BIG DATA ANALYTICS & DATA VISUALIZATION

By

Dhyey Bhandari

Problem Statement

Urban ride-hailing services like **Uber and Lyft** generate massive volumes of trip data every day across complex metropolitan areas. However, these companies often face challenges in effectively managing and analyzing this data to ensure operational efficiency and customer satisfaction. One of the primary concerns is the **inefficient allocation of drivers**, especially during **peak hours** or in **underserved zones**, which leads to longer **wait times, poor service quality, and loss of potential revenue.**

Another major issue is the **unpredictability of trip durations and fares**, which can be influenced by **traffic patterns, weather conditions, and route selection.** Without accurate predictive insights, both customers and drivers experience uncertainty, which negatively affects trust and platform reliability. In addition, companies encounter difficulty in **detecting fare anomalies**, such as unjustified surcharges or inconsistencies in pricing across similar trips.

From a business intelligence perspective, there is a growing need to **analyze trip volumes, revenue trends, and geographic patterns.** Identifying busiest hours, days, and months, as well as top pickup and dropoff locations, is crucial for strategic planning and resource deployment. However, manually deriving these insights from raw trip data is both time-consuming and error-prone.

In summary, the lack of integrated, **data-driven solutions** for these multifaceted problems hampers decision-making, efficiency, and long-term strategic planning in the ride-hailing industry. This project aims to address these issues by transforming raw data into actionable insights through analysis and predictive modeling.

Big Data Visualization

Overview of the "NYC For-Hire Vehicles Trip" dataset

Introduction:

The big dataset contains detailed **5.4 GB** of **trip-level records** for **For-Hire Vehicle (FHV) services** in New York City, comprising millions of entries collected over a defined period. It captures essential operational and financial information related to individual FHV rides, including identifiers for the dispatching company (such as Uber or Lyft), timestamps for pickup and dropoff, and pickup/dropoff location IDs.

Additional features include trip distance, trip duration, base passenger fare, tolls, surcharges (Black Car Fund, congestion, and airport fees), taxes, tips, and the final amount paid to the driver. Analyzing this data provides valuable insights into travel patterns, fare structures, and driver earnings, supporting optimization in urban mobility planning, policy-making, and FHV service management.

Source:

The big dataset is provided by https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Dataset Structure:

The NYC FHV trip dataset is stored in **Parquet format** and comprises a total of **239,448,588 records (rows)** and **15 columns (features**). Each row represents a single for-hire vehicle trip recorded in New York City.

Data Headers:

Column Name	Data Type	Description
dispatching_base_num	string	Identifies the FHV base/company (e.g., Uber, Lyft) that dispatched the vehicle.
pickup_datetime	timestamp	The date and time when the trip began.
dropoff_datetime	timestamp	The date and time when the trip ended.
pulocationid	int	TLC Taxi Zone ID for the pickup location.
dolocationid	int	TLC Taxi Zone ID for the dropoff location.
trip_miles	float	Distance of the trip in miles.
trip_time	int	Duration of the trip in seconds.
base_passenger_fare	float	The base fare paid by the passenger, excluding taxes and surcharges.
tolls	float	Toll charges incurred during the trip.
bcf	float	Black Car Fund surcharge, a mandatory fee for FHV rides in NYC.
sales_tax	float	Sales tax applicable to the trip.
congestion_surcharge	float	Additional charge for trips in high-traffic zones during peak hours.
airport_fee	float	Fixed fee applied to trips to/from NYC airports.
tips	float	Tip amount paid by the customer.
driver_pay	float	Total amount paid to the driver for the trip, including base fare and tips.

AWS aspects that are explored:

1. AWS S3: Creating & uploading 5.4 GB of big data into the S3 bucket created.

2. AWS Athena: To run SQL queries and process data to gain valuable insights.

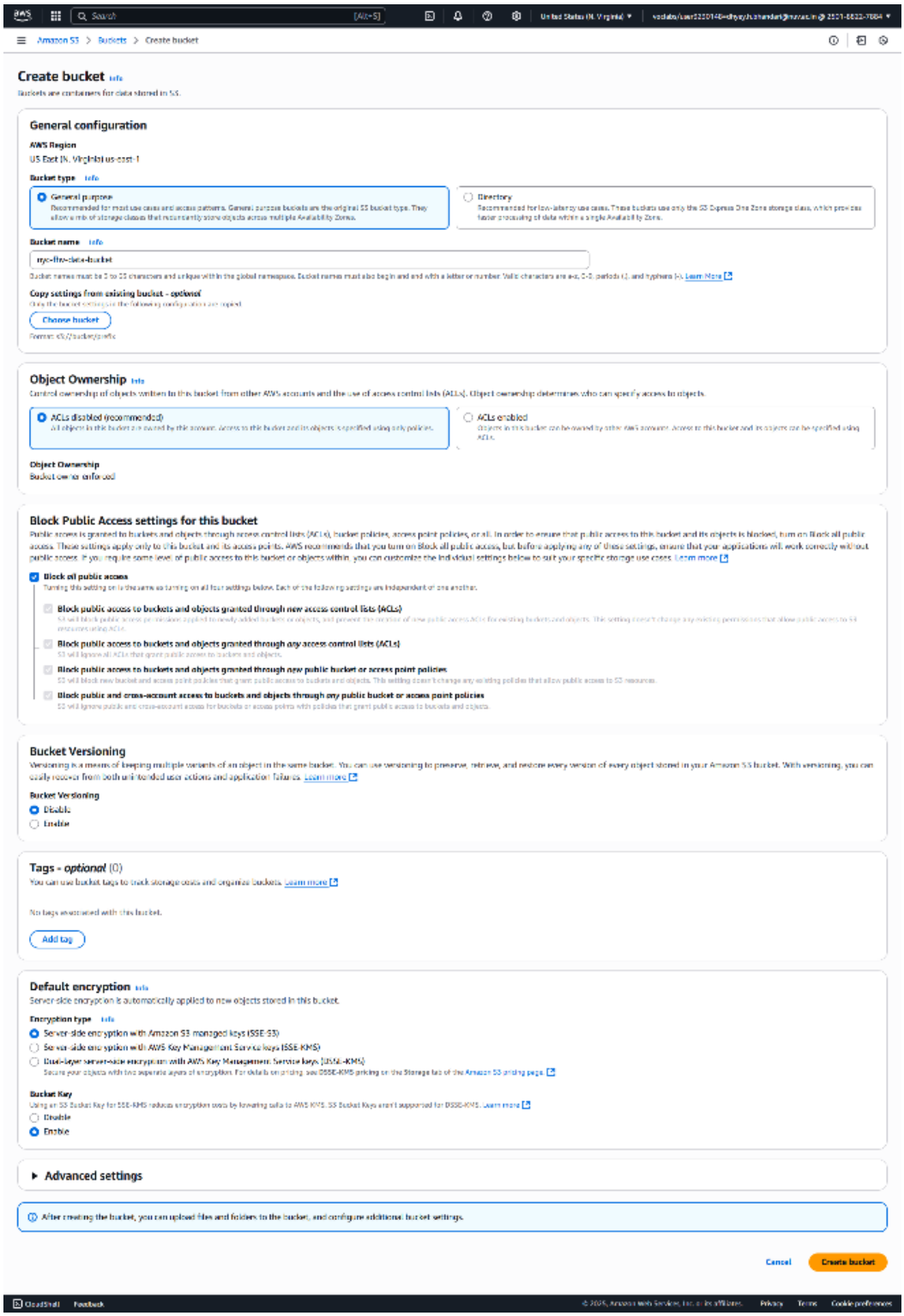
3. AWS Cloud9: Creating an environment for Jupyter Notebook to visualize data on aggregated output given by Athena queries.

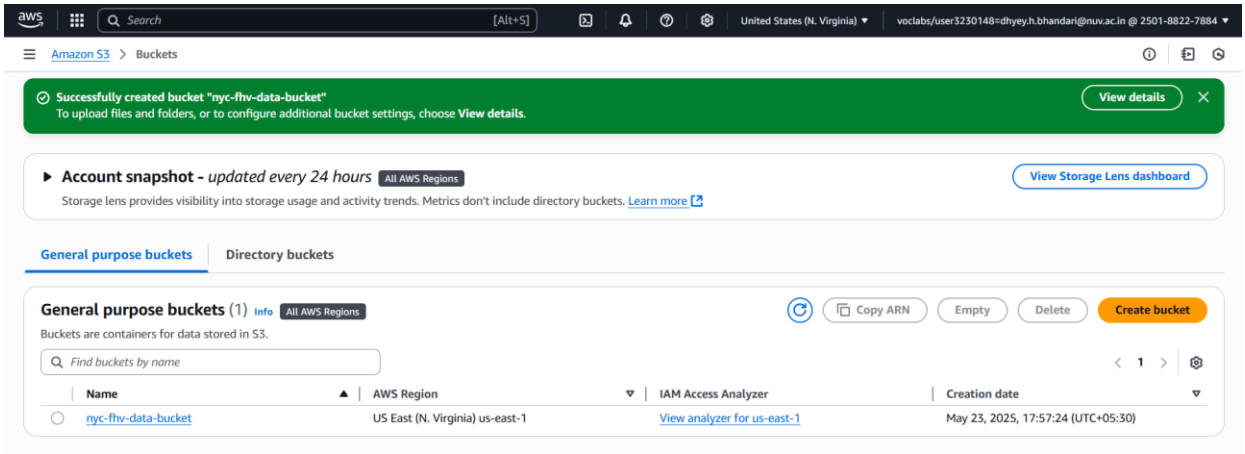
Steps Followed:

1. Creating S3 bucket:

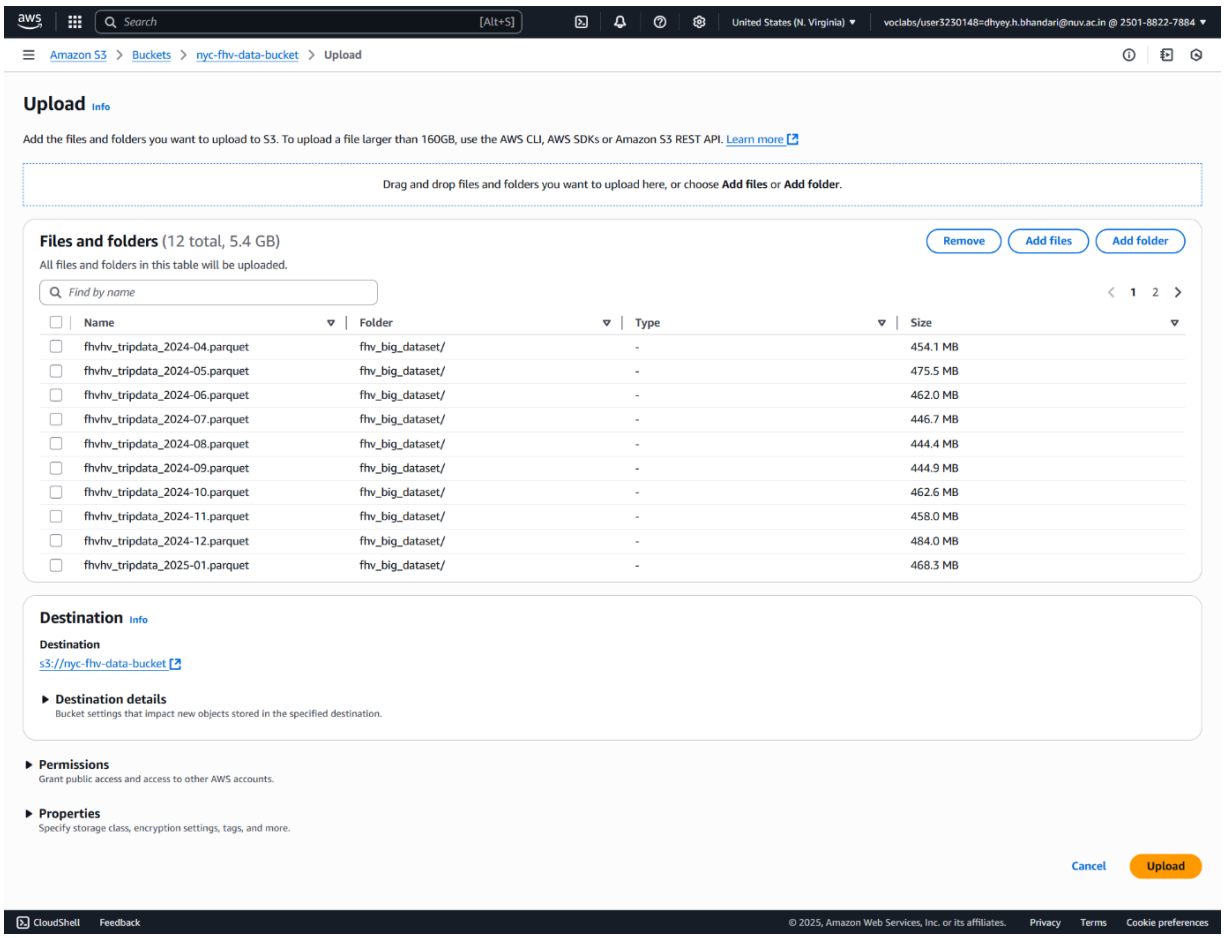
Bucket Name: nyc-fhv-data-bucket

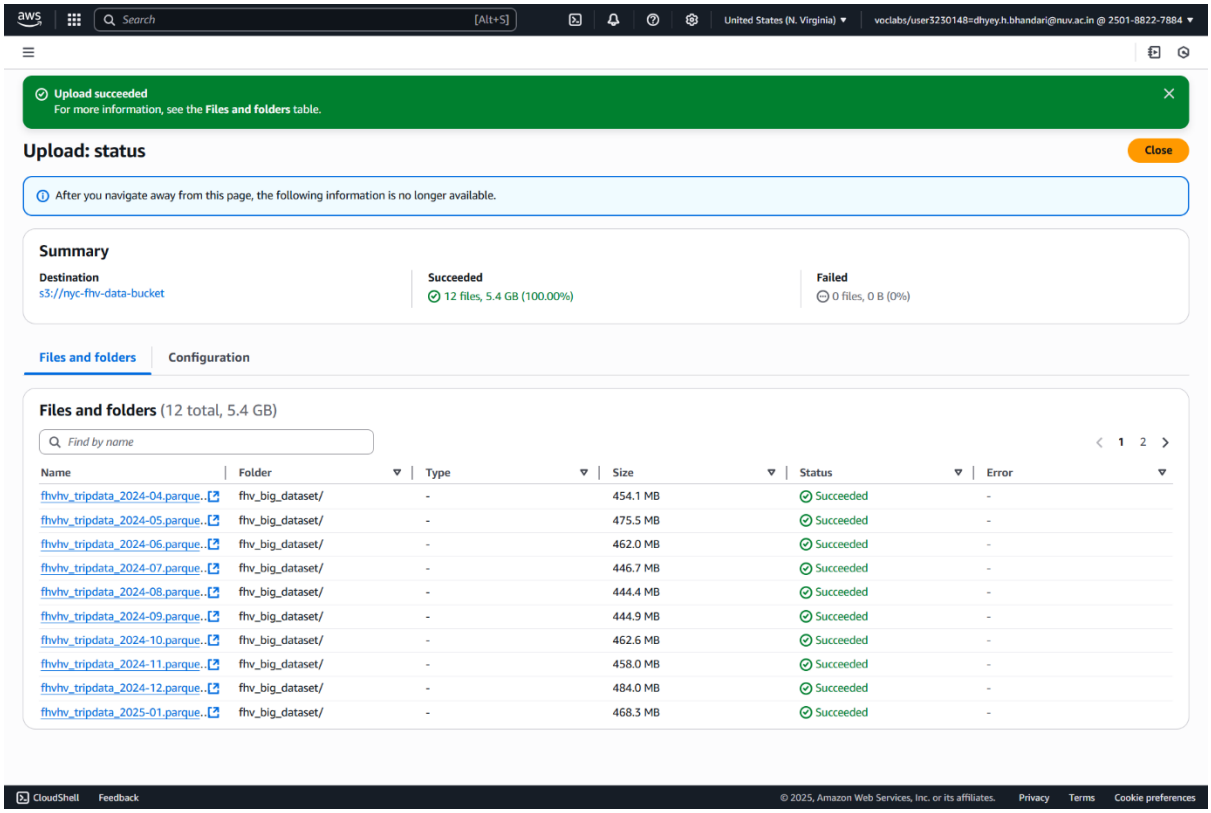
Region: us-east-1





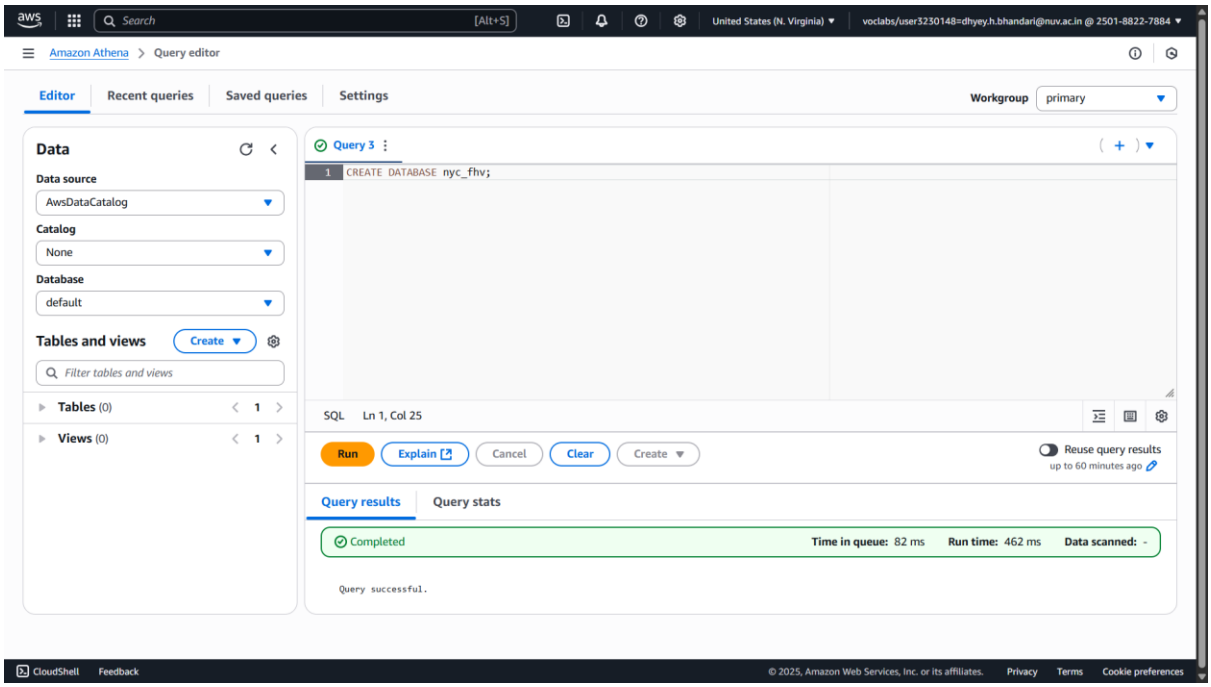
2. Uploading Data: 12 parquet files, 5.4 GB





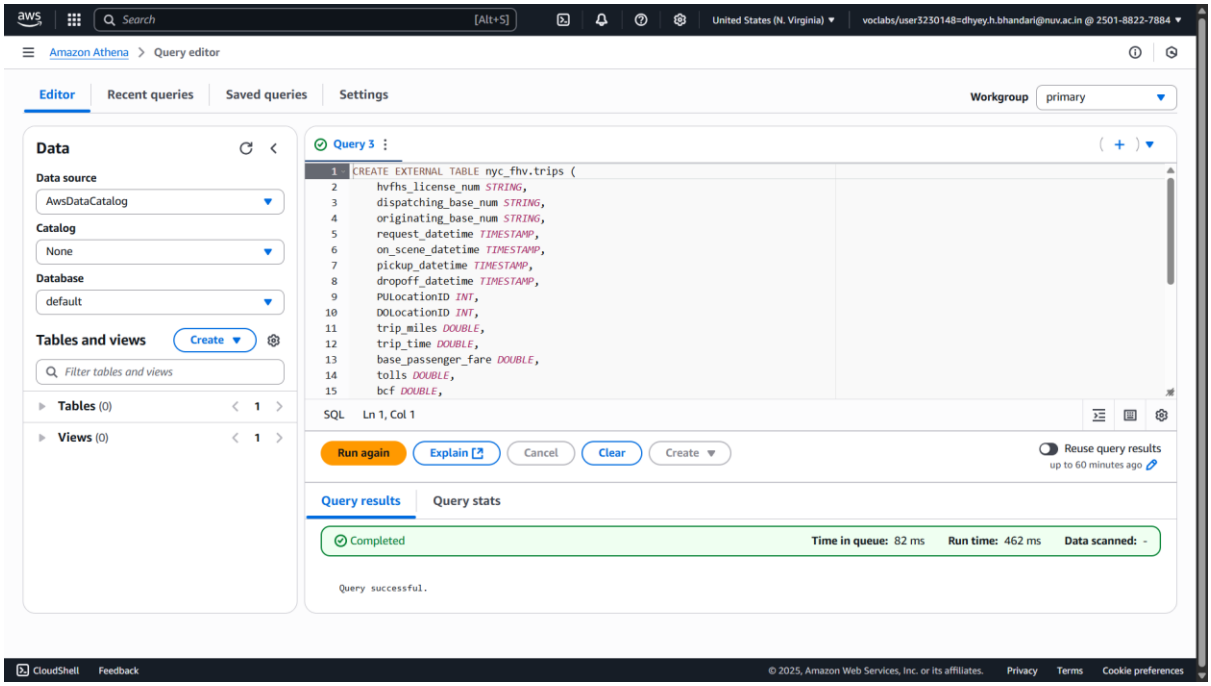
3. Database Creation:

Database Name: nyc_fhv



4. Table Creation:

Table Name: trips



5. Running SQL Queries:

Hourly Trips:

```
SELECT
  hour(pickup_datetime) AS trip_hour,
  COUNT(*) AS total_trips
FROM nyc_fhv.trips
GROUP BY hour(pickup_datetime)
ORDER BY trip_hour;
```

Daily Trips:

```
SELECT
  date(pickup_datetime) AS trip_date,
  COUNT(*) AS total_trips
FROM nyc_fhv.trips
```

```
GROUP BY date(pickup_datetime)
ORDER BY trip_date;
```

Weekly Trips:

```
SELECT
  year(pickup_datetime) AS trip_year,
  week(pickup_datetime) AS trip_week,
  COUNT(*) AS total_trips
FROM nyc_fhv.trips
GROUP BY year(pickup_datetime), week(pickup_datetime)
ORDER BY trip_year, trip_week;
```

Monthly Trips:

```
SELECT
  year(pickup_datetime) AS trip_year,
  month(pickup_datetime) AS trip_month,
  COUNT(*) AS total_trips
FROM nyc_fhv.trips
GROUP BY year(pickup_datetime), month(pickup_datetime)
ORDER BY trip_year, trip_month;
```

Peak Hour of the Day:

```
SELECT
  hour(pickup_datetime) AS trip_hour,
  COUNT(*) AS trip_count
FROM nyc_fhv.trips
GROUP BY hour(pickup_datetime)
ORDER BY trip_count DESC
LIMIT 10;
```

Peak Day of the Week:

```
SELECT
  day_of_week(pickup_datetime) AS day_of_week,
  COUNT(*) AS trip_count
FROM nyc_fhv.trips
GROUP BY day_of_week(pickup_datetime)
ORDER BY trip_count DESC;
```

Peak Week:

```
SELECT
  year(pickup_datetime) AS trip_year,
  week(pickup_datetime) AS trip_week,
  COUNT(*) AS trip_count
FROM nyc_fhv.trips
GROUP BY year(pickup_datetime), week(pickup_datetime)
ORDER BY trip_count DESC
LIMIT 10;
```

Monthly Revenue Trend:

```
SELECT
  year(pickup_datetime) AS trip_year,
  month(pickup_datetime) AS trip_month,
  SUM(base_passenger_fare + tolls + bcf + sales_tax + congestion_surcharge + airport_fee
+ tips) AS total_revenue
FROM nyc_fhv.trips
GROUP BY year(pickup_datetime), month(pickup_datetime)
ORDER BY trip_year, trip_month;
```

Daily Average Fare:

```
SELECT
```

```
  date(pickup_datetime) AS trip_date,
  AVG(base_passenger_fare) AS avg_fare
FROM nyc_fhv.trips
GROUP BY date(pickup_datetime)
ORDER BY trip_date;
```

Most Popular Routes:

```
SELECT
  PULocationID,
  DOLocationID,
  COUNT(*) AS trip_count
FROM nyc_fhv.trips
GROUP BY PULocationID, DOLocationID
ORDER BY trip_count DESC
LIMIT 10;
```

Top Pickup Location:

```
SELECT
  PULocationID,
  COUNT(*) AS pickup_count
FROM nyc_fhv.trips
GROUP BY PULocationID
ORDER BY pickup_count DESC
LIMIT 10;
```

Top Dropoff Location:

```
SELECT
  DOLocationID,
  COUNT(*) AS dropoff_count
FROM nyc_fhv.trips
```

```
GROUP BY DOLocationID
ORDER BY dropoff_count DESC
LIMIT 10;
```

Trips by Company:

```
SELECT
  CASE
    WHEN hvfhs_license_num LIKE 'HV0003%' THEN 'Uber'
    WHEN hvfhs_license_num LIKE 'HV0005%' THEN 'Lyft'
    ELSE 'Other'
  END AS company,
  COUNT(*) AS total_trips
FROM nyc_fhv.trips
GROUP BY
  CASE
    WHEN hvfhs_license_num LIKE 'HV0003%' THEN 'Uber'
    WHEN hvfhs_license_num LIKE 'HV0005%' THEN 'Lyft'
    ELSE 'Other'
  END;
```

Revenue by Company:

```
SELECT
  CASE
    WHEN hvfhs_license_num LIKE 'HV0003%' THEN 'Uber'
    WHEN hvfhs_license_num LIKE 'HV0005%' THEN 'Lyft'
    ELSE 'Other'
  END AS company,
  SUM(base_passenger_fare + tolls + bcf + sales_tax + congestion_surcharge + airport_fee
+ tips) AS total_revenue
FROM nyc_fhv.trips
```

```
GROUP BY
  CASE
    WHEN hvfhs_license_num LIKE 'HV0003%' THEN 'Uber'
    WHEN hvfhs_license_num LIKE 'HV0005%' THEN 'Lyft'
    ELSE 'Other'
  END;
```

Shared vs Solo Rides:

```
SELECT
  shared_request_flag,
  COUNT(*) AS trip_count
FROM nyc_fhv.trips
GROUP BY shared_request_flag;
```

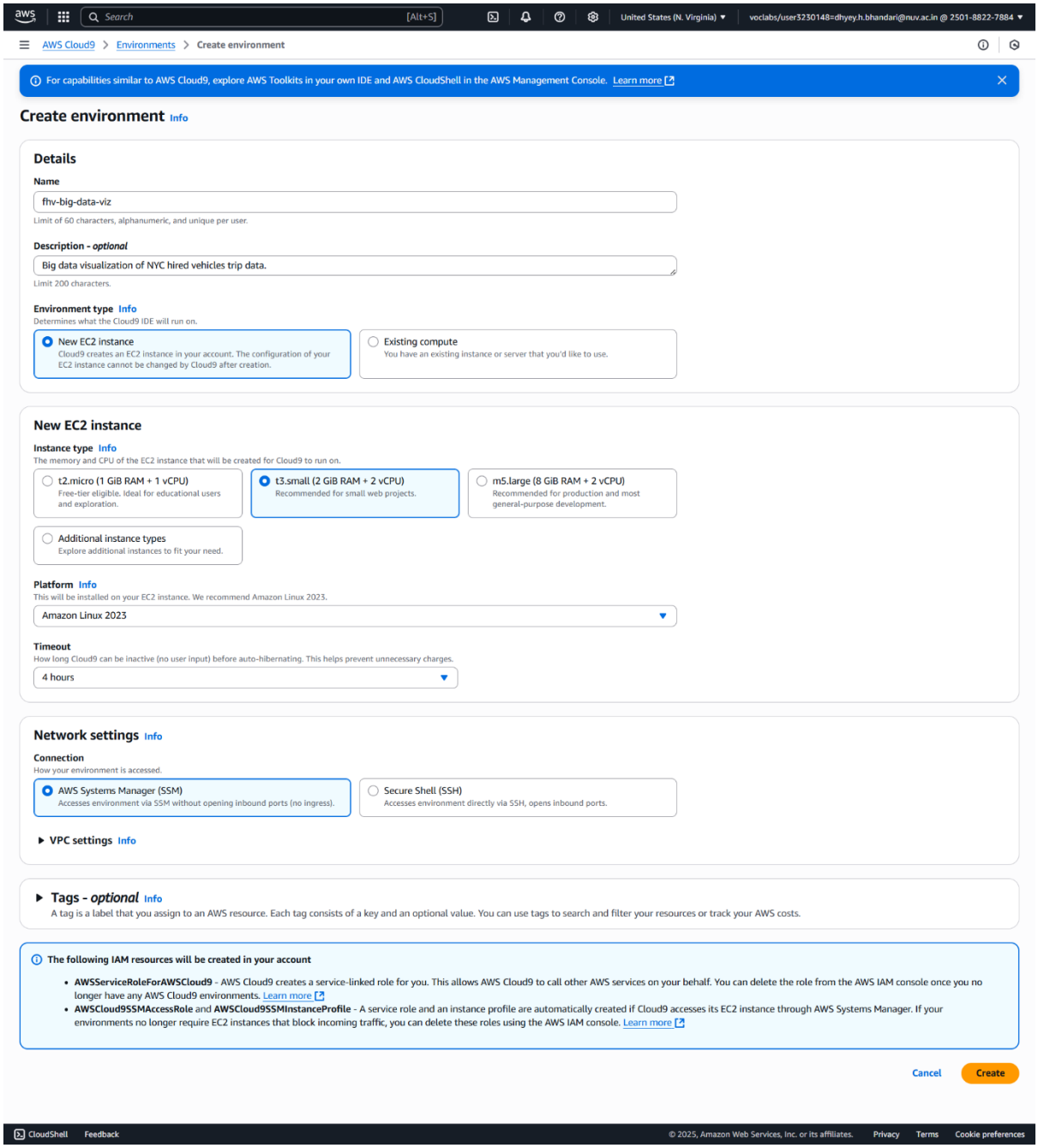
WAV (Wheelchair Accessible Vehicle) Requests:

```
SELECT
  wav_request_flag,
  COUNT(*) AS trip_count
FROM nyc_fhv.trips
GROUP BY wav_request_flag;
```

6. Creating Cloud9 environment:

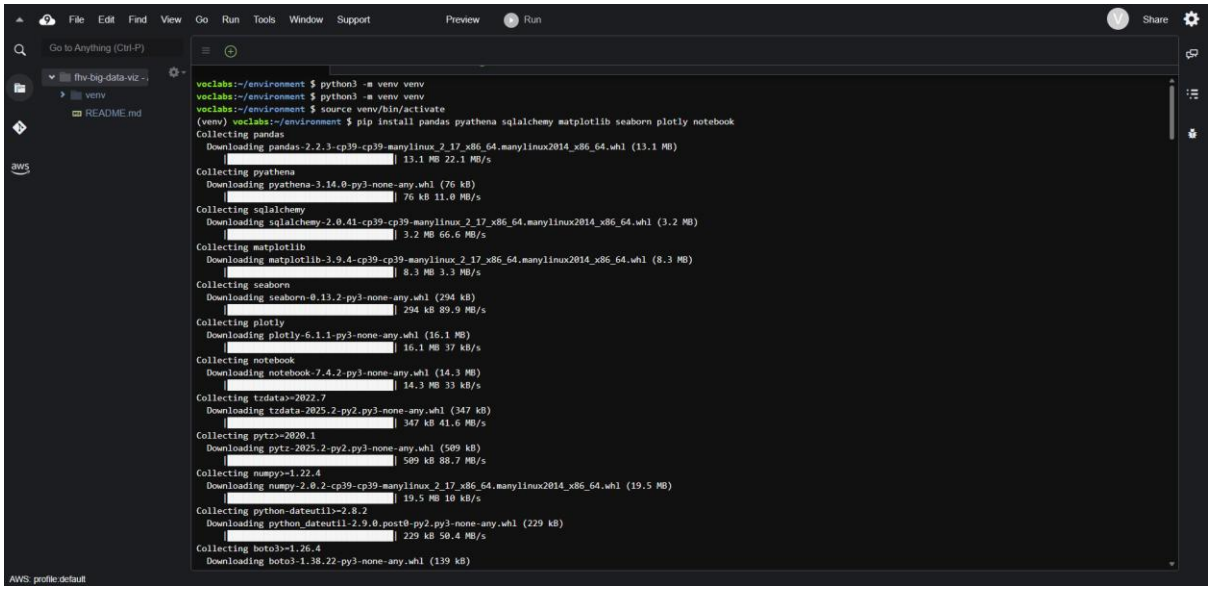
Environment Name: fhv-big-data-viz

EC2 Instance Type: t3.small (2 GiB RAM + 2 vCPU)

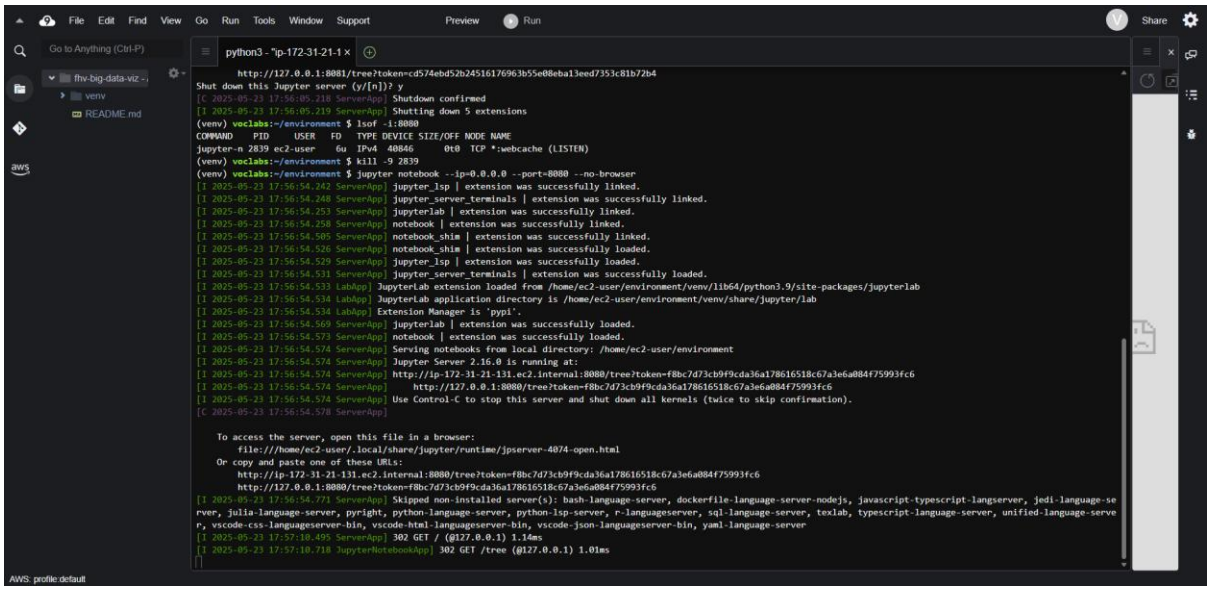


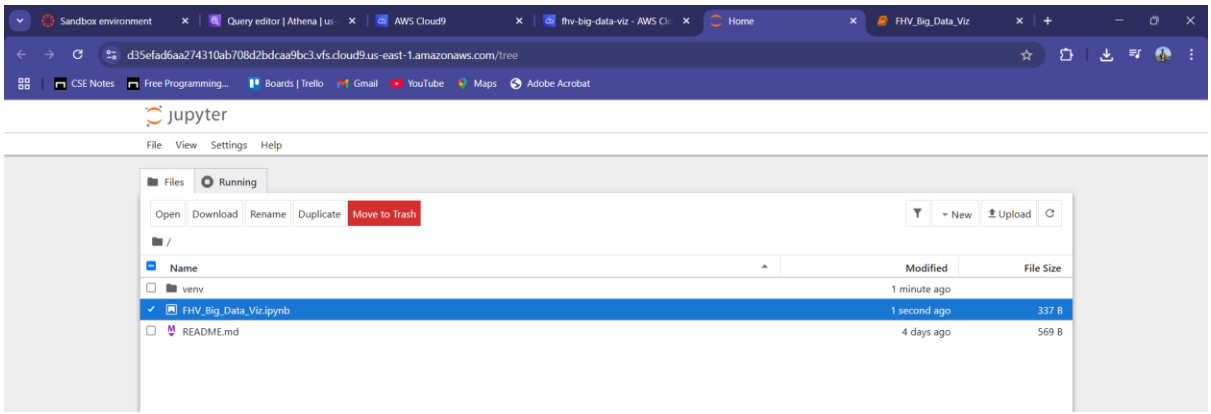
7. Installing libraries in Cloud9 terminal:

Python Data Processing & Visualization Libraries: pandas, pyathena, sqlalchemy, matplotlib, seaborn, plotly, notebook



8. Initiating server for Jupyter Notebook:

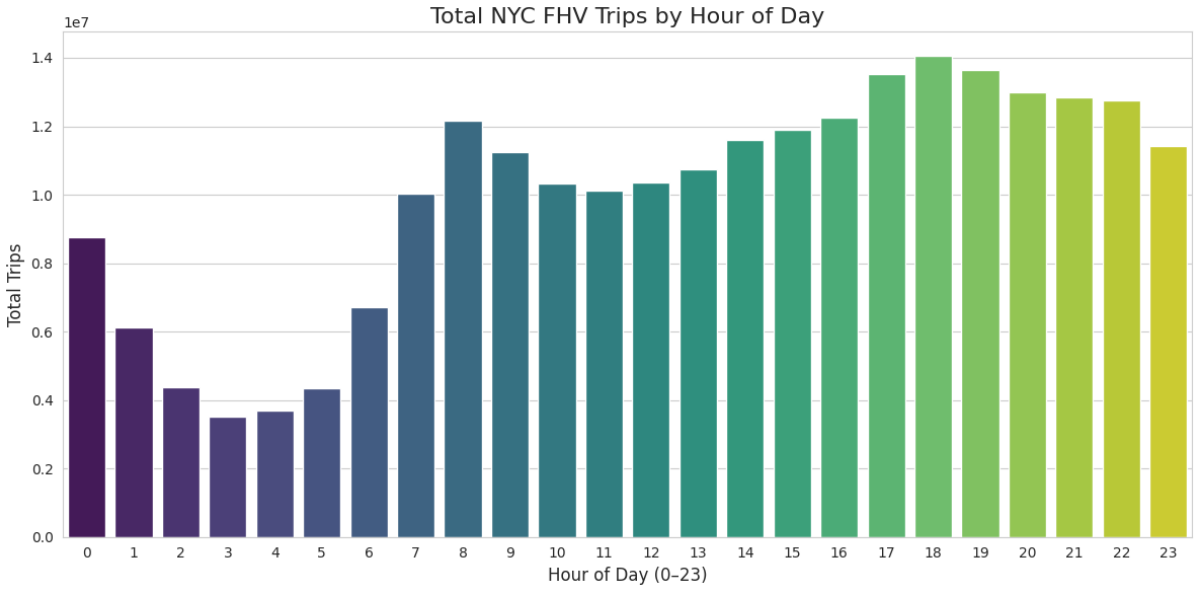




9. Python Codes for Visualization:

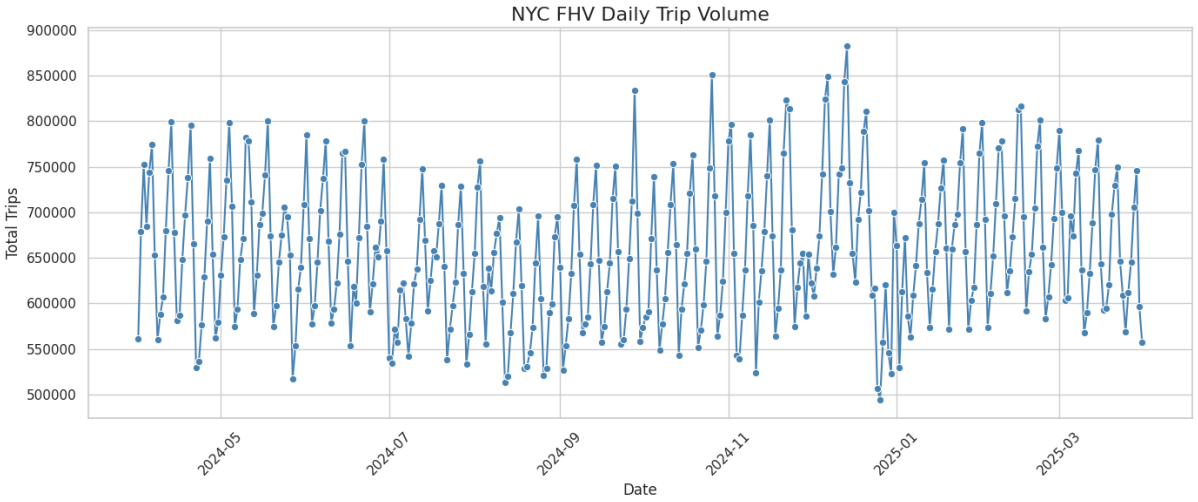
Hourly Trips:

[3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV (adjust filename if needed)
df = pd.read_csv('fc1dcda3-f9e4-4264-bf28-161b8fa90541.csv')  # or your actual file name

# Optional: ensure 'trip_hour' is treated as categorical (for correct ordering in plots)
df['trip_hour'] = df['trip_hour'].astype(int)

# Set style
sns.set_style("whitegrid")

# Create bar plot
plt.figure(figsize=(12,6))
sns.barplot(x='trip_hour', y='total_trips', data=df, palette='viridis')

# Add titles and labels
plt.title('Total NYC FHV Trips by Hour of Day', fontsize=16)
plt.xlabel('Hour of Day (0-23)', fontsize=12)
plt.ylabel('Total Trips', fontsize=12)
plt.xticks(rotation=0)
plt.tight_layout()

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



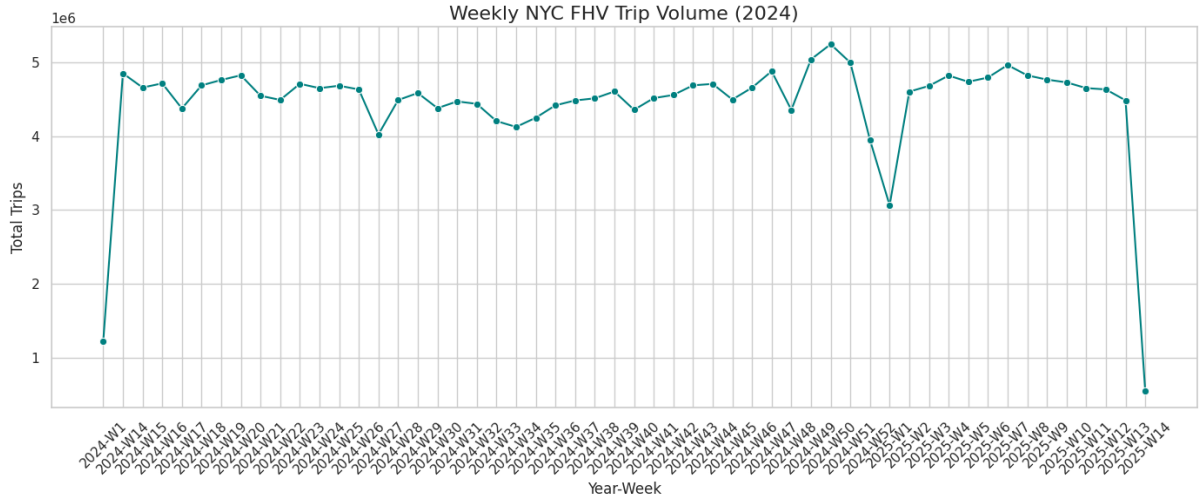
Daily Trips:

[6]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV
df = pd.read_csv('5ab0fcba-6432-4e95-a7b2-2567799eb7f0.csv', parse_dates=['trip_date'])  # Adjust filename if needed

# Preview data
print(df.head())

# Set Seaborn style
sns.set(style='whitegrid')

# Create Line plot
plt.figure(figsize=(14,6))
sns.lineplot(x='trip_date', y='total_trips', data=df, marker='o', color='steelblue')

# Add titles and labels
plt.title('NYC FHV Daily Trip Volume', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Total Trips', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()

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



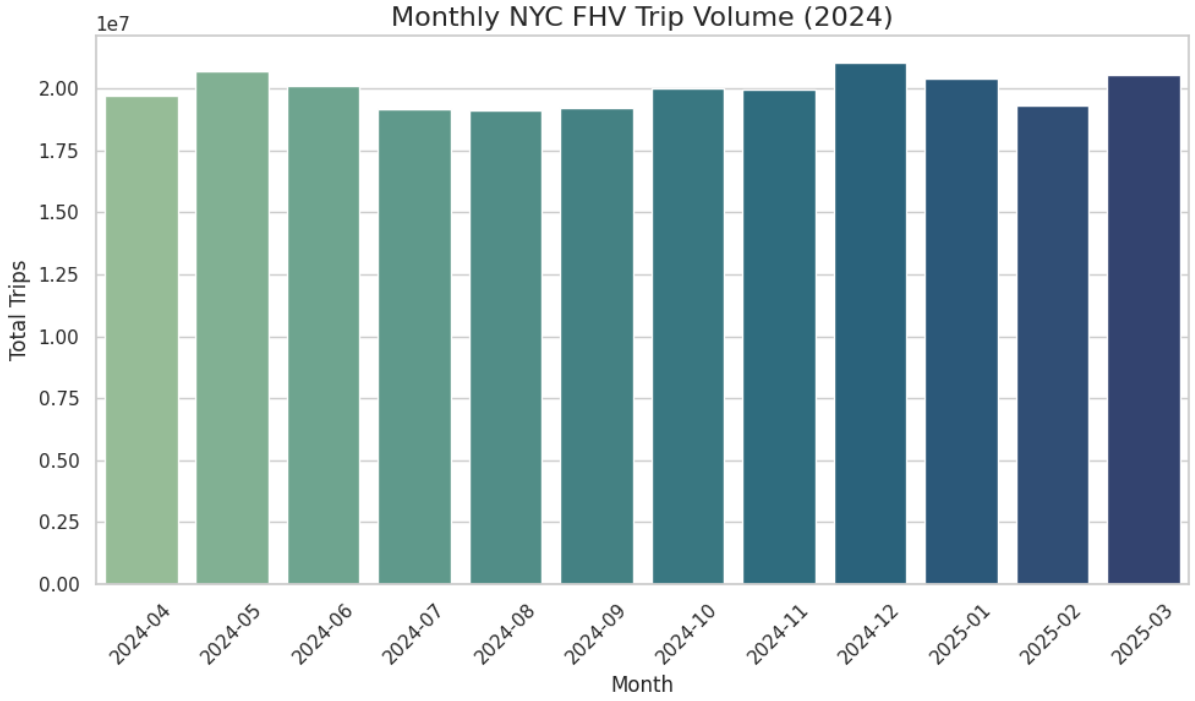
Weekly Trips:

[8]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV
df = pd.read_csv('1fef1b7a-812e-4013-b0c3-07cb09b63dfc.csv')  # Adjust the filename

# Combine year and week into a single string for x-axis
df['year_week'] = df['trip_year'].astype(str) + '-W' + df['trip_week'].astype(str)

# Optional: convert to datetime (week start date), for time-based x-axis (advanced)
# df['week_start'] = pd.to_datetime(df['trip_year'].astype(str) + df['trip_week'].astype(str) + '1', format='%G%V%u')

# Set Seaborn style
sns.set(style='whitegrid')

# Plot
plt.figure(figsize=(14,6))
sns.lineplot(x='year_week', y='total_trips', data=df, marker='o', color='teal')

# Add labels
plt.title('Weekly NYC FHV Trip Volume (2024)', fontsize=16)
plt.xlabel('Year-Week', fontsize=12)
plt.ylabel('Total Trips', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```



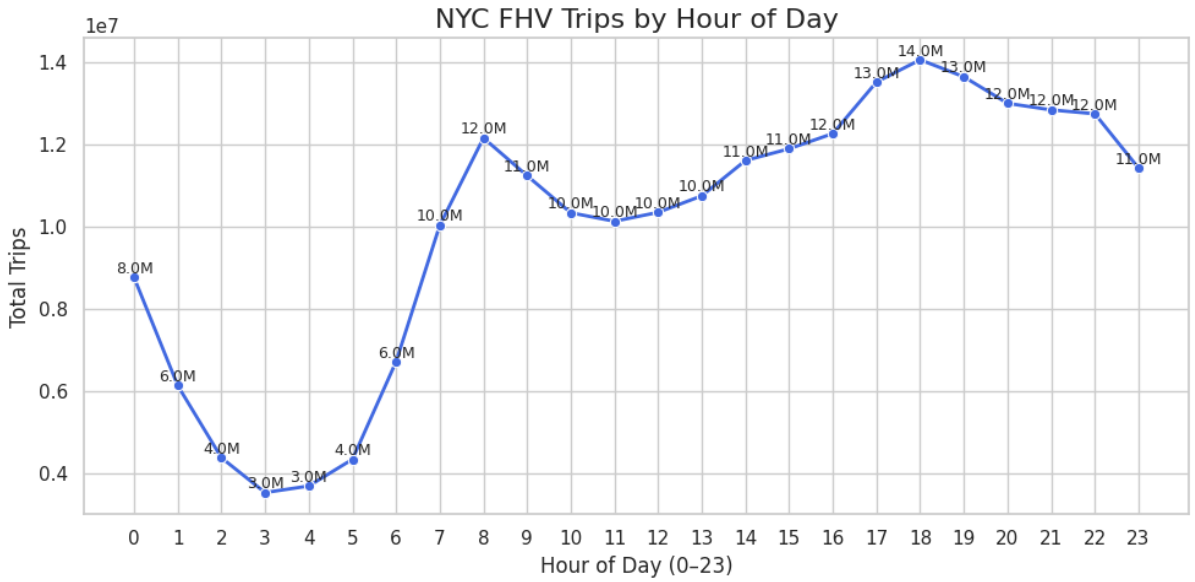
Monthly Trips:

[11]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV
df = pd.read_csv('25cd1537-1e20-481f-b166-fcf1017bc014.csv')  # Adjust filename if needed

# Create a column for "YYYY-MM" labels
df['year_month'] = df['trip_year'].astype(str) + '-' + df['trip_month'].astype(str).str.zfill(2)

# Set Seaborn style
sns.set(style='whitegrid')

# Plot bar chart
plt.figure(figsize=(10,6))
sns.barplot(x='year_month', y='total_trips', data=df, palette='crest')

# Add titles and labels
plt.title('Monthly NYC FHV Trip Volume (2024)', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Total Trips', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()

# Show plot
plt.show()
```



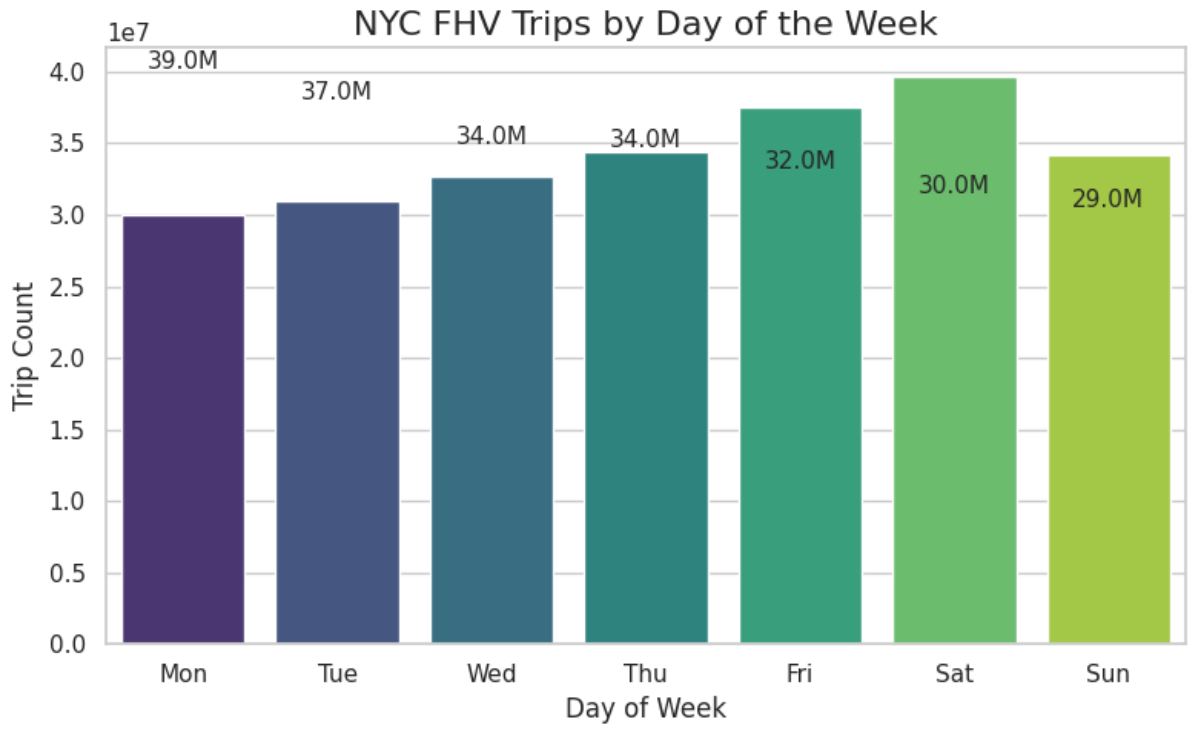
Peak Hour of the Day:

[31]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load data
df = pd.read_csv('fc1dcda3-f9e4-4264-bf28-161b8fa90541.csv')  # Adjust filename if needed

# Set style
sns.set(style='whitegrid')

# Plot line chart
plt.figure(figsize=(10,5))
sns.lineplot(x='trip_hour', y='total_trips', data=df, marker='o', linewidth=2, color='royalblue')

# Annotate each point with total trips (optional)
for i, row in df.iterrows():
    plt.text(row['trip_hour'], row['total_trips'] + 100000, f"{row['total_trips'] // 1_000_000:.1f}M",
             ha='center', fontsize=9)

# Set axis labels and title
plt.title('NYC FHV Trips by Hour of Day', fontsize=16)
plt.xlabel('Hour of Day (0–23)', fontsize=12)
plt.ylabel('Total Trips', fontsize=12)
plt.xticks(range(0, 24))
plt.tight_layout()

# Show plot
plt.show()
```



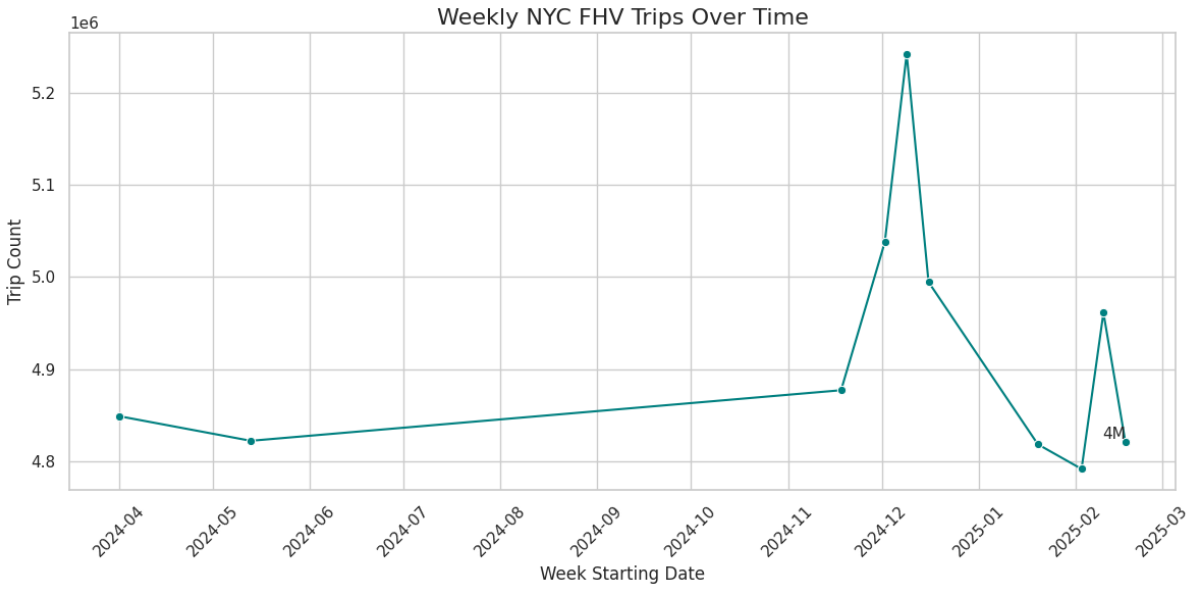
Peak Day of the Week:

[33]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load CSV
df = pd.read_csv('5066ad66-6c9a-4a9f-ac51-c88b54d39a29.csv')  # Adjust filename

# Map day numbers to names
day_map = {
    1: 'Mon', 2: 'Tue', 3: 'Wed', 4: 'Thu',
    5: 'Fri', 6: 'Sat', 7: 'Sun'
}
df['day_name'] = df['day_of_week'].map(day_map)

# Sort days in correct week order
day_order = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
df['day_name'] = pd.Categorical(df['day_name'], categories=day_order, ordered=True)
df = df.sort_values('day_name')

# Plot bar chart
plt.figure(figsize=(8,5))
sns.barplot(x='day_name', y='trip_count', data=df, palette='viridis')

# Add value labels
for i, row in df.iterrows():
    plt.text(i, row['trip_count'] + 5e5, f"{row['trip_count'] // 1_000_000:.1f}M",
             ha='center', fontsize=11)

# Labels and title
plt.title('NYC FHV Trips by Day of the Week', fontsize=16)
plt.xlabel('Day of Week', fontsize=12)
plt.ylabel('Trip Count', fontsize=12)
plt.tight_layout()
plt.show()
```



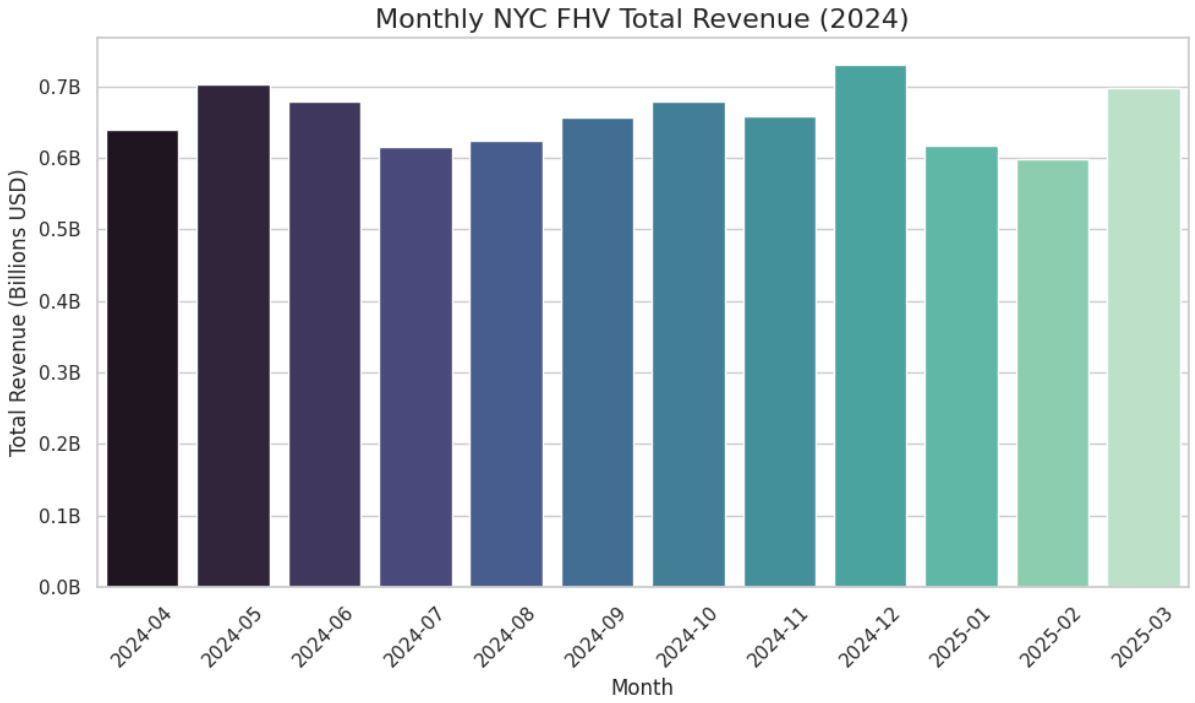
Peak Week:

[35]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load CSV
df = pd.read_csv('b02b3d90-6d48-4106-899d-68f1a9195aa9.csv')  # Adjust filename

# Create a datetime column from year and week number (ISO week)
# We'll use pandas to convert year-week to datetime (Monday of the week)
df['week_start_date'] = pd.to_datetime(df['trip_year'].astype(str) + df['trip_week'].astype(str) + '1', format='%G%V%u')

# Sort by date
df = df.sort_values('week_start_date')

# Set plot style
sns.set(style='whitegrid')

# Plot line chart
plt.figure(figsize=(12,6))
sns.lineplot(x='week_start_date', y='trip_count', data=df, marker='o', color='teal')

# Format x-axis for better date display
plt.xticks(rotation=45)
plt.xlabel('Week Starting Date', fontsize=12)
plt.ylabel('Trip Count', fontsize=12)
plt.title('Weekly NYC FHV Trips Over Time', fontsize=16)

# Optional: annotate last point
last_row = df.iloc[-1]
plt.text(last_row['week_start_date'], last_row['trip_count'], f"{last_row['trip_count'] // 1_000_000}M",
         fontsize=11, ha='right', va='bottom')

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



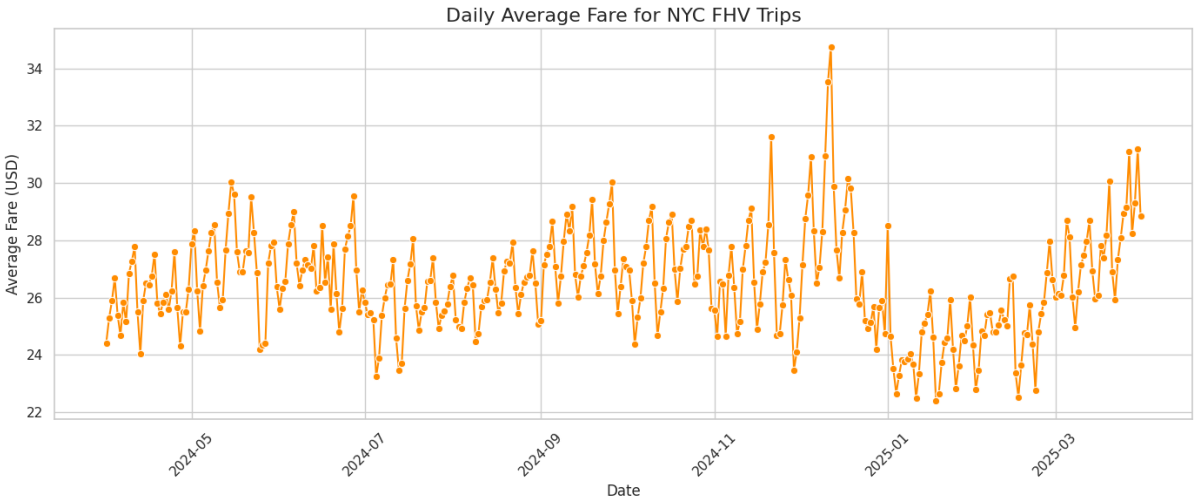
Monthly Revenue Trends:

[13]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as ticker

# Load CSV
df = pd.read_csv('a6d8a6a1-cf83-4ce4-80e2-0ce227b65b63.csv')  # Adjust filename

# Create a 'year_month' column like '2024-04'
df['year_month'] = df['trip_year'].astype(str) + '-' + df['trip_month'].astype(str).str.zfill(2)

# Set style
sns.set(style='whitegrid')

# Plot bar chart
plt.figure(figsize=(10,6))
ax = sns.barplot(x='year_month', y='total_revenue', data=df, palette='mako')

# Format y-axis labels as billions or millions
def billions(x, pos):
    return f'{x*1e-9:.1f}B'  # Convert to billions

ax.yaxis.set_major_formatter(ticker.FuncFormatter(billions))

# Titles and labels
plt.title('Monthly NYC FHV Total Revenue (2024)', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Total Revenue (Billions USD)', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



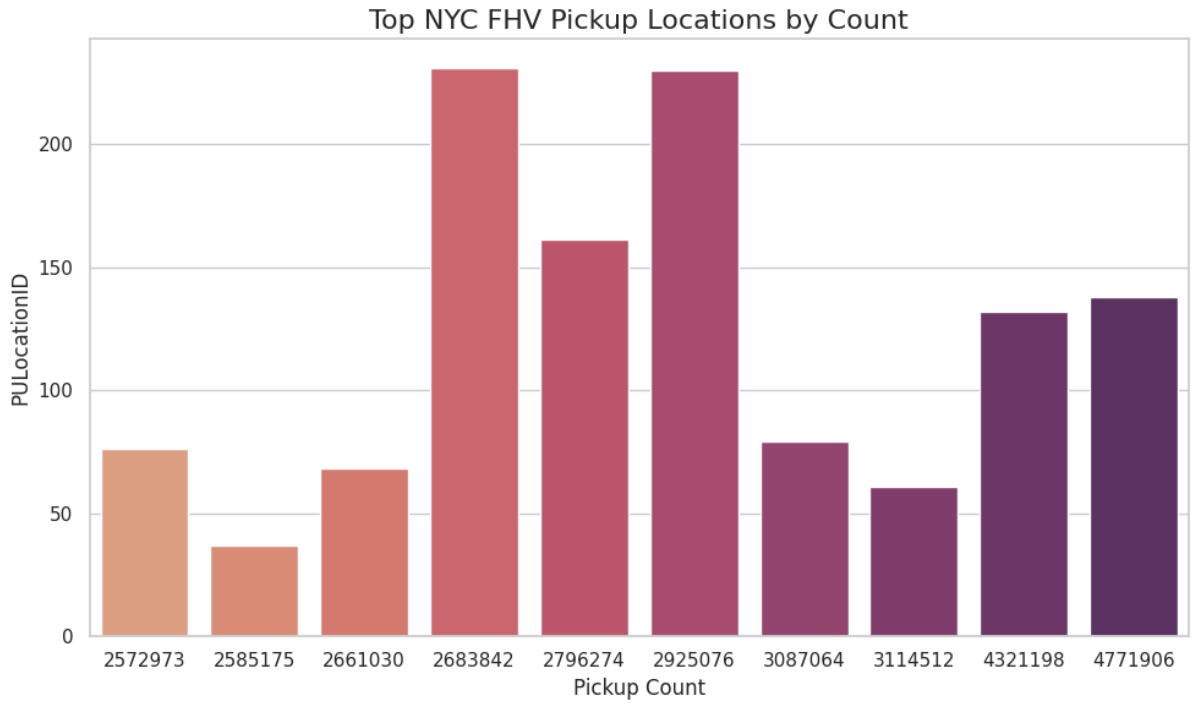
Daily Average Fare:

[15]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load CSV
df = pd.read_csv('f6f33939-b97e-4123-bef1-4965e7884b13.csv', parse_dates=['trip_date'])  # Adjust filename

# Set style
sns.set(style='whitegrid')

# Plot line chart
plt.figure(figsize=(14,6))
sns.lineplot(x='trip_date', y='avg_fare', data=df, marker='o', color='darkorange')

# Titles and labels
plt.title('Daily Average Fare for NYC FHV Trips', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Average Fare (USD)', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



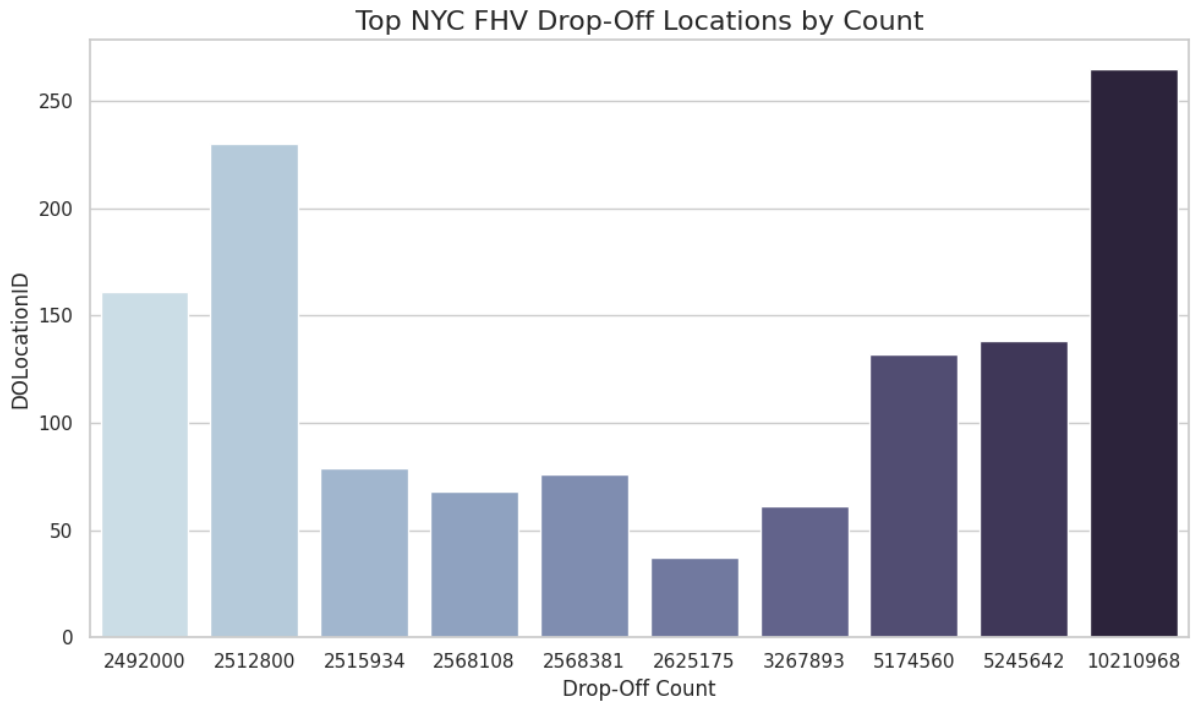
Top Pickup Locations:

[19]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV
df = pd.read_csv('ab8d6f10-b864-4d56-b548-e5997203da9f.csv')  # Adjust filename

# Optional: sort by pickup_count for cleaner bar chart
df = df.sort_values(by='pickup_count', ascending=True)

# Create Seaborn barplot (horizontal)
plt.figure(figsize=(10,6))
sns.barplot(x='pickup_count', y='PULocationID', data=df, palette='flare')

# Add labels and title
plt.title('Top NYC FHV Pickup Locations by Count', fontsize=16)
plt.xlabel('Pickup Count', fontsize=12)
plt.ylabel('PULocationID', fontsize=12)
plt.tight_layout()

plt.show()
```



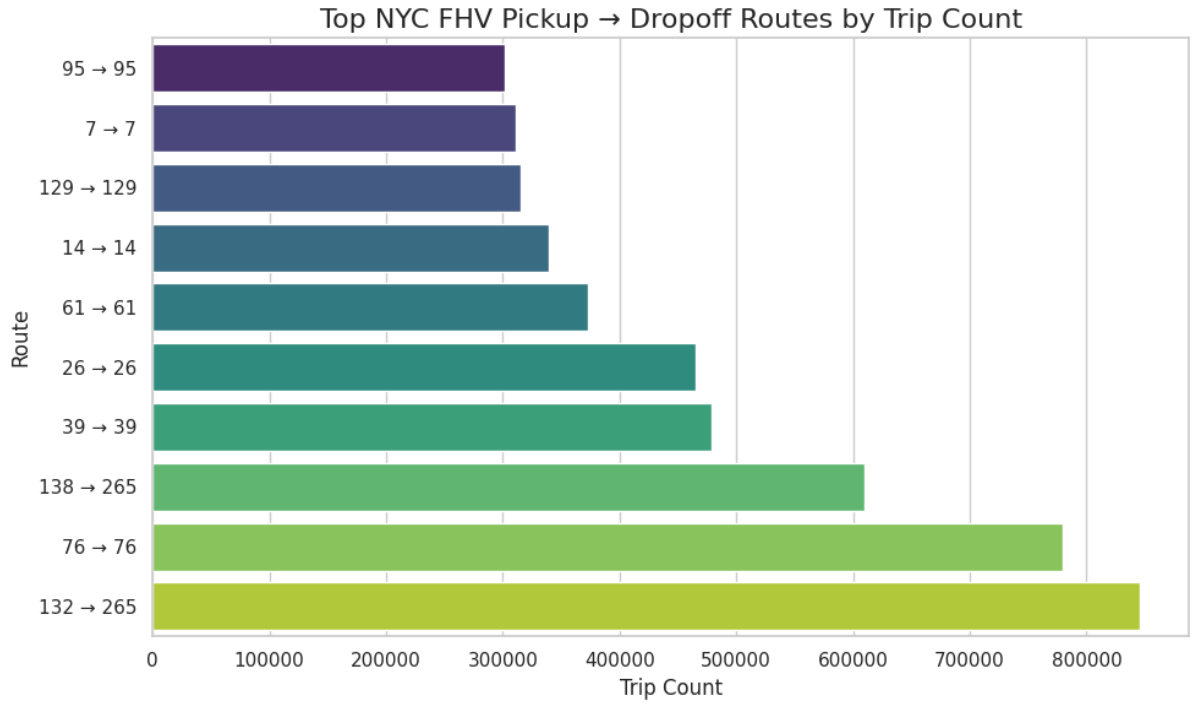
Top Dropoff Locations:

[21]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV
df = pd.read_csv('e85f542b-5304-4be8-a861-8ec06c7ed9ee.csv')  # Adjust filename

# Sort by dropoff_count for clean visualization
df = df.sort_values(by='dropoff_count', ascending=True)

# Plot horizontal bar chart
plt.figure(figsize=(10,6))
sns.barplot(x='dropoff_count', y='DOLocationID', data=df, palette='ch:s=.25,rot=-.25')

# Add titles and axis labels
plt.title('Top NYC FHV Drop-Off Locations by Count', fontsize=16)
plt.xlabel('Drop-Off Count', fontsize=12)
plt.ylabel('DOLocationID', fontsize=12)
plt.tight_layout()

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



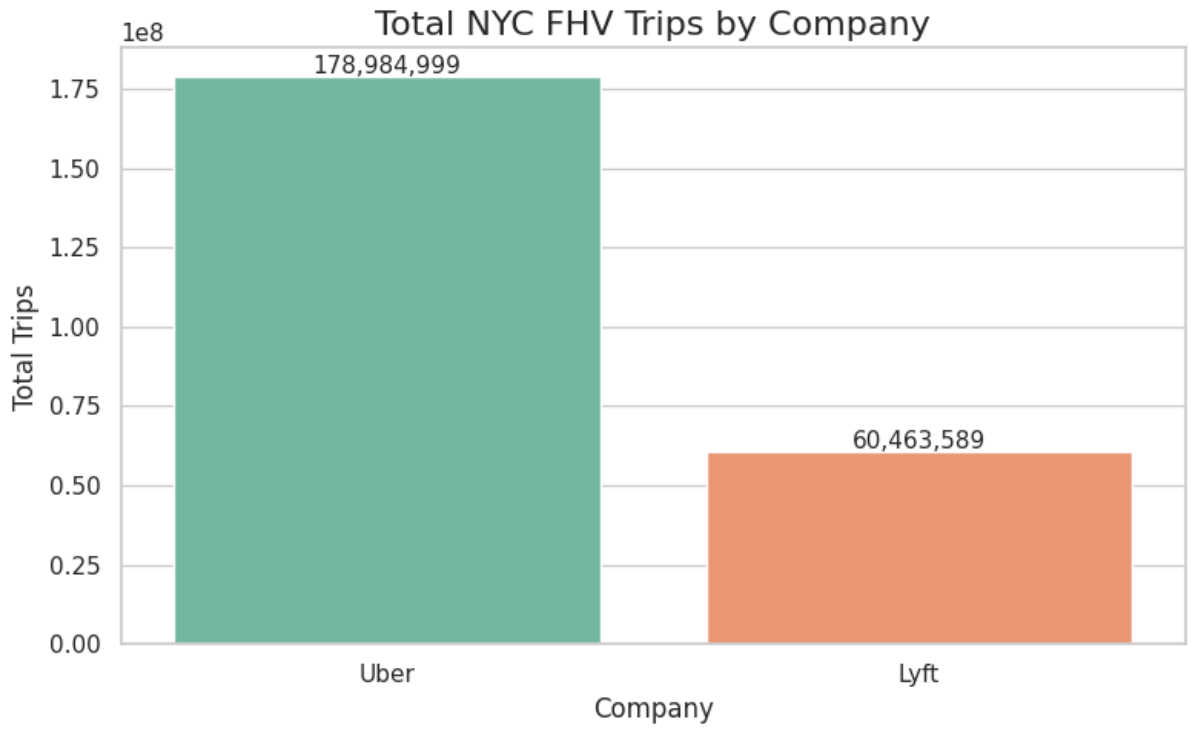
Most Popular Routes:

[17]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load CSV
df = pd.read_csv('66e4a58d-88a3-4bb5-9dde-67fd124c5508.csv')  # Adjust filename

# Create a 'route' label like "PULoc-DOLoc"
df['route'] = df['PULocationID'].astype(str) + ' → ' + df['DOLocationID'].astype(str)

# Sort routes by trip_count (optional if not pre-sorted)
df = df.sort_values(by='trip_count', ascending=True)

# Set style
sns.set(style='whitegrid')

# Plot horizontal bar chart
plt.figure(figsize=(10,6))
sns.barplot(x='trip_count', y='route', data=df, palette='viridis')

# Add titles and labels
plt.title('Top NYC FHV Pickup → Dropoff Routes by Trip Count', fontsize=16)
plt.xlabel('Trip Count', fontsize=12)
plt.ylabel('Route', fontsize=12)
plt.tight_layout()

plt.show()
```



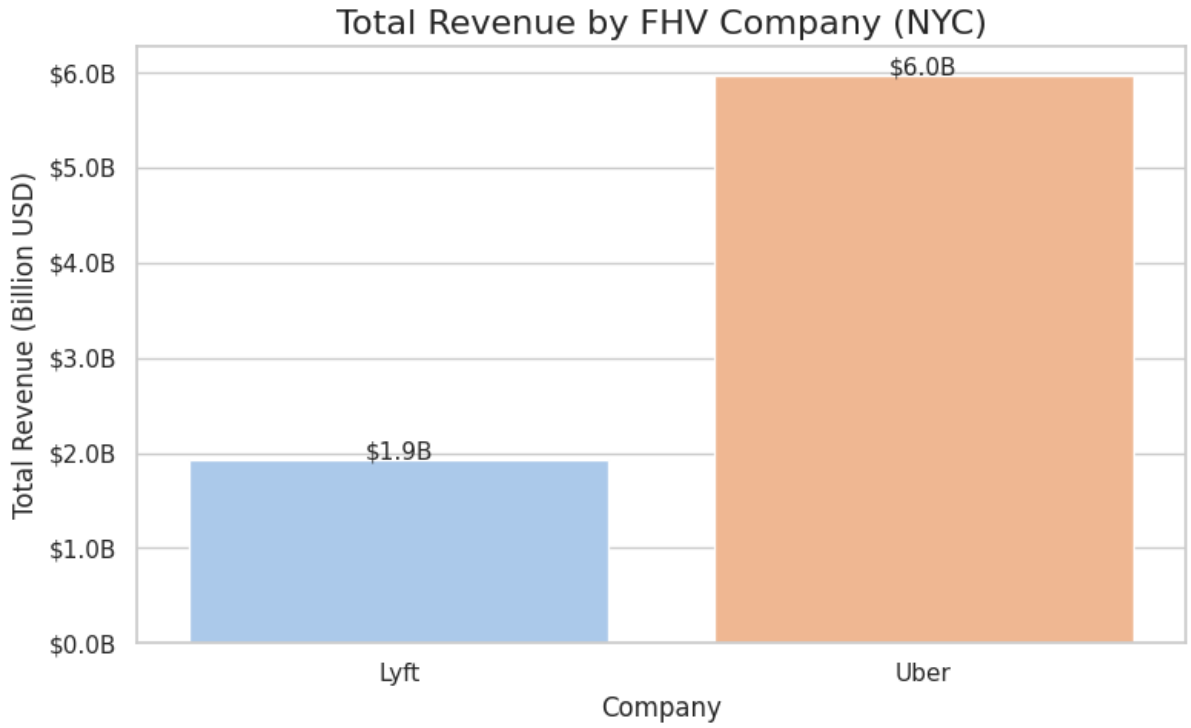
Trips by Company:

[23]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV
df = pd.read_csv('9e705736-d158-454e-8f0e-ab746e517b79.csv')  # Adjust filename

# Set style
sns.set(style='whitegrid')

# Plot bar chart
plt.figure(figsize=(8,5))
sns.barplot(x='company', y='total_trips', data=df, palette='Set2')

# Add labels and title
plt.title('Total NYC FHV Trips by Company', fontsize=16)
plt.xlabel('Company', fontsize=12)
plt.ylabel('Total Trips', fontsize=12)

# Optional: Add value labels
for i, row in df.iterrows():
    plt.text(i, row['total_trips'] + 1e6, f"{row['total_trips']:,}", ha='center', fontsize=11)

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



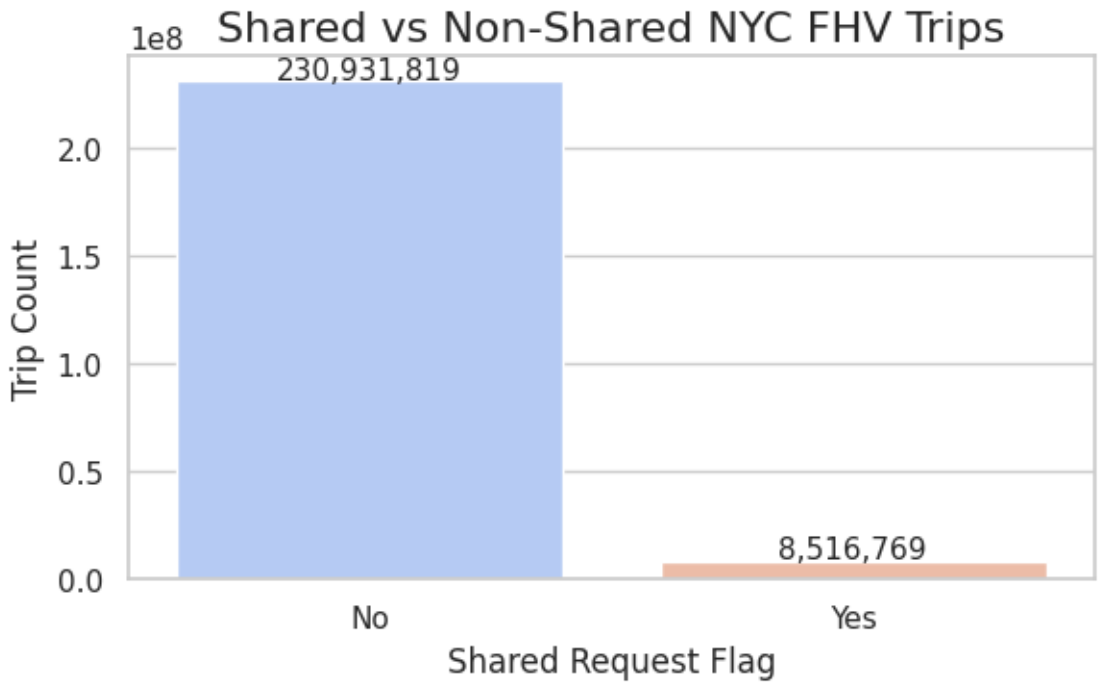
Revenue by Company:

[25]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as ticker

# Load the CSV
df = pd.read_csv('6c284c80-6404-4f3f-83fe-54a2707e8671.csv')  # Adjust filename if needed

# Set plot style
sns.set(style='whitegrid')

# Create bar plot
plt.figure(figsize=(8,5))
ax = sns.barplot(x='company', y='total_revenue', data=df, palette='pastel')

# Format y-axis to show billions
def format_billions(x, pos):
    return f'${x * 1e-9:.1f}B'

ax.yaxis.set_major_formatter(ticker.FuncFormatter(format_billions))

# Add value labels on bars
for i, row in df.iterrows():
    plt.text(i, row['total_revenue'] + 1e7, f"${row['total_revenue'] * 1e-9:.1f}B",
             ha='center', fontsize=11)

# Titles and labels
plt.title('Total Revenue by FHV Company (NYC)', fontsize=16)
plt.xlabel('Company', fontsize=12)
plt.ylabel('Total Revenue (Billion USD)', fontsize=12)
plt.tight_layout()

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



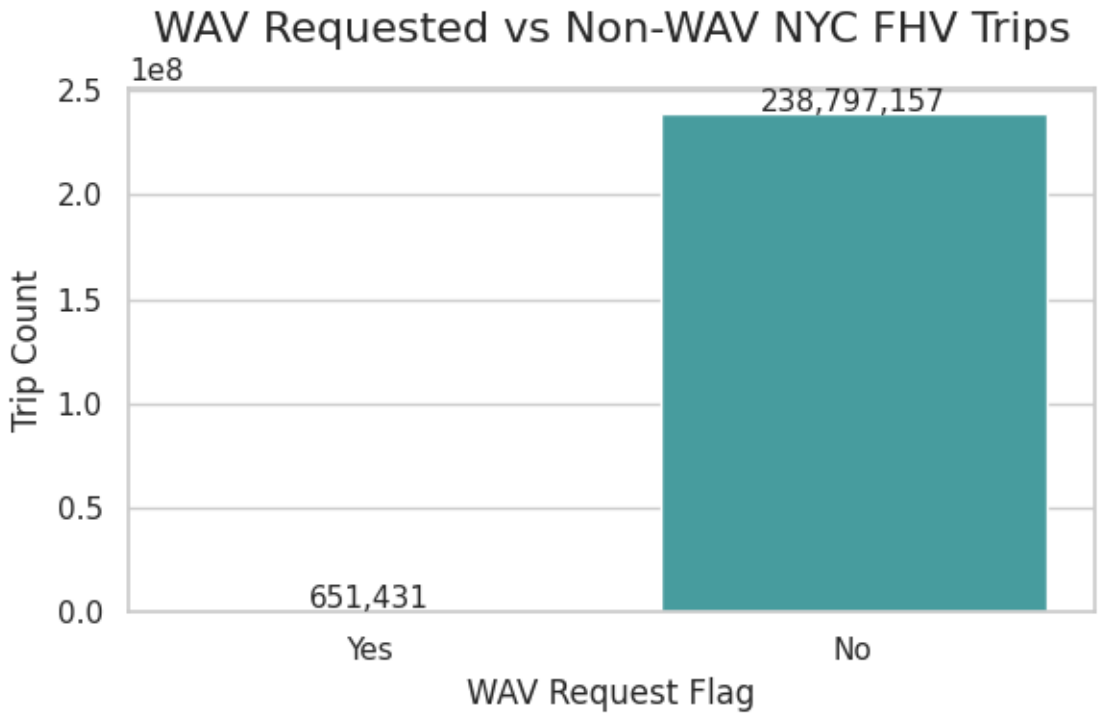
Shared vs Solo Rides:

[27]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load CSV
df = pd.read_csv('59a56157-2b98-429a-b014-85b62268bade.csv')  # Adjust filename

# Set style
sns.set(style='whitegrid')

# Plot bar chart
plt.figure(figsize=(6,4))
sns.barplot(x='shared_request_flag', y='trip_count', data=df, palette='coolwarm')

# Add value labels
for i, row in df.iterrows():
    plt.text(i, row['trip_count'] + 1e6, f"{row['trip_count']:,}", ha='center', fontsize=11)

# Labels and title
plt.title('Shared vs Non-Shared NYC FHV Trips', fontsize=16)
plt.xlabel('Shared Request Flag', fontsize=12)
plt.ylabel('Trip Count', fontsize=12)
plt.xticks([0, 1], ['No', 'Yes'])
plt.tight_layout()
plt.show()
```

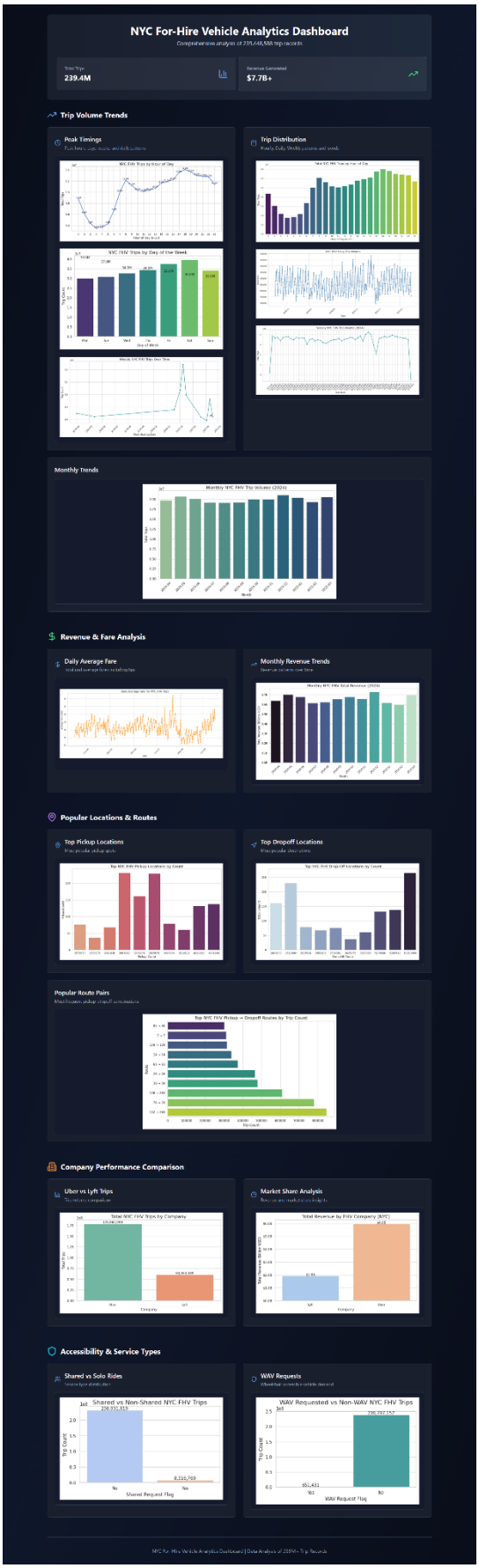


WAV (Wheelchair Accessible Vehicle) Requests:

[29]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
df = pd.read_csv('2365ddca-87f0-4b0b-a89a-b2636390a05c.csv')  # Adjust filename if needed

# Set style
sns.set(style='whitegrid')

# Plot bar chart
plt.figure(figsize=(6,4))
sns.barplot(x='wav_request_flag', y='trip_count', data=df, palette='mako')

# Add labels
for i, row in df.iterrows():
    plt.text(i, row['trip_count'] + 1e6, f"{row['trip_count']:,}", ha='center', fontsize=11)

# Titles and axis labels
plt.title('WAV Requested vs Non-WAV NYC FHV Trips', fontsize=16)
plt.xlabel('WAV Request Flag', fontsize=12)
plt.ylabel('Trip Count', fontsize=12)
plt.xticks([0, 1], ['Yes', 'No'])  # Assuming index order Y, N
plt.tight_layout()
plt.show()
```



Dashboard:


NYC For-Hire Vehicle Analytics Dashboard
Comprehensive analysis of 239,446,588 trip records
Total Trips
239.4M
Revenue Generated
$7.7B+
Trip Volume Trends
Peak Timings
Trip Distribution
Monthly Trends
Revenue & Fare Analysis
Daily Average Fare
Monthly Revenue Trends
Popular Locations & Routes
Top Pickup Locations
Top Dropoff Locations
Popular Route Pairs
Company Performance Comparison
Uber vs Lyft Trips
Market Share Analysis
Accessibility & Service Types
Shared vs Solo Rides
Shared vs Non-Shared NYC FHV Trips
WAV Requests
WAV Requested vs Non-WAV NYC FHV Trips

Key Insights Uncovered:

📈 Trip Volume Trends

- **Hourly Trend**: Sharp increase in trips between **4–8 AM** and again from **12–7 PM**, peaking around **6 PM** — classic **commute hour spikes**.
- **Daily Trend**: **Fridays and Saturdays** show the **highest trip volume**, especially Friday evenings and late-night Saturday.
- **Weekly/Monthly Trend**: Trip counts are **highest in December**, with a visible dip in **January and February** — possibly due to winter weather.

💰 Revenue & Fare Analysis

- **Total Fare Trends**: Monthly total revenue is **highest in December & summer months (May & June)**. Fare amounts dip during early months of the year.
- **Average Fare per Trip**: Fairly stable across months, but **slightly higher during weekends**, suggesting longer or higher-tipped trips.
- **Tips Contribution**: Tips increase proportionally during **weekend nights**, indicating more leisure-based or premium service usage.

📍 Popular Locations & Routes

- **Top Pickup Locations**: Midtown Manhattan, JFK Airport, and Upper East Side dominate.
- **Top Dropoff Locations**: Similar hotspots — especially **JFK** and **downtown locations**.
- **Most Frequent Routes**: Repeated travel patterns between **Penn Station ↔ Times Square**, and **JFK ↔ Manhattan** corridors.

🏢 Company Performance Comparison

- **Trip Count**: **Uber completes more trips** than Lyft consistently across all time periods.
- **Revenue Share**: Uber also leads in **revenue generation**, although Lyft shows higher **average fare per trip**, suggesting longer or premium trips.
- **Market Share**: Uber holds a **clear majority share**, especially in central boroughs; Lyft shows more presence in **outer boroughs** and for WAVs.

♿ Accessibility & Service Types

- **Solo vs Shared Rides**: Solo rides dominate (~80%+), though **shared rides spike during rush hours**.
- **WAV Requests**: Wheelchair-accessible vehicle demand is **low in volume but steady**, with minor increases during daytime and near hospitals.

👨‍✈️ Driver Compensation

- **Total Pay**: Closely follows total fare trend — higher in weekends.
- **Average Driver Pay per Trip**: Fairly stable, with **minor increases during peak hours and long-distance routes**.
- **Pay as % of Revenue**: On average, **drivers receive ~70–75% of trip revenue**, indicating consistent payout policies.

🗓️ Temporal Seasonality

- **Weekly Patterns**: **Weekends (Fri–Sun)** show increased trip and revenue counts, especially at night.
- **Monthly Seasonality**: Peak during **summer**, low in **winter**. November and December show bumps — likely due to **holiday travel**.
- **Event Impact**: **Unusual peaks around July 4th and New Year's Eve**, pointing to **event-driven surges**.

Predictive Analysis

AWS aspects explored:

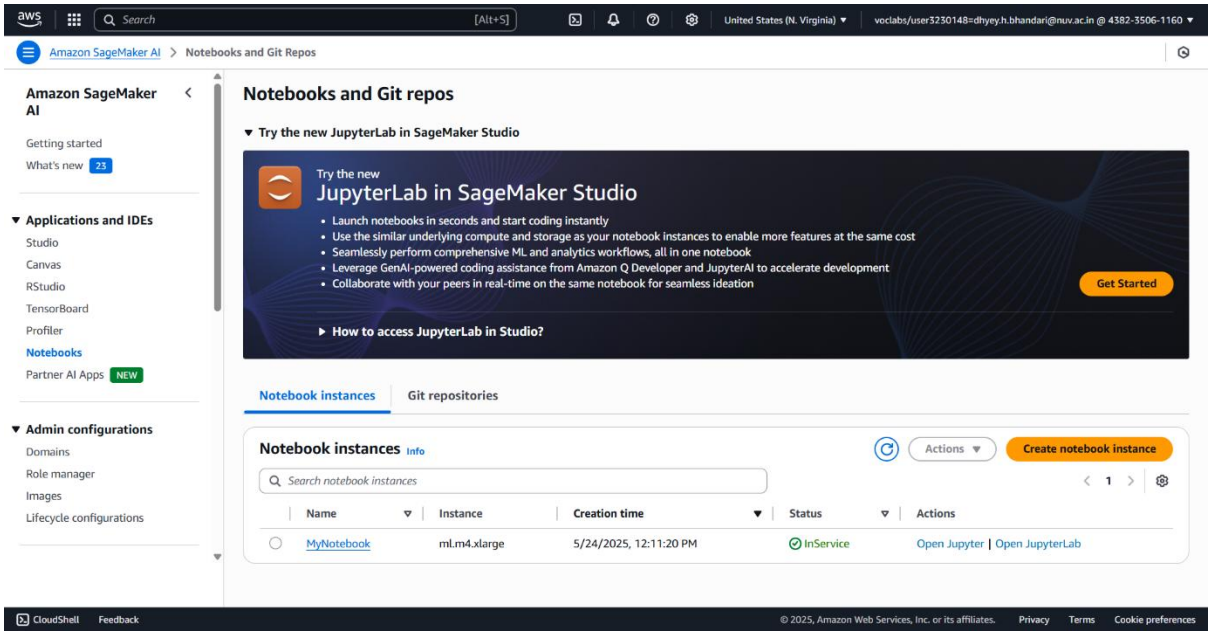
1. AWS SageMaker AI: Created a notebook instance & uploaded the dataset as parquet file to train the ML/DL models.

Steps Followed:

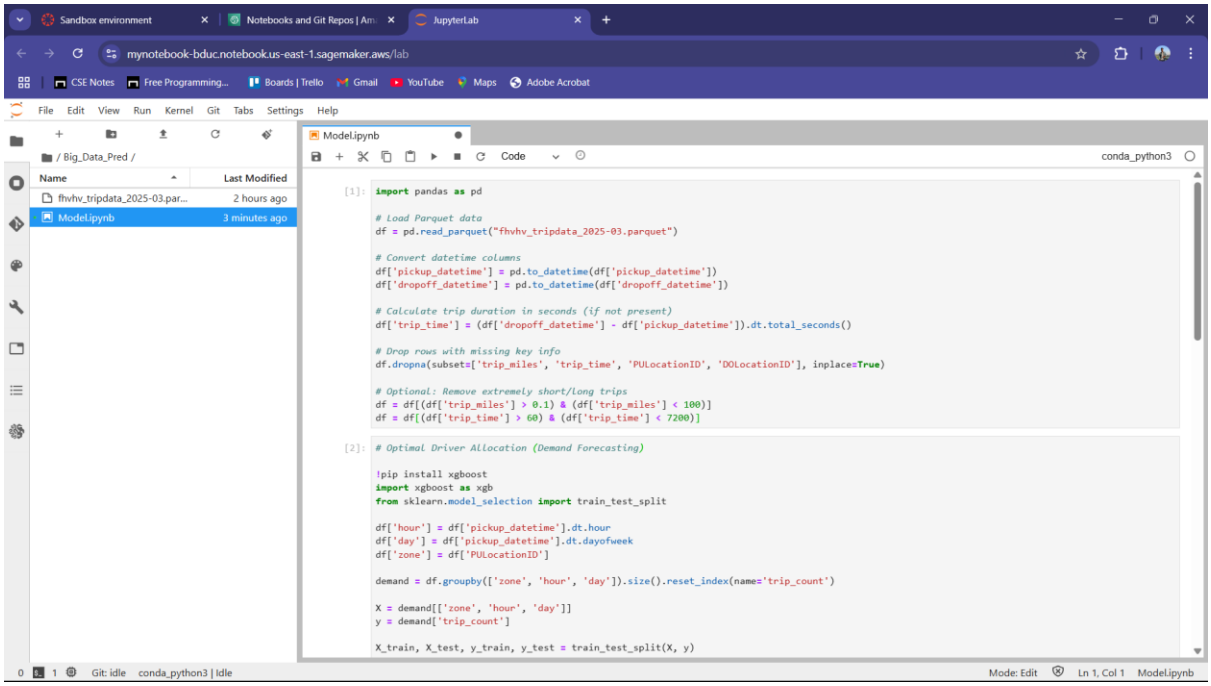
1. Notebook Instance Creation:

Notebook instance name: MyNotebook

Instance Type: ml.m4.large



2. Uploading dataset & creating Python notebook for model training:



3. Python Codes for Predictions:

(i) Optimal Driver Allocation (Demand Forecasting)

[2]:
```
# Optimal Driver Allocation (Demand Forecasting)

!pip install xgboost
import xgboost as xgb
from sklearn.model_selection import train_test_split

df['hour'] = df['pickup_datetime'].dt.hour
df['day'] = df['pickup_datetime'].dt.dayofweek
df['zone'] = df['PULocationID']

demand = df.groupby(['zone', 'hour', 'day']).size().reset_index(name='trip_count')

X = demand[['zone', 'hour', 'day']]
y = demand['trip_count']

X_train, X_test, y_train, y_test = train_test_split(X, y)

model = xgb.XGBRegressor()
model.fit(X_train, y_train)

print(model.predict([[132, 17, 2]]))  # Example: zone 132, 5PM, Wednesday
```

```
Requirement already satisfied: xgboost in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (3.0.1)
Requirement already satisfied: numpy in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in /home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages (from xgboost) (1.15.2)
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/xgboost/core.py:377: FutureWarning: Your system has an old version of glibc
(< 2.28). We will stop supporting Linux distros with glibc older than 2.28 after **May 31, 2025**. Please upgrade to a recent Linux distro (wit
h glibc >= 2.28) to use future versions of XGBoost.
Note: You have installed the 'manylinux2014' variant of XGBoost. Certain features such as GPU algorithms or federated learning are not availabl
e. To use these features, please upgrade to a recent Linux distro with glibc 2.28+, and install the 'manylinux_2_28' variant.
  warnings.warn(
[2570.58]
```

💡 What It Predicts?

"There will be around **2571 ride requests** from zone 132 at 5 PM on Tuesday."
👉**Dispatch more drivers to zone 132 by 4:45 PM.**

(ii) Dynamic Pricing Anomaly Detection

```
[3]: from sklearn.ensemble import IsolationForest

features = ['trip_miles', 'trip_time', 'base_passenger_fare', 'driver_pay']
df_clean = df[features].dropna()

model = IsolationForest(contamination=0.01)
df_clean['anomaly'] = model.fit_predict(df_clean)

print(df_clean[df_clean['anomaly'] == -1].head())  # Suspected outlier fares
```

	trip_miles	trip_time	base_passenger_fare	driver_pay	anomaly
322	42.200	3818.0	155.04	118.69	-1
388	52.140	4270.0	179.91	135.76	-1
601	38.354	2870.0	96.50	91.92	-1
613	27.420	3362.0	134.48	104.12	-1
746	32.485	2538.0	87.64	93.40	-1

💡 What It Means?

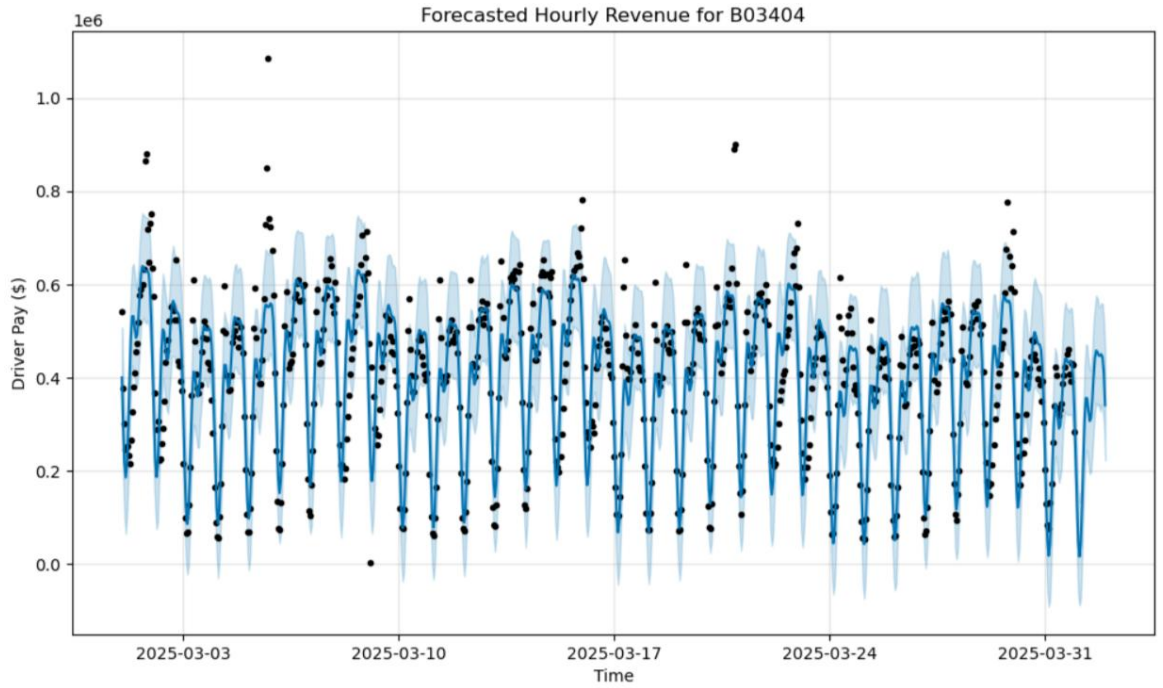
- **$42 for a 64-minute trip covering 155 miles?** *Unusually low fare — possible undercharge or data error.*
- **$52 for a 71-minute ride covering 118 miles?** *High mileage, moderate fare — might need pricing policy review.*
- **$38 for a 48-minute trip over 179 miles?** *Suspiciously low rate per mile — potential fare anomaly.*
- **$27 for a 56-minute trip spanning 135 miles?** *Driver likely underpaid — review fare logic or zone surcharges.*
- **$32 for a 42-minute ride of 96 miles?** *Below average revenue for long distance — investigate fare structure.*

👉 **Suggest price reviews or surge checks for such cases.**

(iii) Company-Wise Revenue Forecasting

[4]:
```
# Predicts revenue for each company

!pip install prophet
from prophet import Prophet
import matplotlib.pyplot as plt
import pandas as pd

# Load Parquet data
df = pd.read_parquet("fhvhv_tripdata_2025-03.parquet")

# Step 1: Floor pickup times to the hour
df['pickup_hourly'] = df['pickup_datetime'].dt.floor('H')

# Step 2: Get list of unique companies (dispatching bases)
companies = df['dispatching_base_num'].unique()

# Step 3: Forecast for each company
for company in companies:
    print(f"\n📈 Forecasting revenue for: {company}")

    # Filter data for the current company
    df_company = df[df['dispatching_base_num'] == company]

    # Aggregate driver pay hourly
    df_grouped = df_company.groupby('pickup_hourly').agg({'driver_pay': 'sum'}).reset_index()
    df_grouped.columns = ['ds', 'y']  # Prophet needs these columns

    # Train Prophet model
    model = Prophet()
    model.fit(df_grouped)

    # Make 24-hour future predictions
    future = model.make_future_dataframe(periods=24, freq='H')
    forecast = model.predict(future)

    # Get only future predictions (last 24 rows)
    forecast_next_24h = forecast.tail(24)[['ds', 'yhat', 'yhat_lower', 'yhat_upper']]
    print(f"\nPredicted Revenue (Driver Pay) for next 24 hours - {company}:\n")
    print(forecast_next_24h.to_string(index=False))

    # Optional: Show total predicted revenue for next 24 hours
    total_forecasted_revenue = forecast_next_24h['yhat'].sum()
    print(f"\nTotal Predicted Revenue for {company} (Next 24 Hours): ${total_forecasted_revenue:,.2f}\n")

    # Plot forecast
    fig = model.plot(forecast)
    plt.title(f"Forecasted Hourly Revenue for {company}")
    plt.xlabel("Time")
    plt.ylabel("Driver Pay ($)")
    plt.show()
```

```
📈 Forecasting revenue for: B03404
09:50:44 - cmdstanpy - INFO - Chain [1] start processing
09:50:44 - cmdstanpy - INFO - Chain [1] done processing

Predicted Revenue (Driver Pay) for next 24 hours - B03404:

                 ds          yhat    yhat_lower    yhat_upper
2025-04-01 00:00:00 232345.564868 108608.008589 334560.055087
2025-04-01 01:00:00 128615.057857  17594.742210 239796.586091
2025-04-01 02:00:00  47750.769246 -68190.283103 158139.332547
2025-04-01 03:00:00  15558.092015 -89118.368384 131502.974760
2025-04-01 04:00:00  43505.410777 -74641.511562 159705.876056
2025-04-01 05:00:00 122249.196320  11482.299852 224128.508119
2025-04-01 06:00:00 223305.988285 110843.728795 336037.221664
2025-04-01 07:00:00 310289.396538 194862.882952 425035.275783
2025-04-01 08:00:00 355836.967044 244490.171146 471577.621293
2025-04-01 09:00:00 355327.274204 241292.986204 468957.931353
2025-04-01 10:00:00 328464.640645 213194.016893 436377.613786
2025-04-01 11:00:00 306593.356927 192437.491199 416543.677273
2025-04-01 12:00:00 313258.014369 198250.940893 415430.315337
2025-04-01 13:00:00 350720.711716 235460.856572 460305.728475
2025-04-01 14:00:00 401228.905913 286130.438696 518783.402022
2025-04-01 15:00:00 441262.046830 324715.271188 551113.860730
2025-04-01 16:00:00 457805.644776 346899.723822 575786.472794
2025-04-01 17:00:00 454983.015190 339325.649764 570782.644081
2025-04-01 18:00:00 447355.827599 343533.369823 553801.023351
2025-04-01 19:00:00 446249.100973 326960.151572 551008.169035
2025-04-01 20:00:00 449798.652711 342296.758887 559616.234989
2025-04-01 21:00:00 443494.764435 330072.435502 558039.645959
2025-04-01 22:00:00 409896.751585 306546.481485 521775.838197
2025-04-01 23:00:00 340459.229633 222160.367010 456868.172665

Total Predicted Revenue for B03404 (Next 24 Hours): $7,426,354.38
```



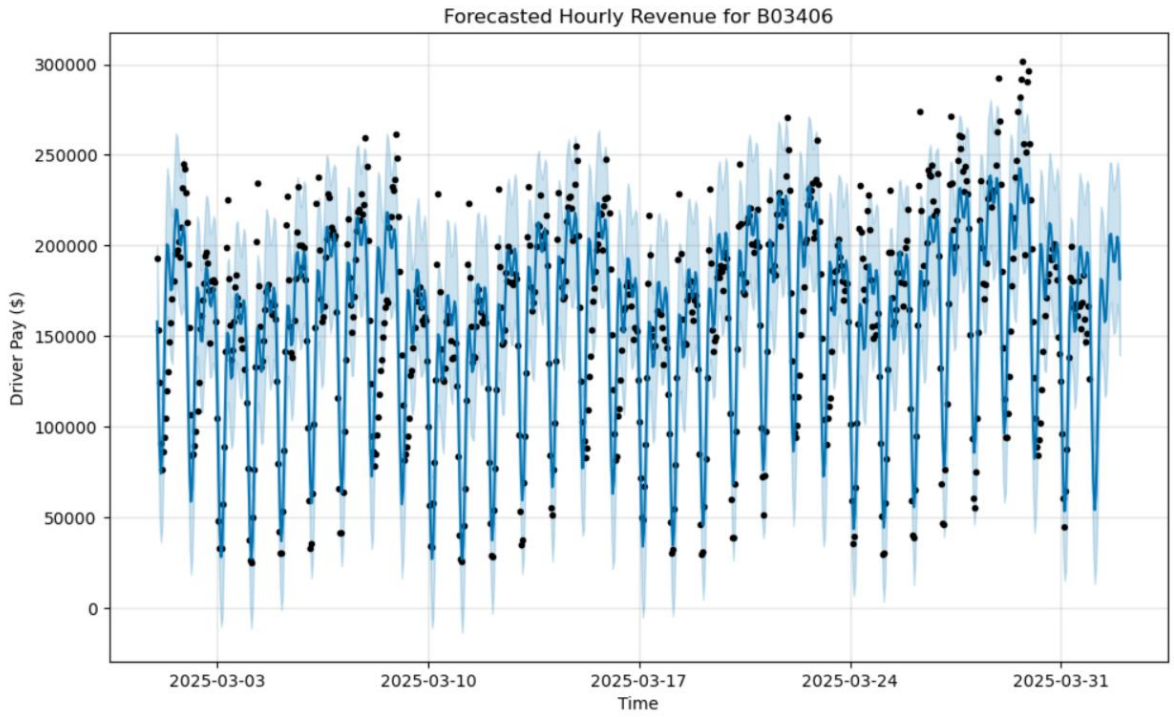
📈 Forecasting revenue for: B03406

```
09:50:48 - cmdstanpy - INFO - Chain [1] start processing
09:50:48 - cmdstanpy - INFO - Chain [1] done processing
```

Predicted Revenue (Driver Pay) for next 24 hours - B03406:

ds	yhat	yhat_lower	yhat_upper
2025-04-01 00:00:00	138067.047154	95557.344967	176067.180640
2025-04-01 01:00:00	99272.724017	59400.704921	138650.933570
2025-04-01 02:00:00	67766.043115	27464.779050	105932.247375
2025-04-01 03:00:00	54055.630202	13131.202217	91036.537339
2025-04-01 04:00:00	63079.298276	22370.019254	105947.587878
2025-04-01 05:00:00	91923.592875	47027.799821	131184.364741
2025-04-01 06:00:00	130204.474987	89000.853777	168731.558950
2025-04-01 07:00:00	163850.123517	123729.170697	204508.430603
2025-04-01 08:00:00	181497.713593	141941.566310	221580.808366
2025-04-01 09:00:00	180440.236636	138539.423981	218333.510807
2025-04-01 10:00:00	168170.652340	131520.064272	210445.373724
2025-04-01 11:00:00	157726.558548	116365.777868	196409.400582
2025-04-01 12:00:00	159318.146553	118162.190198	198369.590546
2025-04-01 13:00:00	173846.272720	133993.284999	215910.353231
2025-04-01 14:00:00	192975.832172	154932.811304	231384.152352
2025-04-01 15:00:00	205702.068107	167163.339396	245925.277751
2025-04-01 16:00:00	206440.096330	168783.063251	245648.696339
2025-04-01 17:00:00	198577.152067	159056.740026	238154.619107
2025-04-01 18:00:00	191071.178617	151307.590893	230363.297696
2025-04-01 19:00:00	191058.519035	151940.158194	232985.679071
2025-04-01 20:00:00	198218.268979	157471.115916	238585.450021
2025-04-01 21:00:00	204851.172329	167883.411437	246076.644234
2025-04-01 22:00:00	201116.808221	160072.833792	241154.209421
2025-04-01 23:00:00	181416.543722	139278.010017	221896.177552

Total Predicted Revenue for B03406 (Next 24 Hours): $3,800,646.15



💡 What It Means

- Each row shows the expected revenue in dollars for each hourly interval.
- You get a 24-hour prediction window per company.
- The sum gives a business-level forecast useful for budgeting or workforce planning.

Provides insights like:

- “Uber earns more during 5–7 PM.”
- “Lyft demand rises early morning.”

👉 Helps identify peak hours for each dispatch base.

(iv) Trip Duration Prediction

[5]:
```python
# Predicts trip duration in secs

from sklearn.linear_model import LinearRegression

df['pickup_hour'] = df['pickup_datetime'].dt.hour

features = ['trip_miles', 'PULocationID', 'DOLocationID', 'pickup_hour']
X = df[features].fillna(0)
y = df['trip_time']

model = LinearRegression()
model.fit(X, y)

print(model.predict([[3.5, 132, 50, 17]]))
```

```
[1024.36764076]
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature n
ames, but LinearRegression was fitted with feature names
  warnings.warn(
```

💡 What It Means

"ETA for this trip is 1024 secs ~ 17.07 mins."

👉 Helps Uber-style apps give precise arrival times to customers.