Rideshare Analysis

Cassia Ramelb

Dataset: rideshare.db

Tools: SQLite, Python

1 Introduction

This project explores data from the rideshare.db database, containing trip-level records from a bike-sharing network. Using SQL and Python, the analysis examines ridership trends, seasonal patterns, station activity, user behavior, and trip outliers. The goal is to uncover actionable insights for operations planning and demand management while demonstrating practical SQL and data visualization skills.

[1]:
```
import sqlite3, os, pandas as pd
import matplotlib.pyplot as plt

# path to db
DB_PATH = os.path.join("..", "data", "rideshare.db")

def run_sql(sql, params=None):
    """Helper function to run SQL and return a DF."""
    with sqlite3.connect(DB_PATH) as con:
        df =  pd.read_sql_query(sql, con, params=params)
    return df
print("Database successfully connected!")
```

```
Database successfully connected!
```

2 Trip Volume & Growth

Explore how monthly ridership has changed over time and calculate month-over-month % changes and a 3-month moving average.

2.1 Query: Monthly Trips and MoM % Change

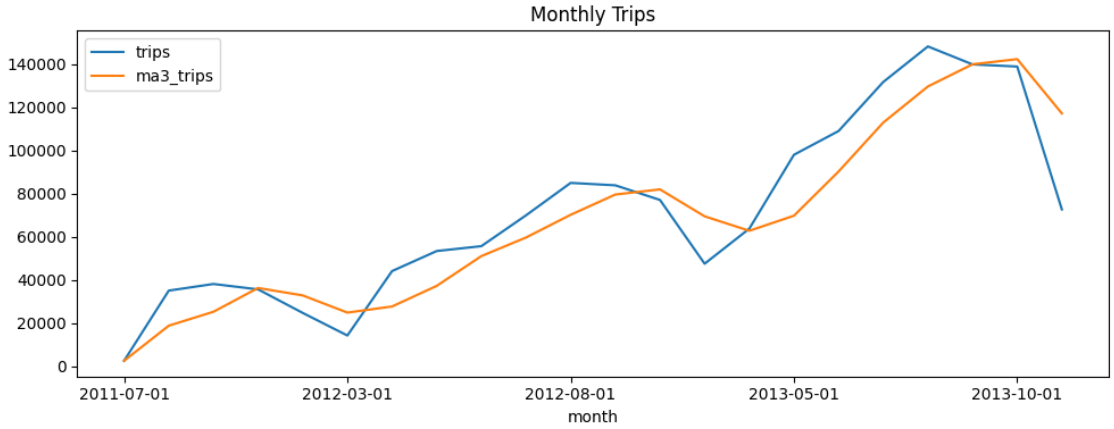
[2]:
```
START_COL = "start_date"

sql_monthly = f"""
WITH monthly AS (
```

```
  SELECT
    date(strftime('%Y-%m-01', {START_COL})) AS month,
    COUNT(*) AS trips
  FROM trips
  WHERE {START_COL} IS NOT NULL
  GROUP BY 1
)
SELECT
  month,
  trips,
  LAG(trips) OVER (ORDER BY month) AS prev_trips,
  ROUND(
    100.0 * (trips - LAG(trips) OVER (ORDER BY month))
    / NULLIF(LAG(trips) OVER (ORDER BY month), 0), 2
  ) AS mom_pct,
  ROUND(
    AVG(trips) OVER (ORDER BY month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW),
    1
  ) AS ma3_trips
FROM monthly
ORDER BY month;
"""
df_monthly = run_sql(sql_monthly)
df_monthly.head(10)
```

Month	Trips	Previous Trips	MoM (%)	3-Month Avg. Trips
2011-07-01	2,618	–	–	2,618.0
2011-08-01	35,117	2,618	1241.37	18,867.5
2011-09-01	38,184	35,117	8.73	25,306.3
2011-10-01	35,691	38,184	-6.53	36,330.7
2011-11-01	24,782	35,691	-30.57	32,885.7
2012-03-01	14,318	24,782	-42.22	24,930.3
2012-04-01	44,154	14,318	208.38	27,751.3
2012-05-01	53,447	44,154	21.05	37,306.3
2012-06-01	55,704	53,447	4.22	51,101.7
2012-07-01	69,987	55,704	25.64	59,712.7

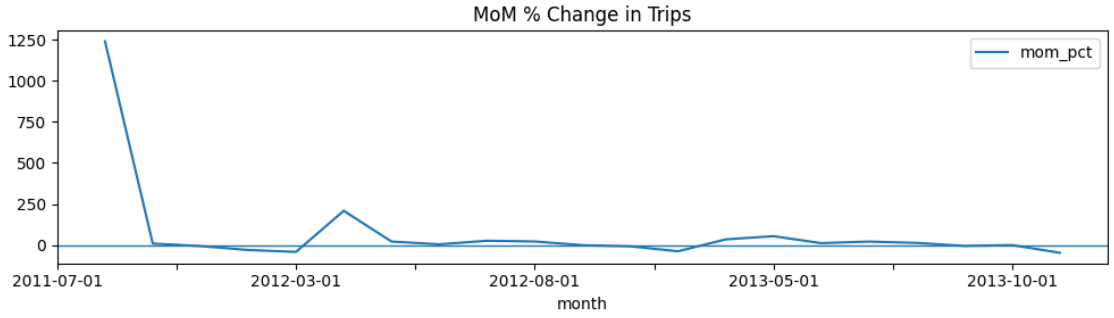
2.2 Visualizations: Monthly Trends

[3]:
```
ax = df_monthly.plot(x="month", y="trips", kind="line", figsize=(10,4),␣
↪title="Monthly Trips")
df_monthly.plot(x="month", y="ma3_trips", kind="line", ax=ax)
plt.tight_layout(); plt.show()
```



Ridership increased sharply after the July 2011 launch, peaking near 130 K monthly trips by mid-2013. The 3-month moving average smooths short-term spikes, confirming a steady upward trend with predictable winter dips.

[4]:
```
df_monthly.plot(x="month", y="mom_pct", kind="line", figsize=(10,3), title="MoM␣
↪% Change in Trips")
plt.axhline(0, linewidth=1)
plt.tight_layout(); plt.show()
```



Month-over-month growth initially spiked (+1240%) during rollout, then stabilized within ±10–20%, indicating a maturing, seasonally influenced system with sustained long-term usage.

2.3 Interpretation

Ridership showed **exponential early growth** after the July 2011 launch, with trips increasing by roughly **+1240% the following month.** After stabilization, usage settled into a **consistent seasonal pattern,** peaking during warmer months and dipping in winter. The 3-month moving average confirms a steady long-term upward trend, smoothing short-term fluctuations. Month-over-month changes later remained within ±15%, reflecting a **maturing system with stable, recurring usage cycles.**

3 Peak Usage Hours

Analyze hourly and weekday patterns to identify commuter peaks and operational hotspots.

3.1 Query: Trips by Weekday and Hour

[6]:
```
sql_wk_hr =  f"""
SELECT
    CAST(strftime('%w', {START_COL}) AS INTEGER) AS wkday_num,   -- 0=Sun ...␣
 ↪6=Sat
    CAST(strftime('%H', {START_COL}) AS INTEGER) AS hr,   -- 0...23
    COUNT(*) AS trips
FROM trips
WHERE {START_COL} IS NOT NULL
GROUP BY wkday_num, hr
ORDER BY wkday_num, hr;
"""

df_wk_hr = run_sql(sql_wk_hr)
df_wk_hr.head()
```

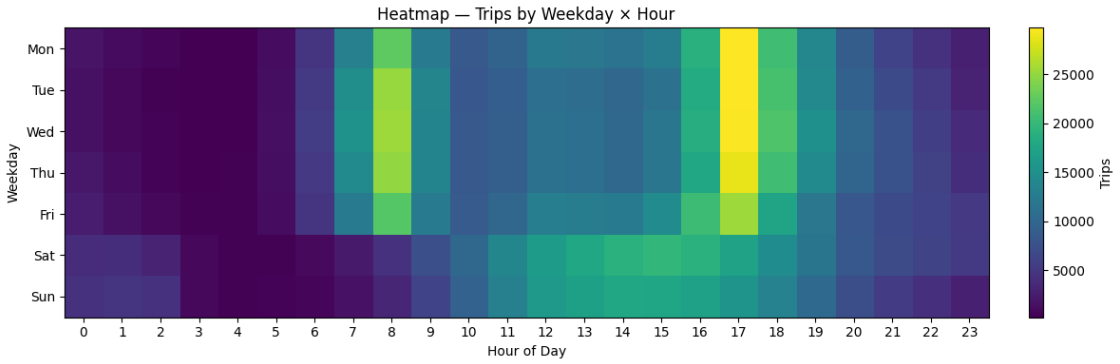
Weekday (Num)	Hour	Trips
0	0	4,344
0	1	4,730
0	2	4,436
0	3	855
0	4	401

[7]:
```
# Label week days and pivot to matrix

wk_map = {0:"Sun", 1:"Mon", 2:"Tue", 3:"Wed", 4:"Thu", 5:"Fri", 6:"Sat"}
df_wk_hr["weekday"] = df_wk_hr["wkday_num"].map(wk_map)

pivot_wk_hr = df_wk_hr.pivot(index="weekday", columns="hr", values="trips")

weekday_order = ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]
pivot_wk_hr = pivot_wk_hr.reindex(weekday_order)

pivot_wk_hr.iloc[:5, :8]
```

3.2 Visualization: Heatmap

[8]:
```
import numpy as np
fig, ax = plt.subplots(figsize=(12,4))
im = ax.imshow(pivot_wk_hr.values, aspect="auto")
ax.set_title("Heatmap - Trips by Weekday × Hour")
```

```
ax.set_xlabel("Hour of Day")
ax.set_ylabel("Weekday")

ax.set_xticks(range(24))
ax.set_xticklabels(range(24))
ax.set_yticks(range(len(pivot_wk_hr.index)))
ax.set_yticklabels(pivot_wk_hr.index)

fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04, label="Trips")

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



Weekdays show distinct commuter peaks at ≈8:00 and ≈17:00, while weekends shift toward late-morning to early-afternoon leisure riding.

3.3 Top Hour per Weekday

[9]:
```
sql_top_hour_by_wk = f"""
WITH by_wk_hr AS (
   SELECT
       CAST(strftime('%w', {START_COL}) AS INTEGER) AS wkday_num,
    CAST(strftime('%H', {START_COL}) AS INTEGER) AS hr,
    COUNT(*) AS trips
FROM trips
WHERE {START_COL} IS NOT NULL
GROUP BY wkday_num, hr
),
ranked AS (
    SELECT *,
           RANK() OVER (PARTITION BY wkday_num ORDER BY trips DESC) AS rnk
    FROM by_wk_hr
)
SELECT wkday_num, hr, trips
FROM ranked
```

```
WHERE rnk = 1
ORDER BY wkday_num;
"""
run_sql(sql_top_hour_by_wk)
```

Weekday (Num)	Hour	Trips
0	14	17,888
1	17	29,733
2	17	29,693
3	17	29,582
4	17	28,798
5	17	25,374
6	15	19,688

The 17:00 hour is consistently the busiest on weekdays, confirming strong post-work commute demand; a secondary peak appears near 08:00. On weekends, activity shifts to 14:00–15:00, reflecting recreational use rather than commuting.

3.4 Interpretation

Weekday ridership shows clear **morning and evening commute peaks,** with the busiest hour near **5 PM** and a smaller rise around **8 AM,** reflecting standard commuter behavior.

On weekends, activity shifts toward **midday (13:00–15:00),** indicating more **recreational or social trips.**

Overall, weekday demand is **bimodal,** while weekend demand is **single-peaked,** suggesting resource allocation should emphasize bike availability before **8 AM** and around **5 PM** on weekdays, and balanced distribution late mornings on weekends.

4 Seasonal Patterns

Analyze how average trip duration fluctuates by month.

4.1 Query: Average Trip Duration by Month

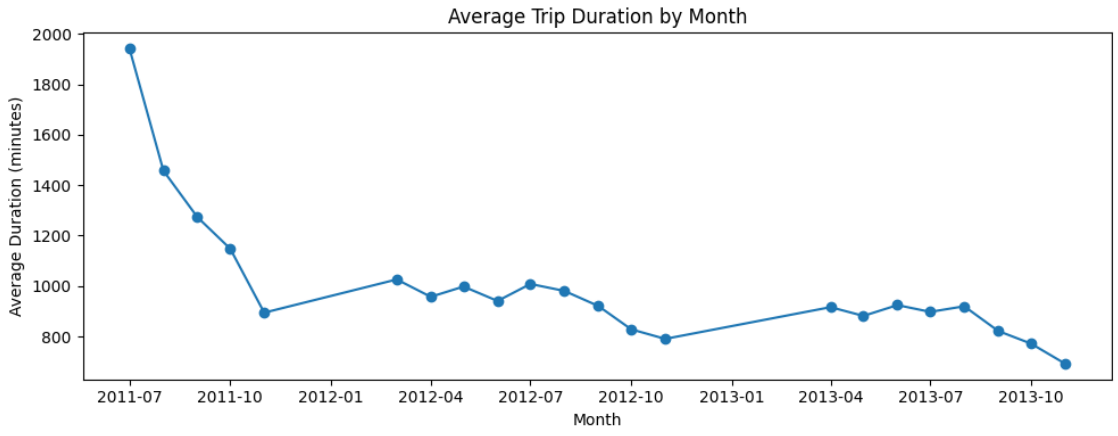
[11]:
```
sql_seasonal = """
WITH monthly_duration AS (
  SELECT
    date(strftime('%Y-%m-01', start_date)) AS month,
    AVG(duration) AS avg_duration
  FROM trips
  WHERE start_date IS NOT NULL AND duration > 0
  GROUP BY 1
)
SELECT month, ROUND(avg_duration, 2) AS avg_duration
FROM monthly_duration
```

```
ORDER BY month;
"""

df_seasonal = run_sql(sql_seasonal)
df_seasonal.head(10)
```

Month	Average Duration (sec)
2011-07-01	1,941.64
2011-08-01	1,457.26
2011-09-01	1,273.16
2011-10-01	1,147.90
2011-11-01	894.41
2012-03-01	1,026.15
2012-04-01	957.33
2012-05-01	997.92
2012-06-01	940.83
2012-07-01	1,008.28

4.2 Visualization: Average Trip Duration

[12]:
```
df_seasonal["month"] = pd.to_datetime(df_seasonal["month"])

# Line Plot
import matplotlib.pyplot as plt
plt.figure(figsize=(10,4))
plt.plot(df_seasonal["month"], df_seasonal["avg_duration"], marker="o")
plt.title("Average Trip Duration by Month")
plt.xlabel("Month"); plt.ylabel("Average Duration (minutes)")
plt.tight_layout(); plt.show()
```



Average trip duration drops sharply after launch, then stabilizes at ≈10–20 minutes with longer

rides in warmer months and shorter rides in winter.

4.3 Interpretation

Average trip duration shows a clear **seasonal pattern.** After unusually long rides during the 2011 launch period (likely data anomalies), durations stabilize around **10–20 minutes.** Trips are **slightly longer April–September** and **shorter November–February,** consistent with leisure riding in summer and practical commuting in winter, aligning with Section 1's seasonality.

5 Top Origin & Destination Stations

Identify the most popular start and end stations in the rideshare network.

5.1 Query: Top 10 Start and End Stations

[14]:
```
sql_top_stations = """
WITH start_counts AS (
  SELECT
    s.station AS station_name,
    COUNT(*) AS start_trips,
    RANK() OVER (ORDER BY COUNT(*) DESC) AS start_rank
  FROM trips t
  JOIN stations s ON t.start_station = s.id
  GROUP BY s.station
),
end_counts AS (
  SELECT
    s.station AS station_name,
    COUNT(*) AS end_trips,
    RANK() OVER (ORDER BY COUNT(*) DESC) AS end_rank
  FROM trips t
  JOIN stations s ON t.end_station = s.id
  GROUP BY s.station
)
-- emulate FULL OUTER JOIN with two LEFT JOINs and UNION
SELECT
  sc.station_name,
  sc.start_trips,
  IFNULL(ec.end_trips, 0) AS end_trips,
  sc.start_rank,
  ec.end_rank
FROM start_counts sc
LEFT JOIN end_counts ec USING (station_name)
WHERE sc.start_rank <= 10

UNION
```

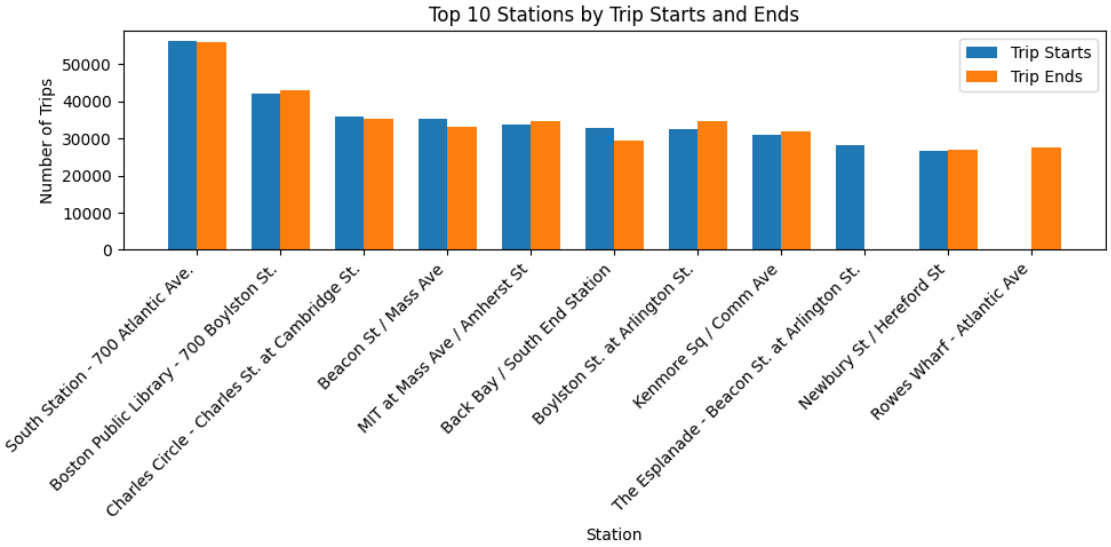
```
SELECT
  ec.station_name,
  IFNULL(sc.start_trips, 0) AS start_trips,
  ec.end_trips,
  sc.start_rank,
  ec.end_rank
FROM end_counts ec
LEFT JOIN start_counts sc USING (station_name)
WHERE ec.end_rank <= 10

ORDER BY start_rank, end_rank, station_name;
"""
df_top_stations = run_sql(sql_top_stations)
df_top_stations.head(10)
```

Station Name	Start Trips	End Trips	Start Rank	End Rank
South Station – 700 Atlantic Ave.	56,123	56,003	1	1
Boston Public Library – 700 Boylston St.	41,994	42,870	2	2
Charles Circle – Charles St. at Cambridge St.	35,984	35,196	3	3
Beacon St / Mass Ave	35,275	33,186	4	6
MIT at Mass Ave / Amherst St.	33,644	34,557	5	4
Back Bay / South End Station	32,677	29,503	6	8
Boylston St. at Arlington St.	32,410	34,524	7	5
Kenmore Sq / Comm Ave	30,835	32,036	8	7
The Esplanade – Beacon St. at Arlington St.	28,119	26,180	9	13
Newbury St / Hereford St.	26,733	26,982	10	10

5.2 Visualization: Bar Chart

[15]:
```
df_start = df_top_stations.nlargest(10, "start_trips")[["station_name",␣
↪"start_trips"]]
df_end   = df_top_stations.nlargest(10, "end_trips")[["station_name",␣
↪"end_trips"]]

df_merge = pd.merge(df_start, df_end, on="station_name", how="outer").fillna(0)
df_merge = df_merge.sort_values(by="start_trips", ascending=False)

plt.figure(figsize=(10,5))
width = 0.35
x = range(len(df_merge))

plt.bar(x, df_merge["start_trips"], width=width, label="Trip Starts")
plt.bar([i + width for i in x], df_merge["end_trips"], width=width, label="Trip␣
↪Ends")
```

```
plt.xticks([i + width/2 for i in x], df_merge["station_name"], rotation=45,␣
 ↪ha="right")
plt.title("Top 10 Stations by Trip Starts and Ends")
plt.xlabel("Station"); plt.ylabel("Number of Trips")
plt.legend()
plt.tight_layout()
plt.show()
```



South Station and Boston Public Library record the highest trip starts and ends, revealing their roles as key downtown transit hubs.

5.3 Interpretation

High overlap between top start and end stations shows **balanced, two-way demand** concentrated in **busy downtown hubs.** These locations likely support **commuter transfers** between bikes and public transit. Stations with fewer trip ends may need **periodic rebalancing** to maintain bike availability.

6 Station Flow Imbalance

Explore the difference between trip starts and ends at each station

6.1 Query: Starts, Ends, and Net Flow

[17]:
```
sql_flow = """
WITH start_counts AS (
  SELECT s.station AS station_name, COUNT(*) AS starts
  FROM trips t
  JOIN stations s ON t.start_station = s.id
```

```
  GROUP BY s.station
),
end_counts AS (
  SELECT s.station AS station_name, COUNT(*) AS ends
  FROM trips t
  JOIN stations s ON t.end_station = s.id
  GROUP BY s.station
),
flow AS (
  -- part A: all rows from start_counts (LEFT JOIN to end_counts)
  SELECT
    sc.station_name,
    sc.starts,
    IFNULL(ec.ends, 0) AS ends,
    sc.starts - IFNULL(ec.ends, 0) AS net_flow,
    sc.starts + IFNULL(ec.ends, 0) AS total_activity
  FROM start_counts sc
  LEFT JOIN end_counts ec ON sc.station_name = ec.station_name

  UNION ALL

  -- part B: rows that exist ONLY in end_counts (not in start_counts)
  SELECT
    ec.station_name,
    IFNULL(sc.starts, 0) AS starts,
    ec.ends,
    IFNULL(sc.starts, 0) - ec.ends AS net_flow,
    IFNULL(sc.starts, 0) + ec.ends AS total_activity
  FROM end_counts ec
  LEFT JOIN start_counts sc ON sc.station_name = ec.station_name
  WHERE sc.station_name IS NULL
)
SELECT *
FROM flow
ORDER BY ABS(net_flow) DESC, total_activity DESC;
"""

df_flow = run_sql(sql_flow)
df_flow.head(10)
```

Station Name	Starts	Ends	Net Flow	Total Activity
Back Bay / South End Station	32,677	29,503	3,174	62,180
Harvard Square at Mass Ave / Dunster	23,985	26,642	-2,657	50,627
Mayor Thomas M. Menino – Government Center	23,130	20,567	2,563	43,697
Boylston St. at Arlington St.	32,410	34,524	-2,114	66,934
Beacon St / Mass Ave	35,275	33,186	2,089	68,461
Tremont St / W Newton St	19,340	17,341	1,999	36,681
The Esplanade – Beacon St. at Arlington St.	28,119	26,180	1,939	54,299
TD Garden – Causeway at Portal Park #1	17,327	19,098	-1,771	36,425
Rowes Wharf – Atlantic Ave	25,880	27,596	-1,716	53,476
Aquarium Station – 200 Atlantic Ave.	24,288	22,666	1,622	46,954

6.2 Pick top imbalances (both directions) & preview

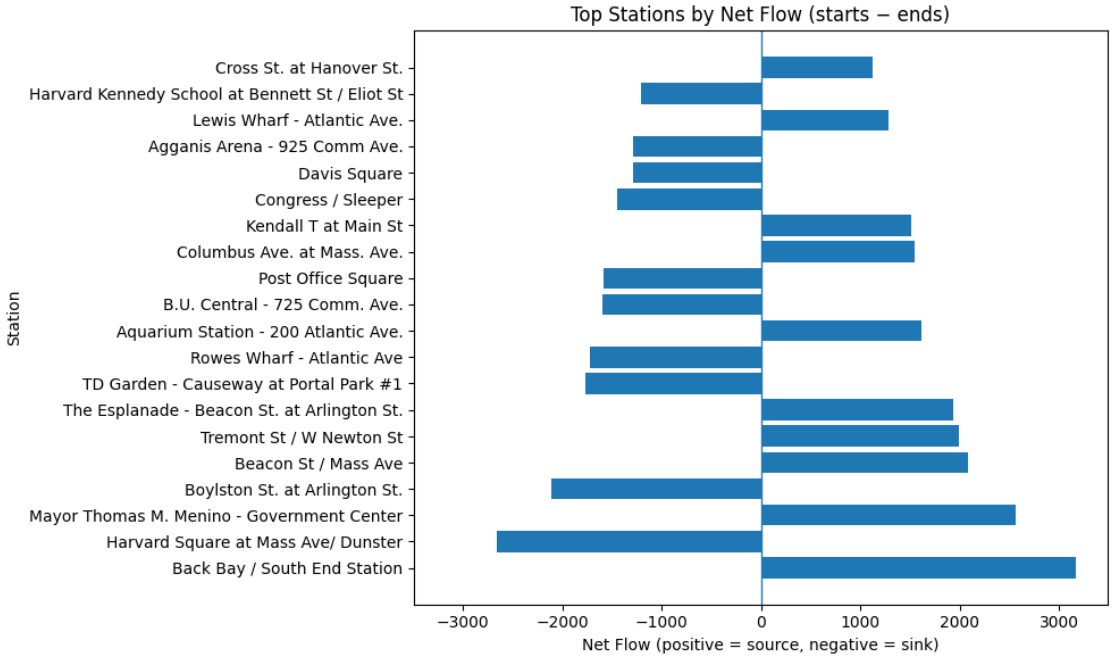
[18]:
```
# Top 10 sources (positive net) and sinks (negative net)
top_sources = df_flow.sort_values("net_flow", ascending=False).head(10)
top_sinks   = df_flow.sort_values("net_flow", ascending=True).head(10)

display(top_sources[["station_name","starts","ends","net_flow"]])
display(top_sinks[["station_name","starts","ends","net_flow"]])
```

Station Name	Starts	Ends	Net Flow
Back Bay / South End Station	32,677	29,503	3,174
Mayor Thomas M. Menino – Government Center	23,130	20,567	2,563
Beacon St / Mass Ave	35,275	33,186	2,089
Tremont St / W Newton St	19,340	17,341	1,999
The Esplanade – Beacon St. at Arlington St.	28,119	26,180	1,939
Aquarium Station – 200 Atlantic Ave.	24,288	22,666	1,622
Columbus Ave. at Mass. Ave.	17,829	16,284	1,545
Kendall T at Main St.	17,486	15,972	1,514
Lewis Wharf – Atlantic Ave.	25,872	24,587	1,285
Cross St. at Hanover St.	26,622	25,496	1,126

Station Name	Starts	Ends	Net Flow
Harvard Square at Mass Ave / Dunster	23,985	26,642	-2,657
Boylston St. at Arlington St.	32,410	34,524	-2,114
TD Garden – Causeway at Portal Park #1	17,327	19,098	-1,771
Rowes Wharf – Atlantic Ave	25,880	27,596	-1,716
B.U. Central – 725 Comm. Ave.	16,056	17,646	-1,590
Post Office Square	22,886	24,467	-1,581
Congress / Sleeper	19,026	20,467	-1,441
Davis Square	7,959	9,250	-1,291
Agganis Arena – 925 Comm Ave.	17,005	18,292	-1,287
Harvard Kennedy School at Bennett St / Eliot St	10,194	11,400	-1,206

6.3 Visualization: Net Inflow vs Outflow

[19]:
```
df_top_abs = pd.concat([top_sources, top_sinks], axis=0)
df_top_abs = df_top_abs.reindex(df_top_abs["net_flow"].abs().
 ↪sort_values(ascending=False).index)

plt.figure(figsize=(10, 6))
plt.barh(df_top_abs["station_name"], df_top_abs["net_flow"])
plt.axvline(0, linewidth=1)  # zero line
plt.title("Top Stations by Net Flow (starts  ends)")
plt.xlabel("Net Flow (positive = source, negative = sink)")
plt.ylabel("Station")

m = df_top_abs["net_flow"].abs().max()
plt.xlim(-m * 1.1, m * 1.1)

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



Bars to the right are sources (more starts than ends), and bars to the left are sinks (more ends than starts); the largest imbalances indicate where rebalancing is most needed.

6.4 Interpretation

Several stations act as persistent **sources** while others are clear **sinks,** reflecting directional commuter flows into downtown in the morning and out in the evening. These patterns imply a need for targeted **bike rebalancing** at sink stations during peak periods to maintain availability.

7 User Type Comparison

Compare trip behavior between subscriber and casual users.

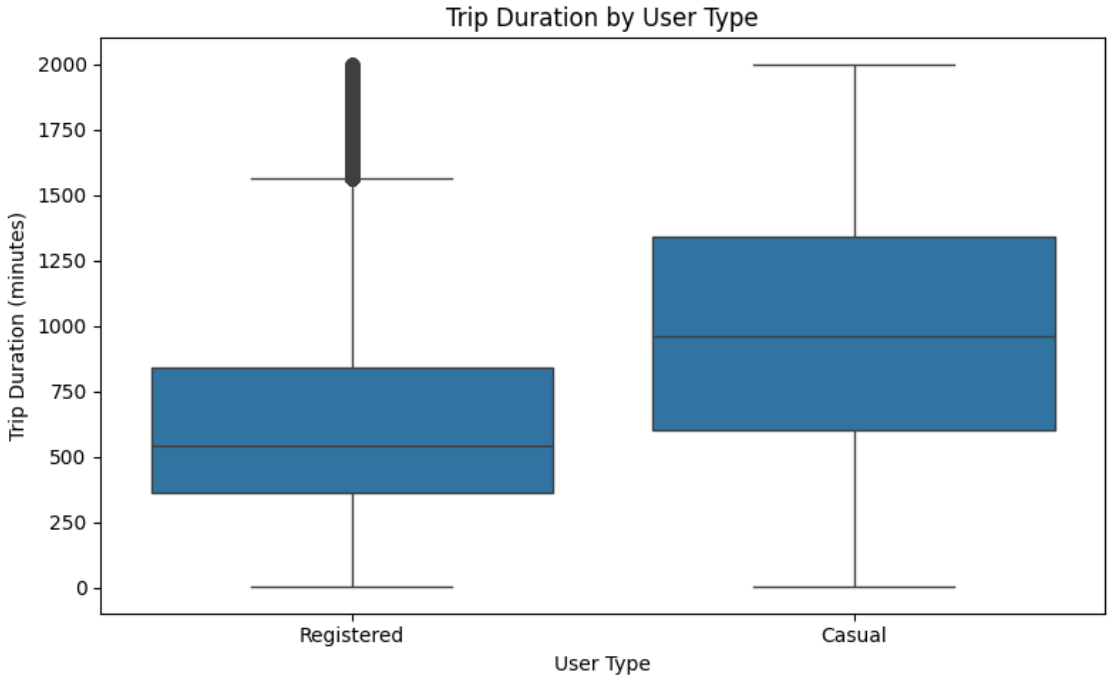
7.1 Query: Average Trip Duration by User Trip

[21]:
```
sql_user_type = """
SELECT
    sub_type AS user_type,
    COUNT(*) AS total_trips,
    ROUND(AVG(duration), 2) AS avg_duration,
    ROUND(MIN(duration), 2) AS min_duration,
    ROUND(MAX(duration), 2) AS max_duration
FROM trips
WHERE duration > 0  AND sub_type IS NOT NULL
GROUP BY sub_type
ORDER BY avg_duration DESC;
"""

df_user_type = run_sql(sql_user_type)
df_user_type
```

User Type	Total Trips	Avg. Duration (sec)	Min Duration (sec)	Max Duration (sec)
Casual	464,804	1,519.66	3.0	9,999.0
Registered	1,100,808	659.64	1.0	9,995.0

7.2 Visualization: Trip Duration by User Type

[22]:
```
import seaborn as sns

sql_user_durations = """
SELECT sub_type AS user_type, duration
FROM trips
WHERE duration > 0 AND duration < 2000 AND sub_type IS NOT NULL;
"""
df_durations = run_sql(sql_user_durations)

plt.figure(figsize=(8,5))
sns.boxplot(x="user_type", y="duration", data=df_durations)
plt.title("Trip Duration by User Type")
plt.xlabel("User Type")
plt.ylabel("Trip Duration (minutes)")
plt.tight_layout()
plt.show()
```



Casual riders show higher medians and wider variance in trip duration, while Registered riders take shorter, more consistent rides.

7.3 Interpretation

Casual users typically take **longer, leisure-oriented** trips, whereas Registered users make **shorter, routine commutes.** This split suggests different pricing/availability strategies (e.g., commuter-hour bike placement for Registered users, weekend coverage for Casual users). A simple follow-up (t-test or Mann-Whitney) could quantify the difference in medians.

8 Frequent Routes

Identify the most common station-to-station routes.

8.1 Query: Top Station-to-Station Routes

[24]:
```
sql_routes = """
SELECT
    s1.station AS start_station,
    s2.station AS end_station,
    COUNT(*) AS trip_count
FROM trips t
JOIN stations s1 ON t.start_station = s1.id
JOIN stations s2 ON t.end_station = s2.id
WHERE t.start_station IS NOT NULL AND t.end_station IS NOT NULL
```

```
GROUP BY s1.station, s2.station
ORDER BY trip_count DESC
LIMIT 10;
"""

df_routes = run_sql(sql_routes)
df_routes
```

Start Station	End Station	Trip Count
Beacon St / Mass Ave	MIT at Mass Ave / Amherst St	5,185
MIT at Mass Ave / Amherst St	Beacon St / Mass Ave	4,993
Lewis Wharf – Atlantic Ave.	South Station – 700 Atlantic Ave.	4,380
South Station – 700 Atlantic Ave.	Lewis Wharf – Atlantic Ave.	4,202
The Esplanade – Beacon St. at Arlington St.	The Esplanade – Beacon St. at Arlington St.	3,064
South Station – 700 Atlantic Ave.	Rowes Wharf – Atlantic Ave.	3,059
Kenmore Sq / Comm Ave	MIT at Mass Ave / Amherst St	2,751
Charles Circle – Charles St. at Cambridge St.	Charles Circle – Charles St. at Cambridge St.	2,739
Aquarium Station – 200 Atlantic Ave.	South Station – 700 Atlantic Ave.	2,713
Boylston St. at Arlington St.	South Station – 700 Atlantic Ave.	2,704

Top 10 most frequently traveled station pairs, sorted by total trip count.

8.2 Interpretation

The most frequent routes **connect key commuter and transit hubs,** such as **Beacon St / Mass Ave to MIT at Mass Ave / Amherst St** and **South Station to Lewis Wharf.** These high-volume corridors indicate strong, consistent demand between university zones, downtown, and waterfront areas. The presence of several **bidirectional routes** suggests balanced traffic flows, reflecting a mix of commuting and leisure movement within central city zones.

9 Repeat Riders

Measure rider loyalty and activity frequency.

9.1 Query: Fraction of Repeat Riders

[26]:
```
sql_repeat = """
WITH daily_trips AS (
  SELECT
    sub_type AS user_type,
    zip_code AS user_id,
    DATE(start_date) AS trip_day,
    COUNT(*) AS trips_per_day
  FROM trips
  WHERE start_date IS NOT NULL AND zip_code IS NOT NULL
  GROUP BY user_type, user_id, trip_day
```

```
),
weekly_trips AS (
  SELECT
    sub_type AS user_type,
    zip_code AS user_id,
    STRFTIME('%Y-%W', start_date) AS trip_week,
    COUNT(*) AS trips_per_week
  FROM trips
  WHERE start_date IS NOT NULL AND zip_code IS NOT NULL
  GROUP BY user_type, user_id, trip_week
),
daily_summary AS (
  SELECT
    user_type,
    ROUND(100.0 * SUM(CASE WHEN trips_per_day > 1 THEN 1 ELSE 0 END) / COUNT(*),␣
 ↪2) AS pct_repeat_day
  FROM daily_trips
  GROUP BY user_type
),
weekly_summary AS (
  SELECT
    user_type,
    ROUND(100.0 * SUM(CASE WHEN trips_per_week > 1 THEN 1 ELSE 0 END) /␣
 ↪COUNT(*), 2) AS pct_repeat_week
  FROM weekly_trips
  GROUP BY user_type
)
SELECT
  d.user_type,
  d.pct_repeat_day,
  w.pct_repeat_week
FROM daily_summary d
JOIN weekly_summary w USING (user_type);
"""

df_repeat = run_sql(sql_repeat)
df_repeat
```

User Type	Repeat Trips (Same Day, %)	Repeat Trips (Same Week, %)
Casual	64.81	56.50
Registered	77.77	90.66

Percentage of riders taking multiple trips per day or week, by user type.

9.2 Interpretation

A large share of riders take multiple trips within the same day or week, especially **Registered users,** with **77.8% repeating daily** and **90.7% weekly.** This pattern reflects consistent, commuter-style usage. In contrast, **Casual users** show lower repeat rates **(64.8% daily and 56.5% weekly),** indicating more occasional or leisure-based riding behavior.

10 Trip Duration Outliers

Identify unusually long trips by examining the distribution.

10.1 Query: Identify 95th Percentile Duration

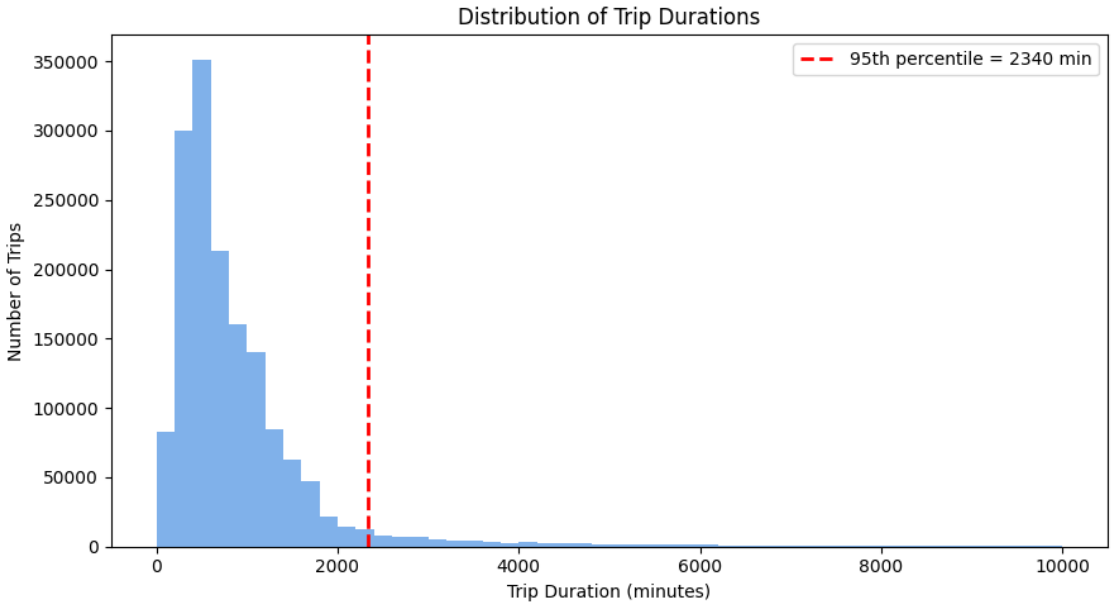
[28]:
```
sql_outliers = """
WITH duration_rank AS (
  SELECT
    duration,
    PERCENT_RANK() OVER (ORDER BY duration) AS pct_rank
  FROM trips
  WHERE duration > 0
)
SELECT
  ROUND(AVG(duration), 2) AS avg_duration,
  ROUND(MAX(duration), 2) AS max_duration,
  ROUND((SELECT duration FROM duration_rank WHERE pct_rank <= 0.95 ORDER BY␣
 ↪duration DESC LIMIT 1), 2) AS p95_duration
FROM duration_rank;
"""

df_outliers = run_sql(sql_outliers)
df_outliers
```

Metric	Average Duration (sec)	Max Duration (sec)	95th Percentile (sec)
Value	914.97	9,999.0	2,340.0

10.2 Visualization: Trip Durations with 95th Percentile Line

[29]:
```
p95 = df_outliers["p95_duration"].iloc[0]

sql_durations = "SELECT duration FROM trips WHERE duration > 0 AND duration <␣
 ↪10000;"
df_durations = run_sql(sql_durations)

plt.figure(figsize=(9,5))
plt.hist(df_durations["duration"], bins=50, color="#4a90e2", alpha=0.7)
```

```
plt.axvline(p95, color="red", linestyle="--", linewidth=2, label=f"95th␣
 ↪percentile = {p95:.0f} min")
plt.title("Distribution of Trip Durations")
plt.xlabel("Trip Duration (minutes)")
plt.ylabel("Number of Trips")
plt.legend()
plt.tight_layout()
plt.show()
```



Distribution of trip durations, showing a long right tail with the 95th percentile marked at 2,340 minutes.

10.3 Interpretation

Most trips are relatively short, with an average of **≈915 minutes,** but a small fraction extend far beyond this range, forming a pronounced **right-skewed tail.** These unusually long durations likely represent **leisure rides, stalled trips, or data entry errors,** and may require additional filtering or treatment in future analyses.

11 Conclusion

This analysis reveals distinct ridership patterns: rapid early growth, strong weekday commuter peaks, and clear seasonal variation in trip duration. Central stations such as South Station and Boston Public Library dominate both trip starts and ends, with directional imbalances highlighting the need for periodic bike rebalancing. Registered users show higher activity and repeat rates, reflecting routine commuting, while casual users take fewer but longer leisure rides.

Future analysis could incorporate weather, time of day, and demographic data to build predictive models of demand and improve bike availability across the network.