Metrocar Funnel Analysis

This notebook will further explore the Metrocar customer/ride data and prepare it for use in Tableau

In [1]:
```
!pip install sqlalchemy psycopg2 pandas
```

```
Requirement already satisfied: sqlalchemy in c:\users\bhaze\anaconda3\envs\mastersch
ool_metrocar\lib\site-packages (2.0.22)
Requirement already satisfied: psycopg2 in c:\users\bhaze\anaconda3\envs\masterschoo
l_metrocar\lib\site-packages (2.9.9)
Requirement already satisfied: pandas in c:\users\bhaze\appdata\roaming\python\pytho
n310\site-packages (2.1.1)
Requirement already satisfied: typing-extensions>=4.2.0 in c:\users\bhaze\anaconda3
\envs\masterschool_metrocar\lib\site-packages (from sqlalchemy) (4.8.0)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\bhaze\anaconda3\envs\mas
terschool_metrocar\lib\site-packages (from sqlalchemy) (3.0.0)
Requirement already satisfied: numpy>=1.22.4 in c:\users\bhaze\anaconda3\envs\master
school_metrocar\lib\site-packages (from pandas) (1.26.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\bhaze\anaconda3\en
vs\masterschool_metrocar\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\bhaze\anaconda3\envs\masters
chool_metrocar\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\bhaze\anaconda3\envs\maste
rschool_metrocar\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\bhaze\anaconda3\envs\masterschoo
l_metrocar\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

In [2]:
```
import pandas as pd
from sqlalchemy import create_engine

# Database connection string
db_url = "postgresql://Test:bQNxVzJL4g6u@ep-noisy-flower-846766-pooler.us-east-2.aw

# Create an engine instance
engine = create_engine(db_url)
```

We are first going to explore user segmentaion by platform and age range. After we will explore other attributes

In [3]:
```
# Platform Segmentation Query
query = """
WITH PlatformMapping AS (
    SELECT
        platform,
        CASE
            WHEN platform = 'android' THEN 'Android'
            WHEN platform = 'ios' THEN 'iOS'
```

```
            WHEN platform = 'web' THEN 'Web'
            ELSE platform
        END AS formatted_platform
    FROM (SELECT DISTINCT platform FROM app_downloads) AS platforms
),
FunnelSteps AS (
    -- App Download
    SELECT
        platform,
        'App Download' AS step,
        COUNT(DISTINCT app_download_key) AS count
    FROM app_downloads
    GROUP BY platform

    UNION ALL

    -- Signup
    SELECT
        ad.platform,
        'Signup' AS step,
        COUNT(DISTINCT s.user_id) AS count
    FROM signups s
    JOIN app_downloads ad ON s.session_id = ad.app_download_key
    GROUP BY ad.platform

    UNION ALL

    -- Request Ride
    SELECT
        ad.platform,
        'Request Ride' AS step,
        COUNT(DISTINCT rr.user_id) AS count
    FROM ride_requests rr
    JOIN signups s ON rr.user_id = s.user_id
    JOIN app_downloads ad ON s.session_id = ad.app_download_key
    GROUP BY ad.platform

    UNION ALL

    -- Driver Acceptance
    SELECT
        ad.platform,
        'Driver Acceptance' AS step,
        COUNT(DISTINCT rr.user_id) AS count
    FROM ride_requests rr
    JOIN signups s ON rr.user_id = s.user_id
    JOIN app_downloads ad ON s.session_id = ad.app_download_key
    WHERE rr.accept_ts IS NOT NULL
    GROUP BY ad.platform

    UNION ALL

    -- Ride Completed
    SELECT
        ad.platform,
        'Ride Completed' AS step,
```

```
        COUNT(DISTINCT rr.user_id) AS count
    FROM ride_requests rr
    JOIN signups s ON rr.user_id = s.user_id
    JOIN app_downloads ad ON s.session_id = ad.app_download_key
    WHERE rr.dropoff_ts IS NOT NULL
    GROUP BY ad.platform

    UNION ALL

    -- Payment
    SELECT
        ad.platform,
        'Payment' AS step,
        COUNT(DISTINCT rr.user_id) AS count
    FROM transactions t
    JOIN ride_requests rr ON t.ride_id = rr.ride_id
    JOIN signups s ON rr.user_id = s.user_id
    JOIN app_downloads ad ON s.session_id = ad.app_download_key
    WHERE t.charge_status = 'Approved'
    GROUP BY ad.platform

    UNION ALL

    -- Review
    SELECT
        ad.platform,
        'Review' AS step,
        COUNT(DISTINCT r.user_id) AS count
    FROM reviews r
    JOIN ride_requests rr ON r.ride_id = rr.ride_id
    JOIN signups s ON rr.user_id = s.user_id
    JOIN app_downloads ad ON s.session_id = ad.app_download_key
    GROUP BY ad.platform
),
OrderedSteps AS (
    SELECT
        fs.platform,
        fs.step,
        fs.count,
        CASE
            WHEN fs.step = 'App Download' THEN 1
            WHEN fs.step = 'Signup' THEN 2
            WHEN fs.step = 'Request Ride' THEN 3
            WHEN fs.step = 'Driver Acceptance' THEN 4
            WHEN fs.step = 'Ride Completed' THEN 5
            WHEN fs.step = 'Payment' THEN 6
            WHEN fs.step = 'Review' THEN 7
        END AS ordering
    FROM FunnelSteps fs
)
SELECT
    pm.formatted_platform AS platform,
    os.step,
    os.count,
    ROUND(100.0 * os.count / FIRST_VALUE(os.count) OVER (PARTITION BY pm.formatted_
    CASE
```

```
        WHEN os.step = 'App Download' THEN NULL
        ELSE ROUND(100.0 * os.count / LAG(os.count) OVER (PARTITION BY pm.formatted
    END AS percent_of_previous
FROM OrderedSteps os
JOIN PlatformMapping pm ON os.platform = pm.platform
ORDER BY pm.formatted_platform, os.ordering;

"""
```

In [4]:
```python
# Execute the query and store the result in a Pandas DataFrame


# Execute the query and store the result in a Pandas DataFrame
df_funnel_segmentation = pd.read_sql(query, engine)

# Display the DataFrame
df_funnel_segmentation
```

Out[4]:

	platform	step	count	percent_of_top	percent_of_previous
0	Android	App Download	6935	100.00	NaN
1	Android	Signup	5148	74.23	74.23
2	Android	Request Ride	3619	52.18	70.30
3	Android	Driver Acceptance	3580	51.62	98.92
4	Android	Ride Completed	1830	26.39	51.12
5	Android	Payment	1830	26.39	100.00
6	Android	Review	1273	18.36	69.56
7	Web	App Download	2383	100.00	NaN
8	Web	Signup	1747	73.31	73.31
9	Web	Request Ride	1237	51.91	70.81
10	Web	Driver Acceptance	1227	51.49	99.19
11	Web	Ride Completed	611	25.64	49.80
12	Web	Payment	611	25.64	100.00
13	Web	Review	424	17.79	69.39
14	iOS	App Download	14290	100.00	NaN
15	iOS	Signup	10728	75.07	75.07
16	iOS	Request Ride	7550	52.83	70.38
17	iOS	Driver Acceptance	7471	52.28	98.95
18	iOS	Ride Completed	3792	26.54	50.76
19	iOS	Payment	3792	26.54	100.00
20	iOS	Review	2651	18.55	69.91

Now we will create visualizations to explore the funnel by platform

In [5]:

```
!pip install plotly
!pip install kaleido
```

```
Requirement already satisfied: plotly in c:\users\bhaze\anaconda3\envs\masterschool_
metrocar\lib\site-packages (5.17.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\bhaze\anaconda3\envs\mast
erschool_metrocar\lib\site-packages (from plotly) (8.2.3)
Requirement already satisfied: packaging in c:\users\bhaze\anaconda3\envs\masterscho
ol_metrocar\lib\site-packages (from plotly) (23.2)
Requirement already satisfied: kaleido in c:\users\bhaze\anaconda3\envs\masterschool
_metrocar\lib\site-packages (0.2.1)
```

In [6]:
```
import plotly.graph_objects as go
import plotly.io as pio

platforms = ['iOS', 'Android', 'Web']
figs = []

for platform in platforms:
    # Filter data by platform
    platform_data = df_funnel_segmentation[df_funnel_segmentation['platform'] == pl

    # Create funnel chart for the platform
    fig = go.Figure(go.Funnel(
        y=platform_data['step'],
        x=platform_data['count'],
        textinfo='value+percent initial'
    ))
    fig.update_layout(title=f'{platform} Funnel')
    figs.append((platform, fig))

# Save the funnel charts as HTML files
for platform, fig in figs:
    fig.write_html(f'charts/{platform}_funnel.html')
    print(f"Chart for {platform} saved as {platform}_funnel.html")
```

```
Chart for iOS saved as iOS_funnel.html
Chart for Android saved as Android_funnel.html
Chart for Web saved as Web_funnel.html
```

In [67]:
```
# Aggregate data across all platforms
# Define the order of the funnel steps
funnel_step_order = [
    'App Download',
    'Signup',
    'Request Ride',
    'Driver Acceptance',
    'Ride Completed',
    'Payment',
    'Review'
]

# Aggregate data across all platforms
overall_funnel_data = df_funnel_segmentation.groupby('step')['count'].sum().reset_i

# Ensure the steps are in the correct order
overall_funnel_data['step'] = pd.Categorical(overall_funnel_data['step'], categorie
overall_funnel_data = overall_funnel_data.sort_values('step')

# Create the overall funnel chart
fig = go.Figure(go.Funnel(
    y=overall_funnel_data['step'],
    x=overall_funnel_data['count'],
    textinfo='value+percent initial'
))

fig.update_layout(title='Overall User Funnel')
```

```
# Save the funnel chart as an HTML file
fig.write_html('charts/overall_user_funnel.html')
print("Overall User Funnel chart saved as overall_user_funnel.html")
```

```
Overall User Funnel chart saved as overall_user_funnel.html
```

In [7]:

```
# Create a list to store traces
traces = []

# Loop through each platform to create traces
for platform in platforms:
    platform_data = df_funnel_segmentation[df_funnel_segmentation['platform'] == pl
    trace = go.Funnel(
        name=platform.capitalize(),
        y=platform_data['step'],
        x=platform_data['count'],
        textinfo='value+percent initial',
        legendgroup=platform
    )
    traces.append(trace)

# Create the combined funnel chart
fig = go.Figure(data=traces)

# Update layout for better visualization
fig.update_layout(
    title="Combined Funnel Chart for iOS, Android, and Web",
    funnelmode="group",  # This groups the funnels for comparison
    funnelgap=0.1  # Gap between funnels
)

# Display the combined funnel chart
fig.show()
fig.write_html(f'charts/combined_funnel.html')
```

In [8]:
```
# Create a list to store traces
traces = []

# Loop through each platform to create traces
for platform in platforms:
    platform_data = df_funnel_segmentation[df_funnel_segmentation['platform'] == pl
    trace = go.Bar(
        name=platform.capitalize(),
        x=platform_data['step'],
        y=platform_data['count'],
        legendgroup=platform
    )
    traces.append(trace)

# Create the grouped bar chart
fig = go.Figure(data=traces)

# Update layout for better visualization
fig.update_layout(
    title="Grouped Bar Chart for iOS, Android, and Web Funnel Steps",
    barmode="group",  # This groups the bars for comparison
    xaxis_title="Funnel Steps",
    yaxis_title="Count",
    bargap=0.1  # Gap between bars
```

```
)

# Display the grouped bar chart
fig.show()
fig.write_html(f'charts/platform_grouped_bar.html')
```

Segmentation by age range

We must remember that to have an age associated with it then the user had to sign up. This means downloads will equal sign ups

In [9]:
```
# Query to segment by age range
query = """
WITH FunnelSteps AS (
     -- App Download
    SELECT
        age_range,
        'App Download' AS step,
        COUNT(DISTINCT app_download_key) AS count
    FROM app_downloads ad
    LEFT JOIN signups s ON ad.app_download_key = s.session_id
    GROUP BY age_range
```

```
    UNION ALL

    -- Signup
    SELECT
        s.age_range,
        'Signup' AS step,
        COUNT(DISTINCT s.user_id) AS count
    FROM signups s
    GROUP BY s.age_range

    UNION ALL

    -- Request Ride
    SELECT
        s.age_range,
        'Request Ride' AS step,
        COUNT(DISTINCT rr.user_id) AS count
    FROM ride_requests rr
    JOIN signups s ON rr.user_id = s.user_id
    GROUP BY s.age_range

    UNION ALL

    -- Driver Acceptance
    SELECT
        s.age_range,
        'Driver Acceptance' AS step,
        COUNT(DISTINCT rr.user_id) AS count
    FROM ride_requests rr
    JOIN signups s ON rr.user_id = s.user_id
    WHERE rr.accept_ts IS NOT NULL
    GROUP BY s.age_range

    UNION ALL

    -- Ride Completed
    SELECT
        s.age_range,
        'Ride Completed' AS step,
        COUNT(DISTINCT rr.user_id) AS count
    FROM ride_requests rr
    JOIN signups s ON rr.user_id = s.user_id
    WHERE rr.dropoff_ts IS NOT NULL
    GROUP BY s.age_range

    UNION ALL

    -- Payment
    SELECT
        s.age_range,
        'Payment' AS step,
        COUNT(DISTINCT rr.user_id) AS count
    FROM transactions t
    JOIN ride_requests rr ON t.ride_id = rr.ride_id
    JOIN signups s ON rr.user_id = s.user_id
    WHERE t.charge_status = 'Approved'
```

```
    GROUP BY s.age_range

    UNION ALL

    -- Review
    SELECT
        s.age_range,
        'Review' AS step,
        COUNT(DISTINCT r.user_id) AS count
    FROM reviews r
    JOIN ride_requests rr ON r.ride_id = rr.ride_id
    JOIN signups s ON rr.user_id = s.user_id
    GROUP BY s.age_range
)

SELECT
    age_range,
    step,
    count,
    ROUND(100.0 * count / FIRST_VALUE(count) OVER (PARTITION BY age_range), 2) AS p
    CASE
        WHEN step = 'App Download' THEN NULL
        ELSE ROUND(100.0 * count / LAG(count) OVER (PARTITION BY age_range ORDER BY
    END AS percent_of_previous
FROM (
    SELECT
        age_range,
        step,
        count,
        CASE
            WHEN step = 'App Download' THEN 1
            WHEN step = 'Signup' THEN 2
            WHEN step = 'Request Ride' THEN 3
            WHEN step = 'Driver Acceptance' THEN 4
            WHEN step = 'Ride Completed' THEN 5
            WHEN step = 'Payment' THEN 6
            WHEN step = 'Review' THEN 7
        END AS ordering
    FROM FunnelSteps
) AS OrderedSteps
ORDER BY age_range, ordering;
"""
```

In [10]:
```
df_age_segmentation = pd.read_sql(query, engine)

df_age_segmentation
```

Out[10]:

	age_range	step	count	percent_of_top	percent_of_previous
0	18-24	App Download	1865	100.00	NaN
1	18-24	Signup	1865	100.00	100.00
2	18-24	Request Ride	1300	69.71	69.71
3	18-24	Driver Acceptance	1289	69.12	99.15
4	18-24	Ride Completed	670	35.92	51.98
5	18-24	Payment	670	35.92	100.00
6	18-24	Review	473	25.36	70.60
7	25-34	App Download	3447	100.00	NaN
8	25-34	Signup	3447	100.00	100.00
9	25-34	Request Ride	2425	70.35	70.35
10	25-34	Driver Acceptance	2393	69.42	98.68
11	25-34	Ride Completed	1227	35.60	51.27
12	25-34	Payment	1227	35.60	100.00
13	25-34	Review	842	24.43	68.62
14	35-44	App Download	5181	100.00	NaN
15	35-44	Signup	5181	100.00	100.00
16	35-44	Request Ride	3662	70.68	70.68
17	35-44	Driver Acceptance	3628	70.03	99.07
18	35-44	Ride Completed	1861	35.92	51.30
19	35-44	Payment	1861	35.92	100.00
20	35-44	Review	1332	25.71	71.57
21	45-54	App Download	1826	100.00	NaN
22	45-54	Signup	1826	100.00	100.00
23	45-54	Request Ride	1285	70.37	70.37
24	45-54	Driver Acceptance	1267	69.39	98.60
25	45-54	Ride Completed	630	34.50	49.72
26	45-54	Payment	630	34.50	100.00
27	45-54	Review	453	24.81	71.90
28	Unknown	App Download	5304	100.00	NaN
29	Unknown	Signup	5304	100.00	100.00

	age_range	step	count	percent_of_top	percent_of_previous
30	Unknown	Request Ride	3734	70.40	70.40
31	Unknown	Driver Acceptance	3701	69.78	99.12
32	Unknown	Ride Completed	1845	34.79	49.85
33	Unknown	Payment	1845	34.79	100.00
34	Unknown	Review	1248	23.53	67.64
35	None	App Download	5985	100.00	NaN

In [11]:
```
# Filter unique age ranges
age_ranges = df_age_segmentation['age_range'].unique()

# Create subplots
fig = go.Figure()

for age_range in age_ranges:
    df_filtered = df_age_segmentation[df_age_segmentation['age_range'] == age_range

    fig.add_trace(go.Funnel(
        name=age_range,
        y=df_filtered['step'],
        x=df_filtered['count'],
        textinfo="value+percent previous"
    ))

fig.update_layout(title="Combined Funnel Chart by Age Segmentation")
fig.show()
fig.write_html(f'charts/combined_funnel_by_age.html')
```

In [12]:
```
import os

# Ensure the directory for the charts exists
output_dir = 'charts'
os.makedirs(output_dir, exist_ok=True)

# Get unique age ranges from the dataframe
age_ranges = df_age_segmentation['age_range'].unique()
figs = []

for age_range in age_ranges:
    # Filter data by age_range
    age_data = df_age_segmentation[df_age_segmentation['age_range'] == age_range]

    # Create funnel chart for the age range
    fig = go.Figure(go.Funnel(
        y=age_data['step'],
        x=age_data['count'],
        textinfo='value+percent initial'
    ))
    fig.update_layout(title=f'{age_range} Age Range Funnel')
    figs.append((age_range, fig))

# Save the funnel charts as HTML files
```

```python
for age_range, fig in figs:
    file_path = os.path.join(output_dir, f'{age_range}_age_range_funnel.html')
    fig.write_html(file_path)
    print(f"Chart for {age_range} saved to {file_path}")
```

```
Chart for 18-24 saved to charts\18-24_age_range_funnel.html
Chart for 25-34 saved to charts\25-34_age_range_funnel.html
Chart for 35-44 saved to charts\35-44_age_range_funnel.html
Chart for 45-54 saved to charts\45-54_age_range_funnel.html
Chart for Unknown saved to charts\Unknown_age_range_funnel.html
Chart for None saved to charts\None_age_range_funnel.html
```

In [13]:

```python
# Get unique steps and age ranges from the dataframe
steps = df_age_segmentation['step'].unique()
age_ranges = df_age_segmentation['age_range'].unique()

# Create a grouped bar chart
fig = go.Figure()

# Add a bar for each age range
for age_range in age_ranges:
    age_data = df_age_segmentation[df_age_segmentation['age_range'] == age_range]
    fig.add_trace(go.Bar(
        x=age_data['step'],
        y=age_data['count'],
        name=age_range
    ))

# Update layout for grouped bar chart
fig.update_layout(
    barmode='group',
    title='Age Segmentation Funnel',
    xaxis_title='Funnel Step',
    yaxis_title='Count',
    xaxis={'categoryorder':'array', 'categoryarray': steps}
)

fig.show()
fig.write_html(f'charts/age_segmentation_funnel_bars.html')
```

Time Distibution of Rides for Surge Pricing strategy

As it is not specified I am making the assumption that the timestamps are in the local time for the ride.

In [14]:
```
# Query to get count of rides per hour of day
query = """
SELECT
    EXTRACT(HOUR FROM request_ts) AS hour_of_day,
    COUNT(ride_id) AS ride_count
FROM ride_requests
WHERE dropoff_ts IS NOT NULL
GROUP BY hour_of_day
ORDER BY hour_of_day;
"""
```

In [15]:
```
df_rides_per_hour = pd.read_sql(query, engine)

df_rides_per_hour
```

Out[15]:

	hour_of_day	ride_count
0	0.0	895
1	1.0	942
2	2.0	924
3	3.0	886
4	4.0	920
5	5.0	969
6	6.0	892
7	7.0	946
8	8.0	34973
9	9.0	34940
10	10.0	5173
11	11.0	4586
12	12.0	4579
13	13.0	4616
14	14.0	4632
15	15.0	4622
16	16.0	34001
17	17.0	33757
18	18.0	23263
19	19.0	23041
20	20.0	1297
21	21.0	952
22	22.0	936
23	23.0	910

In [16]:

```
# Convert hour_of_day to the desired format
df_rides_per_hour['hour_label'] = df_rides_per_hour['hour_of_day'].apply(
    lambda x: "12pm-1pm" if x == 12 else ("12am-1am" if x == 0 else (f"{int(x-12)}p
)

# Bar chart
fig = go.Figure(data=[go.Bar(
    x=df_rides_per_hour['hour_label'],
    y=df_rides_per_hour['ride_count'],
    marker_color='blue'
```

```
)])

# Layout
fig.update_layout(
    title='Distribution of Ride Requests by Hour of Day',
    xaxis_title='Hour of Day',
    yaxis_title='Number of Ride Requests',
    bargap=0.1
)

# Display the chart
fig.show()
fig.write_html(f'charts/distribution_hours_of_day.html')
```

Looking Deeper - Distribtution over weekday

In [17]:
```
# Query to get the count of request timestamp by day and hour
query = """
SELECT
    EXTRACT(DOW FROM request_ts) AS day_of_week,
    EXTRACT(HOUR FROM request_ts) AS hour_of_day,
    COUNT(ride_id) AS ride_count
FROM ride_requests
WHERE dropoff_ts IS NOT NULL
```

```
GROUP BY day_of_week, hour_of_day
ORDER BY day_of_week, hour_of_day;
"""
```

In [18]:
```
df_weekday_rides_per_hour = pd.read_sql(query, engine)

df_weekday_rides_per_hour
```

Out[18]:

	day_of_week	hour_of_day	ride_count
0	0.0	0.0	118
1	0.0	1.0	142
2	0.0	2.0	133
3	0.0	3.0	120
4	0.0	4.0	127
...	...	...	...
163	6.0	19.0	3330
164	6.0	20.0	206
165	6.0	21.0	138
166	6.0	22.0	119
167	6.0	23.0	125

168 rows × 3 columns

In [19]:
```
# Extract data
x = df_weekday_rides_per_hour['hour_of_day']
y = df_weekday_rides_per_hour['day_of_week']
z = df_weekday_rides_per_hour.pivot(index='day_of_week', columns='hour_of_day', val

# Create heatmap
fig = go.Figure(data=go.Heatmap(
    z=z.values,
    x=z.columns,
    y=['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'
    colorscale='Viridis',
    reversescale=True
))

# Update Layout
fig.update_layout(
    title='Distribution of Ride Requests Throughout the Week',
    xaxis_title='Hour of Day',
    yaxis_title='Day of Week'
)

# Show figure
```

```
fig.show()
fig.write_html(f'charts/rides_during_week.html')
```

In [20]:

```
# Grouping by hour_of_day and computing statistics
hourly_stats = df_weekday_rides_per_hour.groupby('hour_of_day')['ride_count'].agg([
hourly_stats.reset_index(inplace=True)
hourly_stats
```

Out[20]:

	hour_of_day	mean	median	std
0	0.0	127.857143	128.0	5.209881
1	1.0	134.571429	133.0	9.071147
2	2.0	132.000000	134.0	7.371115
3	3.0	126.571429	121.0	10.814452
4	4.0	131.428571	134.0	13.538305
5	5.0	138.428571	137.0	7.161404
6	6.0	127.428571	124.0	9.829499
7	7.0	135.142857	135.0	6.121780
8	8.0	4996.142857	5022.0	46.294091
9	9.0	4991.428571	4983.0	69.442679
10	10.0	739.000000	747.0	33.025243
11	11.0	655.142857	649.0	19.178361
12	12.0	654.142857	666.0	29.582814
13	13.0	659.428571	659.0	33.195668
14	14.0	661.714286	666.0	37.681624
15	15.0	660.285714	655.0	19.276188
16	16.0	4857.285714	4855.0	49.748415
17	17.0	4822.428571	4799.0	78.529946
18	18.0	3323.285714	3335.0	47.769287
19	19.0	3291.571429	3292.0	57.058032
20	20.0	185.285714	184.0	10.435744
21	21.0	136.000000	138.0	5.567764
22	22.0	133.714286	135.0	16.710419
23	23.0	130.000000	125.0	12.727922

In [21]:

```
# Calculate min and max values for each hour
min_values = df_weekday_rides_per_hour.groupby('hour_of_day')['ride_count'].min()
max_values = df_weekday_rides_per_hour.groupby('hour_of_day')['ride_count'].max()

# Add min and max values to the stats dataframe
hourly_stats['min'] = min_values.values
hourly_stats['max'] = max_values.values

hourly_stats
```

Out[21]:

	hour_of_day	mean	median	std	min	max
0	0.0	127.857143	128.0	5.209881	118	134
1	1.0	134.571429	133.0	9.071147	119	144
2	2.0	132.000000	134.0	7.371115	116	138
3	3.0	126.571429	121.0	10.814452	118	143
4	4.0	131.428571	134.0	13.538305	106	148
5	5.0	138.428571	137.0	7.161404	131	150
6	6.0	127.428571	124.0	9.829499	118	145
7	7.0	135.142857	135.0	6.121780	127	143
8	8.0	4996.142857	5022.0	46.294091	4931	5041
9	9.0	4991.428571	4983.0	69.442679	4900	5097
10	10.0	739.000000	747.0	33.025243	693	778
11	11.0	655.142857	649.0	19.178361	634	687
12	12.0	654.142857	666.0	29.582814	592	679
13	13.0	659.428571	659.0	33.195668	604	708
14	14.0	661.714286	666.0	37.681624	599	709
15	15.0	660.285714	655.0	19.276188	643	699
16	16.0	4857.285714	4855.0	49.748415	4783	4939
17	17.0	4822.428571	4799.0	78.529946	4738	4957
18	18.0	3323.285714	3335.0	47.769287	3234	3385
19	19.0	3291.571429	3292.0	57.058032	3195	3375
20	20.0	185.285714	184.0	10.435744	174	206
21	21.0	136.000000	138.0	5.567764	128	141
22	22.0	133.714286	135.0	16.710419	109	158
23	23.0	130.000000	125.0	12.727922	116	149

Each weekday is fairly consistent with demand

Hourly Conversion Rates

In [22]:
```
# Query to get hourly conversion rates - from ride request to completed

query = """
WITH HourlyRequests AS (
    SELECT
```

```
        EXTRACT(HOUR FROM request_ts) AS hour_of_day,
        COUNT(*) AS total_requests,
        COUNT(dropoff_ts) AS completed_rides
    FROM ride_requests
    GROUP BY EXTRACT(HOUR FROM request_ts)
)

SELECT
    hour_of_day,
    total_requests,
    completed_rides,
    (completed_rides::float / total_requests::float) * 100 AS conversion_rate
FROM HourlyRequests
ORDER BY hour_of_day;
"""
```

In [23]:
```
df_hourly_conversion = pd.read_sql(query, engine)

df_hourly_conversion
```

Out[23]:

	hour_of_day	total_requests	completed_rides	conversion_rate
0	0.0	1554	895	57.593308
1	1.0	1593	942	59.133710
2	2.0	1627	924	56.791641
3	3.0	1543	886	57.420609
4	4.0	1576	920	58.375635
5	5.0	1633	969	59.338641
6	6.0	1548	892	57.622739
7	7.0	1618	946	58.467244
8	8.0	60071	34973	58.219440
9	9.0	60210	34940	58.030228
10	10.0	9024	5173	57.324911
11	11.0	7928	4586	57.845610
12	12.0	7972	4579	57.438535
13	13.0	7960	4616	57.989950
14	14.0	7934	4632	58.381649
15	15.0	7957	4622	58.087219
16	16.0	58527	34001	58.094555
17	17.0	58176	33757	58.025646
18	18.0	40372	23263	57.621619
19	19.0	39495	23041	58.339030
20	20.0	2254	1297	57.542147
21	21.0	1701	952	55.967078
22	22.0	1624	936	57.635468
23	23.0	1580	910	57.594937

In [24]:

```
# Data
x = df_hourly_conversion['hour_of_day']
y = df_hourly_conversion['conversion_rate']

# Create bar chart
fig = go.Figure(data=[go.Bar(x=x, y=y)])

# Update Layout
fig.update_layout(
    title='Hourly Conversion Rates',
```

```
    xaxis_title='Hour of Day',
    yaxis_title='Conversion Rate (%)',
    xaxis=dict(tickvals=list(range(24)), ticktext=[f"{int(h)}:00-{int(h)+1}:00" for
    yaxis=dict(tickformat=".2f")
)

# Show plot
fig.show()
fig.write_html(f'charts/hourly_conversion_rates.html')
```

In [25]:

```
# Find the hour with the lowest conversion rate
lowest_hour = df_hourly_conversion['conversion_rate'].idxmin()
lowest_conversion_rate = df_hourly_conversion.loc[lowest_hour, 'conversion_rate']

# Find the hour with the highest conversion rate
highest_hour = df_hourly_conversion['conversion_rate'].idxmax()
highest_conversion_rate = df_hourly_conversion.loc[highest_hour, 'conversion_rate']

print(f"The hour with the lowest conversion rate is {lowest_hour}:00-{lowest_hour+1
print(f"The hour with the highest conversion rate is {highest_hour}:00-{highest_hou
```

```
The hour with the lowest conversion rate is 21:00-22:00 with a rate of 55.97%.
The hour with the highest conversion rate is 5:00-6:00 with a rate of 59.34%.
```

Duration of Rides

Analyzing the average duration of rides by time of day could help in understanding when longer or shorter trips are more common. This could be derived from the difference between pickup_ts and dropoff_ts.

In [26]:
```
# Query to get the average duration of rides by hour
query = """
SELECT
    EXTRACT(HOUR FROM pickup_ts) AS hour_of_day,
    AVG(EXTRACT(EPOCH FROM (dropoff_ts - pickup_ts))/60) AS avg_duration_minutes
FROM
    ride_requests
WHERE
    dropoff_ts IS NOT NULL
GROUP BY
    hour_of_day
ORDER BY
    hour_of_day;
"""
```

In [27]:
```
df_avg_ride_duration_by_hour = pd.read_sql(query, engine)

df_avg_ride_duration_by_hour
```

Out[27]:

	hour_of_day	avg_duration_minutes
0	0.0	53.444318
1	1.0	52.562173
2	2.0	52.412595
3	3.0	53.049774
4	4.0	53.010661
5	5.0	52.645570
6	6.0	52.252459
7	7.0	51.527808
8	8.0	52.566196
9	9.0	52.540994
10	10.0	52.568189
11	11.0	52.155715
12	12.0	52.743674
13	13.0	52.702855
14	14.0	52.619853
15	15.0	52.527437
16	16.0	52.649206
17	17.0	52.683568
18	18.0	52.728161
19	19.0	52.661266
20	20.0	52.451511
21	21.0	52.802617
22	22.0	52.435381
23	23.0	51.827133

Every hour is basically the same, no need to dive deeper at this time

Time to Driver Acceptance

By analyzing the time it takes for a driver to accept a ride request based on the time of day, we can identify if there are periods when drivers are less available or more hesitant to accept rides.

In [28]:
```
# Query to get average time to acceptance by hour

query = """
WITH TimeToAcceptance AS (
    SELECT
        EXTRACT(HOUR FROM request_ts) AS hour_of_day,
        AVG(EXTRACT(EPOCH FROM (accept_ts - request_ts))/60) AS avg_minutes_to_acce
    FROM ride_requests
    WHERE accept_ts IS NOT NULL
    GROUP BY hour_of_day
)

SELECT
    hour_of_day,
    ROUND(avg_minutes_to_accept, 2) AS avg_minutes_to_accept
FROM TimeToAcceptance
ORDER BY hour_of_day;
"""
```

In [29]:
```
df_time_to_acceptance = pd.read_sql(query, engine)

df_time_to_acceptance
```

Out[29]:

	hour_of_day	avg_minutes_to_accept
0	0.0	6.98
1	1.0	6.80
2	2.0	7.16
3	3.0	6.84
4	4.0	6.83
5	5.0	7.00
6	6.0	6.96
7	7.0	6.91
8	8.0	6.89
9	9.0	6.88
10	10.0	6.89
11	11.0	6.89
12	12.0	6.88
13	13.0	6.90
14	14.0	6.77
15	15.0	6.84
16	16.0	6.87
17	17.0	6.91
18	18.0	6.92
19	19.0	6.90
20	20.0	7.01
21	21.0	6.96
22	22.0	6.98
23	23.0	6.78

In [30]:

```
# Data
x = df_time_to_acceptance['hour_of_day']
y = df_time_to_acceptance['avg_minutes_to_accept']

# Create the plot
fig = go.Figure(data=go.Scatter(x=x, y=y, mode='lines+markers'))

# Update layout
fig.update_layout(
    title='Average Time to Driver Acceptance by Hour of Day',
```

```
    xaxis_title='Hour of Day',
    yaxis_title='Average Time to Acceptance (minutes)',
    xaxis=dict(tickvals=list(range(24)), ticktext=[f"{i}:00-{i+1}:00" for i in rang
    yaxis=dict(range=[5,8])  # Setting y-axis range to zoom in on the narrow range
)

# Show the plot
fig.show()
fig.write_html(f'charts/time_to_driver_acceptance.html')
```

It's basically a flat line, this likely isn't a huge impact. But, 7 minutes seems like a long time to wait for a ride to be accepted for me. Are there enough drivers???

Ride Ratings by Time of Day

Understanding if ride ratings (from the reviews table) vary by time of day could provide insights into potential issues during specific hours.

In [31]:
```
# Query to get average ratings by hour

query = """
WITH HourlyRatings AS (
```

```
    SELECT
        EXTRACT(HOUR FROM rr.pickup_ts) AS hour_of_day,
        r.rating
    FROM ride_requests rr
    JOIN reviews r ON rr.ride_id = r.ride_id
    WHERE rr.dropoff_ts IS NOT NULL
)

SELECT
    hour_of_day,
    AVG(rating) AS avg_rating
FROM HourlyRatings
GROUP BY hour_of_day
ORDER BY hour_of_day;
"""
```

In [32]:

```
df_hourly_ratings = pd.read_sql(query, engine)

df_hourly_ratings
```

Out[32]:

	hour_of_day	avg_rating
0	0.0	3.017974
1	1.0	2.970238
2	2.0	3.123824
3	3.0	3.122349
4	4.0	3.061747
5	5.0	3.081001
6	6.0	2.879070
7	7.0	3.021773
8	8.0	3.071952
9	9.0	3.052385
10	10.0	3.053894
11	11.0	3.070713
12	12.0	3.086105
13	13.0	3.071965
14	14.0	3.056850
15	15.0	3.036720
16	16.0	3.066845
17	17.0	3.071241
18	18.0	3.067068
19	19.0	3.081450
20	20.0	2.981083
21	21.0	3.020800
22	22.0	3.135747
23	23.0	3.166405

In [33]:

```
# Data
x = df_hourly_ratings['hour_of_day']
y = df_hourly_ratings['avg_rating']

# Create the plot
fig = go.Figure(data=go.Scatter(x=x, y=y, mode='lines+markers'))

# Update layout
fig.update_layout(
    title='Average Rating by Hour of Day',
```

```
    xaxis_title='Hour of Day',
    yaxis_title='Average Rating',
    xaxis=dict(tickvals=list(range(24)), ticktext=[f"{i}:00-{i+1}:00" for i in rang
    yaxis=dict(range=[2.5,3.5])  # Setting y-axis range to zoom in on the narrow ra
)

# Show the plot
fig.show()
fig.write_html(f'charts/ratings_by_hour_of_day.html')
```

With the average rating being in such a narrow band hovering around 3 it might be worth investigating why. This might involve Word Frequency analysys, Sentiment analysis, Topic Modeling, n-gram analysys, or in the end, a manual review.

In [34]:
```
# Query to fetch all free_responses

query = """
SELECT review_id, rating, review
FROM reviews
WHERE review_id IS NOT NULL;
"""
```

In [35]:
```
df_reviews = pd.read_sql(query, engine)

df_reviews
```

Out[35]:

	review_id	rating	review
0	50000	1	Horrible service. The driver was reckless and ...
1	50001	5	Metrocar's customer service is top-notch. I ha...
2	50002	5	Metrocar never disappoints. Whether it's a sho...
3	50003	5	Metrocar never disappoints. Whether it's a sho...
4	50004	1	Terrible experience with Metrocar. The driver ...
...	...	...	...
156206	206206	3	Okay service, but the fare was higher than exp...
156207	206207	4	The driver was friendly and the car was comfor...
156208	206208	5	Great service! The driver arrived promptly and...
156209	206209	1	Extremely disappointed. The driver was rude an...
156210	206210	1	Terrible experience with Metrocar. The driver ...

156211 rows × 3 columns

Before we can analyze the text data, we need to preprocess it. This involves:

Removing any special characters and numbers. Converting the text to lowercase. Tokenizing the text (splitting it into individual words or tokens). Removing common words (stopwords) that don't add much meaning, like "and", "the", "is", etc.

In [36]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\bhaze\anaconda3\envs\masterschool_me
trocar\lib\site-packages (3.8.1)
Requirement already satisfied: click in c:\users\bhaze\anaconda3\envs\masterschool_m
etrocar\lib\site-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in c:\users\bhaze\appdata\roaming\python\pytho
n310\site-packages (from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in c:\users\bhaze\anaconda3\envs\mast
erschool_metrocar\lib\site-packages (from nltk) (2023.10.3)
Requirement already satisfied: tqdm in c:\users\bhaze\anaconda3\envs\masterschool_me
trocar\lib\site-packages (from nltk) (4.66.1)
Requirement already satisfied: colorama in c:\users\bhaze\anaconda3\envs\masterschoo
l_metrocar\lib\site-packages (from click->nltk) (0.4.6)
```

In [37]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download the stopwords from nltk
```

```
nltk.download('stopwords')
nltk.download('punkt')

# Define a function to preprocess the text
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove special characters and numbers
    text = ''.join([char for char in text if char.isalpha() or char.isspace()])
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stopwords
    tokens = [token for token in tokens if token not in stopwords.words('english')]
    return ' '.join(tokens)

# File path for the processed data
processed_data_file = 'processed_reviews.csv'

# Check if the processed data file exists
if os.path.exists(processed_data_file):
    # If the file exists, load the processed data
    df_reviews = pd.read_csv(processed_data_file)
    print("Loaded processed data from file.")
else:
    # If the file does not exist, process the data and save it to a file
    df_reviews['processed_review'] = df_reviews['review'].apply(preprocess_text)
    df_reviews.to_csv(processed_data_file, index=False)
    print("Processed data and saved to file.")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\bhaze\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\bhaze\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Loaded processed data from file.
```

In [38]:
```
df_reviews
```

Out[38]:

	review_id	rating	review	processed_review	sentiment	dominant_topic	sen
0	50000	1	Horrible service. The driver was reckless and ...	horrible service driver reckless drove well sp...	negative	3	
1	50001	5	Metrocar's customer service is top-notch. I ha...	metrocars customer service topnotch issue fare...	positive	3	
2	50002	5	Metrocar never disappoints. Whether it's a sho...	metrocar never disappoints whether short trip ...	positive	4	
3	50003	5	Metrocar never disappoints. Whether it's a sho...	metrocar never disappoints whether short trip ...	positive	4	
4	50004	1	Terrible experience with Metrocar. The driver ...	terrible experience metrocar driver never show...	negative	0	
...	...	...	...	...	...	...	
156206	206206	3	Okay service, but the fare was higher than exp...	okay service fare higher expected distance tra...	positive	3	
156207	206207	4	The driver was friendly and the car was comfor...	driver friendly car comfortable however estima...	positive	3	
156208	206208	5	Great service! The driver arrived promptly and...	great service driver arrived promptly took des...	positive	3	
156209	206209	1	Extremely disappointed. The driver was rude an...	extremely disappointed driver rude unprofessio...	negative	3	
156210	206210	1	Terrible experience with	terrible experience metrocar driver never show...	negative	0	

review_id	rating	review	processed_review	sentiment	dominant_topic	sen
		Metrocar. The driver ...				

156211 rows × 7 columns

Given that the reviews are free-text responses, one common approach to analyze such data is to use topic modeling. Topic modeling can help identify common themes or topics present in the reviews. One popular method for topic modeling is Latent Dirichlet Allocation (LDA).

Here's a brief overview of the steps we'll take:

Vectorization: Convert the processed reviews into a matrix of token counts using CountVectorizer. LDA Model: Apply the LDA model to identify topics. Visualize Topics: Use pyLDAvis to visualize the topics and their relevance.

In [39]:
```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\bhaze\anaconda3\envs\masters
chool_metrocar\lib\site-packages (1.3.1)
Requirement already satisfied: numpy<2.0,>=1.17.3 in c:\users\bhaze\anaconda3\envs\m
asterschool_metrocar\lib\site-packages (from scikit-learn) (1.26.0)
Requirement already satisfied: scipy>=1.5.0 in c:\users\bhaze\anaconda3\envs\masters
chool_metrocar\lib\site-packages (from scikit-learn) (1.11.3)
Requirement already satisfied: joblib>=1.1.1 in c:\users\bhaze\appdata\roaming\pytho
n\python310\site-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\bhaze\anaconda3\envs
\masterschool_metrocar\lib\site-packages (from scikit-learn) (3.2.0)
```

In [41]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
import numpy as np

# File path for the processed data
processed_data_file = 'processed_reviews.csv'

# Check if the processed data file exists
if os.path.exists(processed_data_file):
    # If the file exists, load the processed data
    df_reviews = pd.read_csv(processed_data_file)
    print("Loaded processed data from file.")
else:
    print("Processed data file does not exist. Please run the previous cell to gene

# Ensure the 'processed_review' column exists
if 'processed_review' in df_reviews.columns:
    # Initialize a CountVectorizer
    vectorizer = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')

    # Fit and transform the processed reviews
    dtm = vectorizer.fit_transform(df_reviews['processed_review'])
```

```
    # Number of topics we want to extract
    n_topics = 5

    # Initialize LDA model
    lda_model = LatentDirichletAllocation(n_components=n_topics, random_state=42)

    # Fit the model
    lda_model.fit(dtm)

    # Compute the required data for pyLDAvis.prepare()

    # 1. Document-Topic Distributions
    doc_topic_dists = lda_model.transform(dtm)

    # 2. Document Lengths
    doc_lengths = [len(doc.split()) for doc in df_reviews['processed_review']]

    # 3. Vocabulary
    vocab = vectorizer.get_feature_names_out()

    # 4. Term Frequency
    term_frequency = np.sum(dtm.toarray(), axis=0)

    print("LDA model fitted and data prepared for visualization.")
else:
    print("Processed reviews are not available. Please check the 'processed_reviews
```

```
Loaded processed data from file.
LDA model fitted and data prepared for visualization.
```

In [42]:

```
!pip install pyLDAvis
```

```
Requirement already satisfied: pyLDAvis in c:\users\bhaze\anaconda3\envs\masterschoo
l_metrocar\lib\site-packages (3.4.1)
Requirement already satisfied: numpy>=1.24.2 in c:\users\bhaze\anaconda3\envs\master
school_metrocar\lib\site-packages (from pyLDAvis) (1.26.0)
Requirement already satisfied: scipy in c:\users\bhaze\anaconda3\envs\masterschool_m
etrocar\lib\site-packages (from pyLDAvis) (1.11.3)
Requirement already satisfied: pandas>=2.0.0 in c:\users\bhaze\appdata\roaming\pytho
n\python310\site-packages (from pyLDAvis) (2.1.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\bhaze\appdata\roaming\pytho
n\python310\site-packages (from pyLDAvis) (1.3.2)
Requirement already satisfied: jinja2 in c:\users\bhaze\anaconda3\envs\masterschool_
metrocar\lib\site-packages (from pyLDAvis) (3.1.2)
Requirement already satisfied: numexpr in c:\users\bhaze\anaconda3\envs\masterschool
_metrocar\lib\site-packages (from pyLDAvis) (2.8.7)
Requirement already satisfied: funcy in c:\users\bhaze\anaconda3\envs\masterschool_m
etrocar\lib\site-packages (from pyLDAvis) (2.0)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\users\bhaze\anaconda3\envs
\masterschool_metrocar\lib\site-packages (from pyLDAvis) (1.3.1)
Requirement already satisfied: gensim in c:\users\bhaze\anaconda3\envs\masterschool_
metrocar\lib\site-packages (from pyLDAvis) (4.3.2)
Requirement already satisfied: setuptools in c:\users\bhaze\anaconda3\envs\mastersch
ool_metrocar\lib\site-packages (from pyLDAvis) (68.0.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\bhaze\anaconda3\en
vs\masterschool_metrocar\lib\site-packages (from pandas>=2.0.0->pyLDAvis) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\bhaze\anaconda3\envs\masters
chool_metrocar\lib\site-packages (from pandas>=2.0.0->pyLDAvis) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\bhaze\anaconda3\envs\maste
rschool_metrocar\lib\site-packages (from pandas>=2.0.0->pyLDAvis) (2023.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\bhaze\anaconda3\envs
\masterschool_metrocar\lib\site-packages (from scikit-learn>=1.0.0->pyLDAvis) (3.2.
0)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\bhaze\anaconda3\envs\ma
sterschool_metrocar\lib\site-packages (from gensim->pyLDAvis) (6.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\bhaze\anaconda3\envs\mast
erschool_metrocar\lib\site-packages (from jinja2->pyLDAvis) (2.1.3)
Requirement already satisfied: six>=1.5 in c:\users\bhaze\anaconda3\envs\masterschoo
l_metrocar\lib\site-packages (from python-dateutil>=2.8.2->pandas>=2.0.0->pyLDAvis)
(1.16.0)
```

In [43]:

```python
from contextlib import contextmanager
import pyLDAvis

# Define a context manager to patch the DataFrame's drop method
@contextmanager
def patch_pandas_drop():
    original_drop = pd.DataFrame.drop

    def patched_drop(self, labels=None, axis=0, index=None, columns=None, level=Non
        if isinstance(labels, str) and axis == 1:
            columns = labels
            labels = None
        return original_drop(self, labels=labels, axis=axis, index=index, columns=c

    pd.DataFrame.drop = patched_drop
    try:
        yield
```

```
    finally:
        pd.DataFrame.drop = original_drop

# Use the context manager while calling pyLDAvis.prepare
with patch_pandas_drop():
    vis_data = pyLDAvis.prepare(topic_term_dists=lda_model.components_,
                                doc_topic_dists=doc_topic_dists,
                                doc_lengths=doc_lengths,
                                vocab=vocab,
                                term_frequency=term_frequency)

# Display the visualization
pyLDAvis.display(vis_data)

pyLDAvis.save_html(vis_data, 'lda_visualization.html')

print("LDA visualization saved as 'lda_visualization.html'")
```

```
LDA visualization saved as 'lda_visualization.html'
```

Some thoughts here:

1. Topic 1: "Route & Duration Concerns" - 43.8% This topic is significantly prevalent in your dataset, indicating that a large portion of your customers has concerns related to route, duration, and expected times of rides. This might involve issues like taking longer routes, unexpected delays, or discrepancies between expected and actual ride times.

Actionable Insights: Optimize Routes: Investigate if the routing algorithm can be optimized to choose quicker or more direct routes. Communication: Ensure clear communication regarding expected times and any delays. Pricing: Review the pricing strategy for longer routes or unexpected delays due to traffic. 2. Topic 2: "Driver Professionalism & Car Condition" - 16.6% This topic suggests that a notable portion of feedback revolves around the professionalism of drivers and the condition of the cars.

Actionable Insights: Training Programs: Implement or enhance driver training programs focusing on professionalism and customer service. Vehicle Maintenance: Ensure regular checks and maintenance of vehicles to uphold a standard of comfort and cleanliness. 3. Topic 3: "Reliability & Driver Friendliness" - 15.3% This topic encompasses the reliability of the service and the friendliness of the drivers, which is quite crucial for customer satisfaction.

Actionable Insights: App Stability: Address any technical issues with the app, especially those leading to crashes or inaccurate information. Driver Feedback: Encourage and reward drivers who receive positive feedback for friendliness and professionalism. 4. Topic 4: "Billing Issues & Service Response" - 15.2% Billing issues can be a significant pain point for customers and can impact their trust in the service.

Actionable Insights: Billing Transparency: Ensure that the billing process is transparent and accurate. Customer Support: Strengthen customer support to address and resolve billing issues promptly. 5. Topic 5: "Service Dissatisfaction & Seeking Alternatives" - 9.1% While smaller than the other topics, this segment of customers expressing dissatisfaction and considering alternatives is crucial to address to prevent churn.

Actionable Insights: Service Recovery: Identify and reach out to dissatisfied customers, offering apologies and possible compensations. Understand Pain Points: Dive deeper into reviews in this topic to understand specific pain points and address them.

Let's now look at a Sentiment Analysis

In [44]:
```
from nltk.sentiment import SentimentIntensityAnalyzer
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\bhaze\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[44]:
```
True
```

In [45]:
```
sia = SentimentIntensityAnalyzer()

# Define a function to get the sentiment
def get_sentiment(text):
    score = sia.polarity_scores(text)
    if score['compound'] >= 0.05:
        return 'positive'
    elif score['compound'] <= -0.05:
        return 'negative'
    else:
        return 'neutral'

# Apply the function to your reviews
df_reviews['sentiment'] = df_reviews['review'].apply(get_sentiment)
print(df_reviews[['review', 'sentiment']].head())
```

```
                                              review sentiment
0  Horrible service. The driver was reckless and ...  negative
1  Metrocar's customer service is top-notch. I ha...  positive
2  Metrocar never disappoints. Whether it's a sho...  positive
3  Metrocar never disappoints. Whether it's a sho...  positive
4  Terrible experience with Metrocar. The driver ...  negative
```

In [46]:
```
sentiment_counts = df_reviews['sentiment'].value_counts()

fig = go.Figure(go.Bar(
    x=sentiment_counts.index,
    y=sentiment_counts.values,
    text=sentiment_counts.values,
    textposition='auto'
))

fig.update_layout(title_text='Overall Sentiment Distribution', xaxis_title='Sentime
fig.show()
fig.write_html(f'charts/overall_sentiment_dist.html')
```

In [47]:
```
# Create a directory to save the plots if it doesn't exist
output_dir = 'charts'
os.makedirs(output_dir, exist_ok=True)

# Assign dominant topic to each review
df_reviews['dominant_topic'] = doc_topic_dists.argmax(axis=1)

topic_names = {
    0: "Route & Duration Concerns",
    1: "Driver Professionalism & Car Condition",
    2: "Reliability & Driver Friendliness",
    3: "Billing Issues & Service Response",
    4: "Service Dissatisfaction & Seeking Alternatives"
}

for topic_num, topic_name in topic_names.items():
    subset = df_reviews[df_reviews['dominant_topic'] == topic_num]
    sentiment_counts = subset['sentiment'].value_counts()

    fig = go.Figure(go.Bar(
        x=sentiment_counts.index,
        y=sentiment_counts.values,
        text=sentiment_counts.values,
        textposition='auto'
```

```
    ))

    fig.update_layout(title_text=f'Sentiment Distribution for {topic_name}', xaxis_

    # Save the plot to an HTML file
    file_path = os.path.join(output_dir, f'sentiment_distribution_topic_{topic_num}
    fig.write_html(file_path)
    print(f"Plot for {topic_name} saved to {file_path}")
```

```
Plot for Route & Duration Concerns saved to charts\sentiment_distribution_topic_0.ht
ml
Plot for Driver Professionalism & Car Condition saved to charts\sentiment_distributi
on_topic_1.html
Plot for Reliability & Driver Friendliness saved to charts\sentiment_distribution_to
pic_2.html
Plot for Billing Issues & Service Response saved to charts\sentiment_distribution_to
pic_3.html
Plot for Service Dissatisfaction & Seeking Alternatives saved to charts\sentiment_di
stribution_topic_4.html
```

In [48]:

```
# Assign sentiment scores
df_reviews['sentiment_score'] = df_reviews['sentiment'].map({'positive': 1, 'negati

# Compute average sentiment scores by topic
avg_sentiment_scores = df_reviews.groupby('dominant_topic')['sentiment_score'].mean

fig = go.Figure(go.Bar(
    x=[topic_names[i] for i in avg_sentiment_scores.index],
    y=avg_sentiment_scores.values,
    text=np.round(avg_sentiment_scores.values, 2),
    textposition='auto'
))

fig.update_layout(title_text='Average Sentiment Score by Topic', xaxis_title='Topic
fig.show()
fig.write_html(f'charts/average_sentiment_score_by_topic.html')
```

Now we will look at the regional distribution of rides

In [49]:
```
# Query to pull the ride_requests table into a DataFrame, we will start enriching d

query = """
SELECT ride_id, user_id, driver_id, request_ts, accept_ts, pickup_location, dropoff
FROM ride_requests;
"""
```

In [50]:
```
df_ride_requests = pd.read_sql(query, engine)

df_ride_requests
```

Out[50]:

	ride_id	user_id	driver_id	request_ts	accept_ts	pickup_location	dropoff_locatio
0	3000023	106891	105286.0	2021-05-27 19:38:00	2021-05-27 19:40:00	40.6851859 -73.99472165	40.8314265 -73.912711
1	3000024	116375	NaN	2021-12-05 00:02:00	NaT	40.81098464 -74.11502434	40.8098204 -73.803201
2	3000025	104571	109087.0	2021-07-09 09:06:00	2021-07-09 09:16:00	40.84414807 -73.84599412	40.866236 -73.977889
3	3000026	109497	NaN	2021-07-19 17:03:00	NaT	40.6581083 -73.90199317	40.782003 -74.105749
4	3000288	116687	NaN	2021-12-12 08:57:00	NaT	40.76639545 -73.877075	40.671571 -73.886817
...	...	...	...	...	...	...	.
385472	3385472	109022	112452.0	2021-10-31 17:19:00	2021-10-31 17:29:00	40.64896195 -73.90845782	40.761023 -74.045585
385473	3385473	111786	115682.0	2021-11-05 09:39:00	2021-11-05 09:49:00	40.77749575 -73.93980724	40.8977965 -74.038307
385474	3385474	109321	102701.0	2022-03-02 19:59:00	2022-03-02 20:06:00	40.79644372 -73.95786084	40.68271 -74.024715
385475	3385475	114256	104984.0	2022-02-24 09:40:00	2022-02-24 09:47:00	40.71281302 -73.83533833	40.702451 -74.098975
385476	3385476	112241	110489.0	2021-11-12 09:43:00	2021-11-12 09:54:00	40.73033251 -73.92682373	40.835939 -73.878502

385477 rows × 10 columns

In [51]:

```
import re

# Splitting the 'pickup_location' and 'destination_location' columns to extract lat
df_ride_requests['pickup_lat'], df_ride_requests['pickup_long'] = zip(*df_ride_requ
df_ride_requests['dropoff_lat'], df_ride_requests['dropoff_long'] = zip(*df_ride_re

# Rounding the coordinates to 2 decimal places
df_ride_requests['pickup_lat'] = df_ride_requests['pickup_lat'].round(2)
df_ride_requests['pickup_long'] = df_ride_requests['pickup_long'].round(2)
df_ride_requests['dropoff_lat'] = df_ride_requests['dropoff_lat'].round(2)
```

```
df_ride_requests['dropoff_long'] = df_ride_requests['dropoff_long'].round(2)

df_ride_requests.head()
```

Out[51]:

	ride_id	user_id	driver_id	request_ts	accept_ts	pickup_location	dropoff_location	picl
0	3000023	106891	105286.0	2021-05-27 19:38:00	2021-05-27 19:40:00	40.6851859 -73.99472165	40.83142658 -73.91271123	20 1!
1	3000024	116375	NaN	2021-12-05 00:02:00	NaT	40.81098464 -74.11502434	40.80982049 -73.80320195	
2	3000025	104571	109087.0	2021-07-09 09:06:00	2021-07-09 09:16:00	40.84414807 -73.84599412	40.8662361 -73.97788948	20 0!
3	3000026	109497	NaN	2021-07-19 17:03:00	NaT	40.6581083 -73.90199317	40.7820038 -74.1057497	
4	3000288	116687	NaN	2021-12-12 08:57:00	NaT	40.76639545 -73.877075	40.67157145 -73.88681784	

In [52]:

```
# Concatenate the lat and long columns for both pickup and dropoff
df_unique_coords = pd.concat([df_ride_requests[['pickup_lat', 'pickup_long']],
                              df_ride_requests[['dropoff_lat', 'dropoff_long']].ren

# Drop duplicates
df_unique_coords = df_unique_coords.drop_duplicates().reset_index(drop=True)

df_unique_coords
```

Out[52]:

	pickup_lat	pickup_long
0	40.69	-73.99
1	40.81	-74.12
2	40.84	-73.85
3	40.66	-73.90
4	40.77	-73.88
...	...	...
603	40.90	-74.01
604	40.84	-73.96
605	40.75	-73.96
606	40.73	-74.12
607	40.85	-74.01

608 rows × 2 columns

Use the API Ninja Reverse Geocoding API to get city names https://api-ninjas.com/api/reversegeocoding

Since all locations were found to be in the NYC area, I am commenting this section out to save time on execution of the notebook and to save API calls.

In [53]:

```
# import requests
# import configparser

# # Read API key from config.ini
# config = configparser.ConfigParser()
# config.read('config.ini')
# API_KEY = config['API_NINJAS']['API_KEY']

# # Define the API endpoint and headers
# API_ENDPOINT = "https://api.api-ninjas.com/v1/reversegeocoding"
# HEADERS = {
#     "X-Api-Key": API_KEY
# }

# # Function to fetch location data using the API
# def fetch_location_data(lat, lon):
#     params = {
#         "lat": lat,
#         "lon": lon
#     }
#     response = requests.get(API_ENDPOINT, params=params, headers=HEADERS)
#     data = response.json()
```

```
#     if data:
#         city = data[0].get("name", "")
#         country = data[0].get("country", "")
#         return city, country
#     else:
#         return None, None

# # Initialize an empty DataFrame for locations
# df_locations = pd.DataFrame(columns=["pickup_lat", "pickup_long", "city", "countr

# # Fetch location data for each unique coordinate and store in the df_locations Da
# for index, row in df_unique_coords.iterrows():
#     lat, lon = row["pickup_lat"], row["pickup_long"]
#     city, country = fetch_location_data(lat, lon)
#     temp_df = pd.DataFrame({
#         "pickup_lat": [lat],
#         "pickup_long": [lon],
#         "city": [city],
#         "country": [country]
#     })
#     df_locations = pd.concat([df_locations, temp_df], ignore_index=True)

# print(df_locations)
```

In [54]:

```
# # Create a scatter plot on a map
# fig = go.Figure(data=go.Scattergeo(
#         lon = df_locations['pickup_long'],
#         lat = df_locations['pickup_lat'],
#         text = df_locations['city'] + ', ' + df_locations['country'],
#         mode = 'markers',
#         marker = dict(
#             size = 8,
#             opacity = 0.6,
#             reversescale = True,
#             autocolorscale = False,
#             symbol = 'circle',
#             line = dict(
#                 width=1,
#                 color='rgba(102, 102, 102)'
#             ),
#             colorscale = 'Blues',
#             cmin = 0,
#             colorbar_title="Locations"
#         )))

# # Update the layout for a better view
# fig.update_layout(
#         title = 'Locations on Map',
#         geo = dict(
#             scope='world',
#             showland = True,
#             landcolor = "rgb(250, 250, 250)",
#             showocean = True,
#             oceancolor = "LightBlue",
#             showcountries=True,
#             showsubunits=True
```

```
#         )
#     )

# fig.show()
```

Everything is in the New York City area, so location won't really reveal anything.

Prep data for extraction into Tableau

In [55]:
```
df_reviews.to_csv('export/review_analysis.csv', index=False)
```

For best results in Tableau I will create one csv for all the data. For how I want to create the funnel, this will work best. I will use Tableau to add the review enrichments as the dataframe would be too large to do in this notebook - even with Google Colab

Extract Data

Beekeeper limits to 50,000 rows. We will have almost 400,000. This extract is to format the data for the Tableau Funnel dashboard.

We will export the Platform, but format the text properly We will determine the day of week and hour of day and export those We do not need location

In [56]:
```
import csv
from sqlalchemy import text


# Query to extract data to csv.

query = """
SELECT
    ad.app_download_key AS app_download_id,
    CASE
        WHEN ad.platform = 'android' THEN 'Android'
        WHEN ad.platform = 'ios' THEN 'iOS'
        WHEN ad.platform = 'web' THEN 'Web'
        ELSE ad.platform
    END AS platform,
    ad.download_ts,
    s.signup_ts,
    s.age_range,
    s.user_id,
    rr.ride_id,
    rr.driver_id,
    rr.request_ts,
    rr.accept_ts,
    rr.dropoff_ts,
    t.charge_status,
    r.review_id,
    EXTRACT(DOW FROM rr.request_ts) AS day_of_week_num,
    CASE
```

```
        WHEN EXTRACT(DOW FROM rr.request_ts) = 0 THEN 'Sunday'
        WHEN EXTRACT(DOW FROM rr.request_ts) = 1 THEN 'Monday'
        WHEN EXTRACT(DOW FROM rr.request_ts) = 2 THEN 'Tuesday'
        WHEN EXTRACT(DOW FROM rr.request_ts) = 3 THEN 'Wednesday'
        WHEN EXTRACT(DOW FROM rr.request_ts) = 4 THEN 'Thursday'
        WHEN EXTRACT(DOW FROM rr.request_ts) = 5 THEN 'Friday'
        ELSE 'Saturday'
    END AS day_of_week,
    CASE
        WHEN EXTRACT(HOUR FROM rr.request_ts) = 0 THEN '12am-1am'
        WHEN EXTRACT(HOUR FROM rr.request_ts) = 12 THEN '12pm-1pm'
        WHEN EXTRACT(HOUR FROM rr.request_ts) > 12 THEN CONCAT(EXTRACT(HOUR FROM rr
        ELSE CONCAT(EXTRACT(HOUR FROM rr.request_ts), 'am-', EXTRACT(HOUR FROM rr.r
    END AS hour_of_day
FROM
    app_downloads ad
FULL JOIN
    signups s ON ad.app_download_key = s.session_id
FULL JOIN
    ride_requests rr ON s.user_id = rr.user_id
FULL JOIN
    transactions t ON rr.ride_id = t.ride_id
FULL JOIN
    reviews r ON rr.ride_id = r.ride_id;
"""

with engine.connect() as connection:
    result = connection.execution_options(stream_results=True).execute(text(query))

    # Fetch the column names
    column_names = result.keys()

    with open ('export/full_data.csv', 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        csv_writer.writerow(column_names)  # write header
        for row in result:
            csv_writer.writerow(row)
```

Analysis to compare the time to cancel vs time to pickup

In [57]:
```
query = """
SELECT
    user_id,
    request_ts,
    cancel_ts,
    pickup_ts
FROM
    ride_requests;
"""
df_cancel_times = pd.read_sql_query(query, engine)
```

In [61]:
```
df_cancel_times['request_ts'] = pd.to_datetime(df_cancel_times['request_ts'])
df_cancel_times['cancel_ts'] = pd.to_datetime(df_cancel_times['cancel_ts'])
df_cancel_times['pickup_ts'] = pd.to_datetime(df_cancel_times['pickup_ts'])
```

```
df_cancel_times['ride_status'] = np.where(df_cancel_times['cancel_ts'].notna(), 'Ca
df_cancel_times['time_diff'] = np.where(df_cancel_times['ride_status'] == 'Canceled
                              (df_cancel_times['cancel_ts'] - df_cancel_times['request
                              (df_cancel_times['pickup_ts'] - df_cancel_times['request
```

In [65]:
```
average_time_to_cancel = df_cancel_times[df_cancel_times['ride_status'] == 'Cancele
average_time_to_pickup = df_cancel_times[df_cancel_times['ride_status'] == 'Picked

def convert_to_minutes_and_seconds(time_in_minutes):
    minutes = int(time_in_minutes)
    seconds = int((time_in_minutes - minutes) * 60)
    return f"{minutes} minutes and {seconds} seconds"

print(f"Average time to cancel: {convert_to_minutes_and_seconds(average_time_to_can
print(f"Average time to pickup: {convert_to_minutes_and_seconds(average_time_to_pic
```

```
Average time to cancel: 12 minutes and 33 seconds
Average time to pickup: 14 minutes and 29 seconds
```

In [ ]: