_prophet['ds'] = pd.to_datetime(daily_footfall_prophet['ds'])
daily_footfall_prophet['y'] = pd.to_numeric(daily_footfall_prophet['y'], errors='coerce')

# Fit Prophet model
prophet_model = Prophet()
prophet_model.fit(daily_footfall_prophet)

# Create future dataframe starting from the user-specified start_date
future_df = pd.date_range(start=start_date, end=end_date)
future_prophet = pd.DataFrame({'ds': future_df})
forecast_prophet = prophet_model.predict(future_prophet)

# Creating a DataFrame for Prophet Forecast
prophet_forecast_df = forecast_prophet[['ds', 'yhat']]
prophet_forecast_df.columns = ['timestamp', 'transaction_id']

# Save Prophet Forecast to CSV
prophet_forecast_df.to_csv(os.path.join(base_path, f'daily_footfall_forecast_prophet_{stn_no}_{eqn_no}.csv'), index=False)

# Aggregation of Prophet Forecast
daily_forecast_prophet = prophet_forecast_df.copy()
daily_forecast_prophet.set_index('timestamp', inplace=True)
weekly_forecast_prophet = daily_forecast_prophet.resample('W').sum()
monthly_forecast_prophet = daily_forecast_prophet.resample('M').sum()
yearly_forecast_prophet = daily_forecast_prophet.resample('Y').sum()
weekend_forecast_prophet = daily_forecast_prophet[daily_forecast_prophet.index.dayofweek >= 5].resample('D').sum()
weekday_forecast_prophet = daily_forecast_prophet[daily_forecast_prophet.index.dayofweek < 5].resample('D').sum()

# Add week of year and day of week to weekly forecast
weekly_forecast_prophet['week_of_year'] = weekly_forecast_prophet.index.isocalendar().week
weekly_forecast_prophet['day_of_week'] = weekly_forecast_prophet.index.day_name()

# Add month name to monthly forecast
monthly_forecast_prophet['month_name'] = monthly_forecast_prophet.index.month_name()

# Add day of week to weekend and weekday forecasts
weekend_forecast_prophet['day_of_week'] = weekend_forecast_prophet.index.day_name()
weekday_forecast_prophet['day_of_week'] = weekday_forecast_prophet.index.day_name()

# Save Aggregated Forecast Data to CSV
weekly_forecast_prophet.to_csv(os.path.join(base_path, f'weekly_forecast_prophet_{stn_no}_{eqn_no}.csv'))
monthly_forecast_prophet.to_csv(os.path.join(base_path, f'monthly_forecast_prophet_{stn_no}_{eqn_no}.csv'))
yearly_forecast_prophet.to_csv(os.path.join(base_path, f'yearly_forecast_prophet_{stn_no}
```

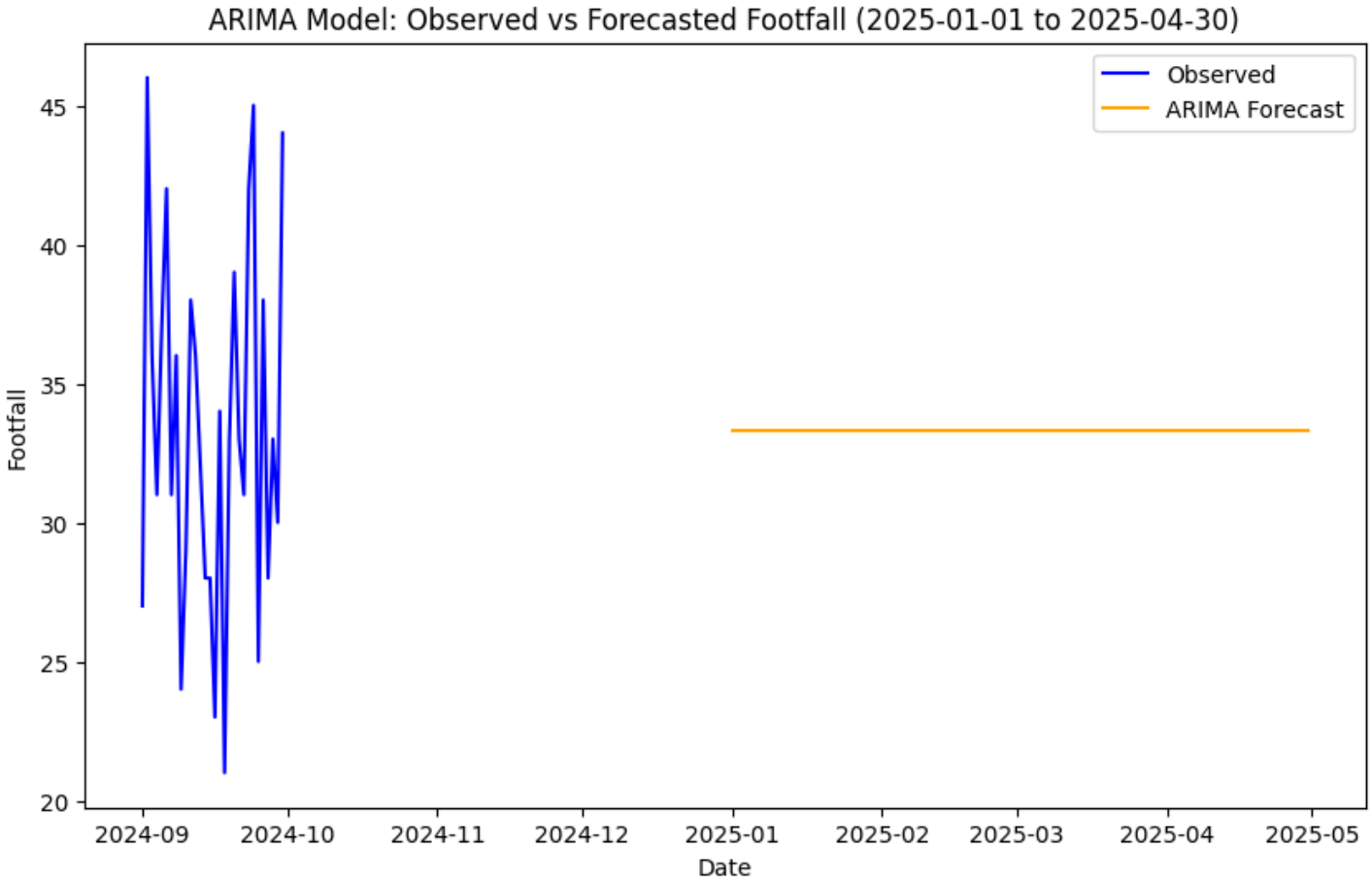
```
_{eqn_no}.csv'))
weekend_forecast_prophet.to_csv(os.path.join(base_path, f'weekend_forecast_prophet_{stn_no}_{eqn_no}.csv'))
weekday_forecast_prophet.to_csv(os.path.join(base_path, f'weekday_forecast_prophet_{stn_no}_{eqn_no}.csv'))

# Display the Prophet Forecast DataFrame
print(prophet_forecast_df.head())

print("Files saved to their respective destinations")
```




```
Auto ARIMA Model Summary:
                               SARIMAX Results
==============================================================================
Dep. Variable:                          y   No. Observations:                   30
Model:                          SARIMAX   Log Likelihood                -98.641
Date:                Mon, 09 Sep 2024   AIC                            201.282
Time:                        18:54:45   BIC                            204.085
Sample:                    09-01-2024   HQIC                           202.179
                         - 09-30-2024
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept     33.3333      1.197     27.852      0.000      30.988      35.679
sigma2        42.0222     13.558      3.100      0.002      15.450      68.595
===================================================================================
Ljung-Box (L1) (Q):                   0.16   Jarque-Bera (JB):                 0.74
Prob(Q):                              0.69   Prob(JB):                         0.69
Heteroskedasticity (H):               1.10   Skew:                             0.17
Prob(H) (two-sided):                  0.88   Kurtosis:                         2.31
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
ARIMA Model - Forecasted Footfall from 2025-01-01 to 2025-04-30:
            timestamp  ARIMA_Forecast
2025-01-01 2025-01-01       33.333328
2025-01-02 2025-01-02       33.333328
2025-01-03 2025-01-03       33.333328
2025-01-04 2025-01-04       33.333328
2025-01-05 2025-01-05       33.333328
```

```
...                 ...             ...
2025-04-26 2025-04-26       33.333328
2025-04-27 2025-04-27       33.333328
2025-04-28 2025-04-28       33.333328
2025-04-29 2025-04-29       33.333328
2025-04-30 2025-04-30       33.333328

[120 rows x 2 columns]
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_13212\2154472396.py:86: FutureWarning: 'M' is
deprecated and will be removed in a future version, please use 'ME' instead.
  monthly_forecast = daily_forecast.resample('M').sum()
C:\Users\admin\AppData\Local\Temp\ipykernel_13212\2154472396.py:87: FutureWarning: 'Y' is
deprecated and will be removed in a future version, please use 'YE' instead.
  yearly_forecast = daily_forecast.resample('Y').sum()
```




```
Index(['timestamp', 'transaction_id', 'EqN', 'stn_no'], dtype='object')
```

```
18:54:46 - cmdstanpy - INFO - Chain [1] start processing
18:54:46 - cmdstanpy - INFO - Chain [1] done processing
C:\Users\admin\AppData\Local\Temp\ipykernel_13212\2154472396.py:162: FutureWarning: 'M' i
s deprecated and will be removed in a future version, please use 'ME' instead.
  monthly_forecast_prophet = daily_forecast_prophet.resample('M').sum()
C:\Users\admin\AppData\Local\Temp\ipykernel_13212\2154472396.py:163: FutureWarning: 'Y' i
s deprecated and will be removed in a future version, please use 'YE' instead.
  yearly_forecast_prophet = daily_forecast_prophet.resample('Y').sum()
```

```
   timestamp  transaction_id
0 2025-01-01       29.375086
1 2025-01-02       36.610215
2 2025-01-03       35.863408
3 2025-01-04       31.873960
4 2025-01-05       31.041969
Files saved to their respective destinations
```

In [6]:

```
import pandas as pd
import matplotlib.pyplot as plt

def plot_raw_data(data_type, file_paths):
```

```
    """
    Plots raw data based on the specified data type from the given file paths.
    Additionally plots weekday vs. weekend data if 'weekday_vs_weekend' is specified.

    Parameters:
    - data_type (str): Type of data to plot ('daily', 'weekly', 'monthly', 'yearly', 'wee
kend', 'weekday', 'weekday_vs_weekend').
    - file_paths (dict): Dictionary containing file paths for each data type.
    """
    if data_type == 'weekday_vs_weekend':
        # Load Weekend Dataset
        weekend_file_path = file_paths['weekend']
        weekend_df = pd.read_csv(weekend_file_path)
        weekend_df['timestamp'] = pd.to_datetime(weekend_df['timestamp'])
        weekend_df.set_index('timestamp', inplace=True)

        # Load Weekday Dataset
        weekday_file_path = file_paths['weekday']
        weekday_df = pd.read_csv(weekday_file_path)
        weekday_df['timestamp'] = pd.to_datetime(weekday_df['timestamp'])
        weekday_df.set_index('timestamp', inplace=True)

        # Print the raw data to check its structure
        print("Raw weekend data:")
        print(weekend_df.head())
        print("Raw weekday data:")
        print(weekday_df.head())

        # Plotting Weekday vs Weekend Footfall Over Time
        plt.figure(figsize=(12, 6))
        plt.plot(weekend_df.index, weekend_df['transaction_id'], label='Weekend Footfall
', color='purple')
        plt.plot(weekday_df.index, weekday_df['transaction_id'], label='Weekday Footfall
', color='cyan')
        plt.title('Weekday vs Weekend Footfall Over Time')
        plt.xlabel('Timestamp')
        plt.ylabel('Transaction ID')
        plt.legend()
        plt.grid(True)
        plt.show()

        # Aggregating total transactions for weekday vs weekend
        total_weekend = weekend_df['transaction_id'].sum()
        total_weekday = weekday_df['transaction_id'].sum()

        # Creating a DataFrame for aggregation
        aggregated_df = pd.DataFrame({
            'Footfall Type': ['Weekday', 'Weekend'],
            'Total Transactions': [total_weekday, total_weekend]
        })

        # Plotting Aggregated Weekday vs Weekend Data
        plt.figure(figsize=(8, 6))
        plt.bar(aggregated_df['Footfall Type'], aggregated_df['Total Transactions'], colo
r=['cyan', 'purple'])
        plt.title('Total Weekday vs Weekend Footfall')
        plt.xlabel('Footfall Type')
        plt.ylabel('Total Transactions')
        plt.grid(True)
        plt.show()

    elif data_type == 'monthly':
        # Load Daily Dataset
        daily_file_path = file_paths['daily']
        daily_df = pd.read_csv(daily_file_path)
        daily_df['timestamp'] = pd.to_datetime(daily_df['timestamp'])
        daily_df.set_index('timestamp', inplace=True)

        # Aggregate Data to Monthly Totals
        monthly_footfall = daily_df.resample('M').sum()  # Aggregating daily data into m
onthly totals
        monthly_footfall['month_name'] = monthly_footfall.index.month_name()
```

```
        monthly_footfall['year'] = monthly_footfall.index.year

        # Print the aggregated data
        print("Monthly Aggregated Data:")
        print(monthly_footfall.head())

        # Optionally, save the aggregated data to a new CSV file
        aggregated_file_path = r'C:\Users\admin\Desktop\airline\sensor-file-ridership\out
put of sensors\monthly_aggregated_data.csv'
        monthly_footfall.to_csv(aggregated_file_path)
        print(f"Aggregated data saved to {aggregated_file_path}")

        # Plotting the aggregated monthly data
        plt.figure(figsize=(12, 6))
        monthly_footfall.plot(y='transaction_id', kind='bar', color='orange', legend=Fal
se)
        plt.title('Monthly Footfall Aggregated from Daily Data')
        plt.xlabel('Month')
        plt.ylabel('Transaction ID')
        plt.xticks(ticks=range(len(monthly_footfall)), labels=monthly_footfall['month_na
me'], rotation=45)
        plt.grid(True)
        plt.show()

    elif data_type == 'yearly':
        # Load Yearly Dataset
        yearly_file_path = file_paths['yearly']
        yearly_df = pd.read_csv(yearly_file_path)
        yearly_df['timestamp'] = pd.to_datetime(yearly_df['timestamp'])
        yearly_df.set_index('timestamp', inplace=True)
        yearly_df['year'] = yearly_df.index.year

        # Print the raw data to check its structure
        print("Raw yearly data:")
        print(yearly_df.head())

        # Plotting the yearly data
        plt.figure(figsize=(12, 6))
        yearly_df.groupby('year').sum().plot(y='transaction_id', kind='bar', color='blue
', legend=False)
        plt.title('Yearly Footfall')
        plt.xlabel('Year')
        plt.ylabel('Transaction ID')
        plt.grid(True)
        plt.show()

    elif data_type == 'weekly':
        # Load Weekly Dataset
        weekly_file_path = file_paths['weekly']
        weekly_df = pd.read_csv(weekly_file_path)
        weekly_df['timestamp'] = pd.to_datetime(weekly_df['timestamp'])
        weekly_df.set_index('timestamp', inplace=True)

        # Compute Week of Month
        weekly_df['week_of_month'] = weekly_df.index.to_series().apply(lambda x: (x.day
- 1) // 7 + 1)

        # Filter to show only the first 4 weeks of the month
        weekly_df = weekly_df[weekly_df['week_of_month'] <= 4]

        # Print the raw data to check its structure
        print("Raw weekly data:")
        print(weekly_df.head())

        # Plotting the weekly data
        plt.figure(figsize=(12, 6))
        weekly_df.groupby('week_of_month').sum().plot(y='transaction_id', kind='bar', co
lor='blue', legend=False)
        plt.title('Weekly Footfall Aggregated by Week of Month (First 4 Weeks)')
        plt.xlabel('Week of Month')
        plt.ylabel('Transaction ID')
        plt.grid(True)
```

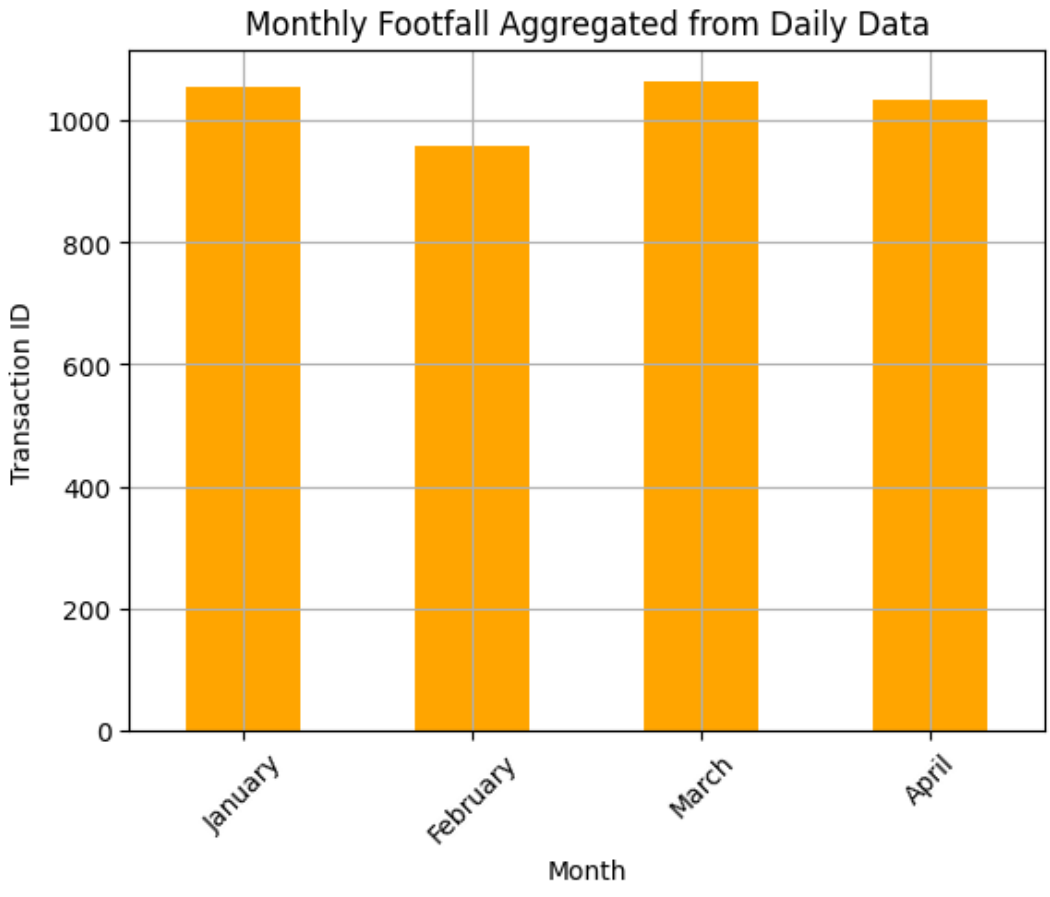
```
        plt.show()

    elif data_type == 'weekend':
        # Load Weekend Dataset
        weekend_file_path = file_paths['weekend']
        weekend_df = pd.read_csv(weekend_file_path)
        weekend_df['timestamp'] = pd.to_datetime(weekend_df['timestamp'])
        weekend_df.set_index('timestamp', inplace=True)
        weekend_df['day_of_week'] = weekend_df.index.day_name()

        # Print the raw data to check its structure
        print("Raw weekend data:")
        print(weekend_df.head())

        # Plotting the weekend data
        plt.figure(figsize=(12, 6))
        weekend_df.groupby('day_of_week').sum().plot(y='transaction_id', kind='bar', col
or='green', legend=False)
        plt.title('Weekend Footfall Aggregated by Day of Week')
        plt.xlabel('Day of Week')
        plt.ylabel('Transaction ID')
        plt.grid(True)
        plt.show()

    elif data_type == 'weekday':
        # Load Weekday Dataset
        weekday_file_path = file_paths['weekday']
        weekday_df = pd.read_csv(weekday_file_path)
        weekday_df['timestamp'] = pd.to_datetime(weekday_df['timestamp'])
        weekday_df.set_index('timestamp', inplace=True)
        weekday_df['day_of_week'] = weekday_df.index.day_name()

        # Print the raw data to check its structure
        print("Raw weekday data:")
        print(weekday_df.head())

        # Plotting the weekday data
        plt.figure(figsize=(12, 6))
        weekday_df.groupby('day_of_week').sum().plot(y='transaction_id', kind='bar', col
or='red', legend=False)
        plt.title('Weekday Footfall Aggregated by Day of Week')
        plt.xlabel('Day of Week')
        plt.ylabel('Transaction ID')
        plt.grid(True)
        plt.show()

    else:
        # Load Dataset
        file_path = file_paths.get(data_type)
        if file_path:
            df = pd.read_csv(file_path)

            # Data Preprocessing
            df['timestamp'] = pd.to_datetime(df['timestamp'])
            df.set_index('timestamp', inplace=True)

            # Print the raw data to check its structure
            print(f"Raw data for {data_type}:")
            print(df.head())

            # Plotting the raw data
            plt.figure(figsize=(12, 6))
            plt.plot(df.index, df['transaction_id'], label=f'Raw {data_type.capitalize()
} Footfall', color='blue')
            plt.title(f'Raw {data_type.capitalize()} Footfall Over Time')
            plt.xlabel('Timestamp')
            plt.ylabel('Transaction ID')
            plt.legend()
            plt.grid(True)
            plt.show()
        else:
            print(f"File path for '{data_type}' data type is not provided.")
```

```
# Define the base path for output files
base_path = r'C:\Users\admin\Desktop\airline\sensor-file-ridership\output of sensors'

# Define file paths for different forecast types
file_paths = {
    'daily': os.path.join(base_path, f'daily_footfall_forecast_prophet_{stn_no}_{eqn_no}.
csv'),
    'weekly': os.path.join(base_path, f'weekly_forecast_prophet_{stn_no}_{eqn_no}.csv'),
    'monthly': os.path.join(base_path, f'monthly_forecast_prophet_{stn_no}_{eqn_no}.csv')
,
    'yearly': os.path.join(base_path, f'yearly_forecast_prophet_{stn_no}_{eqn_no}.csv'),
    'weekend': os.path.join(base_path, f'weekend_forecast_prophet_{stn_no}_{eqn_no}.csv')
,
    'weekday': os.path.join(base_path, f'weekday_forecast_prophet_{stn_no}_{eqn_no}.csv')
}

data_type = 'monthly'  # Replace with desired data type ('daily', 'weekly', 'monthly', 'y
early', 'weekday_vs_weekend' , 'weekend' , 'weekday')
plot_raw_data(data_type, file_paths)
```

```
Monthly Aggregated Data:
            transaction_id month_name  year
timestamp
2025-01-31     1055.345813    January  2025
2025-02-28      957.909472   February  2025
2025-03-31     1063.087105      March  2025
2025-04-30     1034.528838      April  2025
Aggregated data saved to C:\Users\admin\Desktop\airline\sensor-file-ridership\output of s
ensors\monthly_aggregated_data.csv
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_13212\3323614150.py:70: FutureWarning: 'M' is
deprecated and will be removed in a future version, please use 'ME' instead.
  monthly_footfall = daily_df.resample('M').sum()  # Aggregating daily data into monthly
totals
```

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



In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```