Visualization and Analysis of NYC Yellow Taxi Trip Data

PROBLEM STATEMENT:

Visualization and Analysis of NYC Yellow Taxi Trip Data (January to June 2023)

Develop an interactive Streamlit application to visualize and analyze NYC Yellow Taxi trip data from January 1, 2023, to June 30, 2023. The project aims to uncover temporal, geospatial, and economic patterns in taxi usage. Key objectives include identifying peak usage times, popular routes, and fare distributions. The application will help city planners, taxi companies, and researchers understand transportation dynamics, optimize operations, and make informed decisions. The tool will feature dynamic charts, maps, and filters for user-friendly data exploration. This project aims to enhance urban mobility insights and support strategic planning initiatives in New York City.

ABOUT THE DATASET:

1. **NYC Taxi Trip Data**: This dataset contains information about taxi trips within New York City, including details such as pickup and dropoff locations (identified by latitude and longitude), trip duration, passenger count, fare amount, payment type, and additional surcharges. The dataset may also include timestamps for pickup and dropoff times, allowing for analysis of temporal patterns in taxi activity.
2. **NYC Taxi Zone Data**: This dataset provides geographic information about taxi zones within NYC, including the boundaries and names of each zone, as well as their corresponding boroughs. This dataset is often used for spatial analysis, such as mapping taxi pickup and dropoff locations to specific zones and boroughs.
3. **Contigency table**:CSV file named contingency_df consist of contingency of all the LocationID where rows represent pickup location and columns represent dropoff location.

NYC TAXI DATASET

Field Name	Description
VendorID	A code indicating the TPEP provider that provided the record. **1= Creative Mobile Technologies,LLC; 2= VeriFone Inc.**
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged
RateCodeID	The final rate code in effect at the end of the trip. **1= Standard rate** **2=JFK** **3=Newark** **4=Nassau or Westchester** **5=Negotiated fare** **6=Group ride**
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server. **Y= store and forward trip** **N= not a store and forward trip**
Payment_type	A numeric code signifying how the passenger paid for the trip. **1= Credit card** **2= Cash** **3= No charge** **4= Dispute** **5= Unknown** **6= Voided trip**
Fare_amount	The time-and-distance fare calculated by the meter.
Extra	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
MTA_tax	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	$0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.

Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
Total_amount	The total amount charged to passengers. Does not include cash tips.
Congestion_Surcharge	Total amount collected in trip for NYS congestion surcharge.
Airport_fee	$1.25 for pick up only at LaGuardia and John F. Kennedy Airports

SHAPEFILE:

COLUMNS IN THE SHAPEFILE:

OBLECTID	Shape_Leng	Shape_Area	zone	LocationID	Borough	geometry

STREAMLIT:

Streamlit is a powerful Python library that enables the creation of interactive web applications for data science and machine learning projects. In the context of the Taxi Data Analysis Dashboard, Streamlit is utilized for building the user interface and enabling user interaction with the underlying data and visualizations. Here's how Streamlit is used in the dashboard:

1. **Streamlit Setup**: The first step involves setting up Streamlit in the project. This typically includes installing Streamlit via pip and importing it into the Python script.

2. **Dashboard Layout**: Streamlit provides various layout options for organizing content within the dashboard. This includes features like columns, containers, and sidebar menus. In the Taxi Data Analysis Dashboard, Streamlit's layout features are used to structure the different sections of the dashboard, such as the overview, zones, fare analysis, and trip planner.

3. **User Interaction**: Streamlit allows for easy integration of interactive elements such as dropdown menus, sliders, and buttons. These elements enable users to interact with the dashboard and customize their data exploration experience. For example, users can select different months, zones, or data categories using dropdown menus provided by Streamlit.

4. **Data Visualization**: Streamlit seamlessly integrates with popular visualization libraries like Plotly, Matplotlib, and Seaborn, allowing for the creation of interactive charts, plots, and maps directly within the dashboard. Streamlit's **st.plotly_chart()** function is particularly useful for embedding Plotly charts generated from data analysis.

5. **Dynamic Updates**: Streamlit enables dynamic updates of dashboard content based on user input or changes in data. This ensures that visualizations and statistics are updated in real-time as users interact with the dashboard. For example, selecting a different month from the dropdown menu triggers the dashboard to dynamically update the displayed statistics and visualizations for that month.

6. **Deployment**: Streamlit provides easy deployment options, allowing developers to share their interactive web applications with others. Once the dashboard is built and tested locally, it can be deployed to various platforms such as Streamlit Sharing, Heroku, or AWS for public access.

Overall, Streamlit streamlines the process of building interactive web applications for data analysis and visualization, making it accessible to data scientists and developers without extensive web development experience. Its simplicity and flexibility make it an ideal choice for rapidly prototyping and sharing data-driven applications like the Taxi Data Analysis Dashboard.

DEPENDENCIES

The Taxi Data Analysis Dashboard relies on the following Python libraries:

- **streamlit**: For building interactive web applications.
- **pandas**: For data manipulation and analysis.
- **plotly**: For interactive data visualization.
- **pydeck**: For creating 3D visualizations of geospatial data.
- **geopandas**: For working with geospatial data.
- **matplotlib** and **seaborn**: For creating static visualizations.

SECTIONS IN THE SIDE BAR:

1)Overview

2)Zones

3)Fare analysis

4)Trip planner

1)OVERVIEW:

The "Overview" section of the Taxi Data Analysis Dashboard serves as a comprehensive snapshot of key statistics and trends related to taxi trips for a selected month. Its primary purpose is to provide users with a high-level summary of important metrics and insights to quickly grasp the overall picture of taxi usage during the chosen period. Let's delve deeper into each component and aspect of this section:

Purpose:

The main objective of the "Overview" section is to offer users a concise yet informative summary of taxi trip data for the selected month. By presenting key statistics and visualizations, users can gain insights into the volume, distribution, and trends of taxi trips over time and across different days of the week.

Components:

1. **Summary Statistics:**
    - Total Trips: The aggregate number of taxi trips taken during the selected month.
    - Total Miles: The cumulative distance traveled by taxis in miles over the specified period.
    - Total Fare Collected: The overall revenue generated from taxi fares during the chosen month.
2. **Line Chart: Trips per Day:**
    - Visualizes the daily variation in taxi trip counts throughout the selected month.
    - Each data point on the line chart represents the number of taxi trips taken on a specific day.
    - Users can observe trends, spikes, or dips in taxi usage over time and identify any patterns or anomalies.
3. **Donut Chart: Distribution of Trips by Day of the Week:**
    - Illustrates the proportion of taxi trips distributed across different days of the week.

- Each segment of the donut chart represents a day of the week, with the size of each segment corresponding to the percentage of trips on that day.
- Users can quickly discern which days experience the highest and lowest taxi activity, providing insights into weekly demand patterns.

Usage:

- **Interactivity:** Users can interact with the section by selecting a specific month of interest from a dropdown menu or other input method.
- **Visualization Interpretation:** Users can interpret the visualizations presented in this section to understand the overall trends and patterns in taxi trip data.
- **Data Exploration:** Users can explore the data further by analyzing the variations in daily trip counts and the distribution of trips across different days of the week.

Customization:

- **Dynamic Updates:** The visualizations and statistics in the "Overview" section dynamically update based on the month selected by the user, allowing for real-time exploration of data from different time periods.
- **Color-Coded Metrics:** To enhance clarity and visual appeal, metrics and visual elements may be color-coded to highlight important insights or trends. For example, increases or decreases in key metrics may be represented by distinct colors to draw attention to significant changes.

Overall, the "Overview" section serves as a starting point for users to gain a comprehensive understanding of taxi trip data for a selected month, providing valuable insights into overall usage patterns and trends. It offers a user-friendly interface for data exploration and analysis, facilitating informed decision-making and further investigation into specific aspects of taxi usage.

ZONES SECTION:

Purpose:

The primary purpose of the "Zones" section is to provide users with a detailed understanding of taxi trip patterns and statistics within specific New York City (NYC) taxi zones. By focusing on individual zones, users can gain insights into localized trends and behaviors, which can be valuable for various analyses and decision-making processes. Whether it's understanding peak

hours, popular destinations, or demographic patterns, this section aims to offer comprehensive insights tailored to each zone.

Components

1. **Interactive Map**: This component serves as the entry point for exploration. Users can visually identify and select specific taxi zones within NYC. The interactive nature of the map enhances user engagement and facilitates seamless navigation.

2. **Hourly Pickup and Dropoff Patterns**: By visualizing the hourly variation in pickup and dropoff counts, users can discern temporal trends within each zone. This component helps identify peak hours of activity, potential congestion periods, and other time-sensitive patterns.

3. **Trips by Time of Day**: The donut chart illustrates the distribution of taxi trips across different times of the day. It provides a holistic view of the temporal distribution of taxi activity, allowing users to identify trends such as morning commutes, lunchtime rushes, and late-night surges.

4. **Trip Duration Distribution**: This histogram provides insights into the distribution of trip durations within the selected zone. Understanding trip duration patterns can help users identify trends related to trip length, potential factors influencing trip duration, and overall travel behavior.

5. **Passenger Count Distribution**: By visualizing the distribution of passenger counts per trip, users can gain insights into ride-sharing behavior, group travel dynamics, and vehicle occupancy patterns. This component complements the analysis by providing insights into the social aspects of taxi travel.

Usage

1. **Select Month and Zone**: Users can utilize dropdown menus to select a specific month for analysis and choose a taxi zone of interest. This flexibility allows users to focus their analysis on specific time periods and geographical areas within NYC.

2. **Explore Visualizations**: The interactive nature of the visualizations encourages exploration and discovery. Users can interact with each component to drill down into specific data points, compare trends across different zones, and uncover hidden insights.

3. **Analyze Zone-specific Data**: With real-time updates based on user selections, the dashboard enables users to dynamically analyze zone-specific data. Users can examine

metrics such as pickup/dropoff counts, trip durations, and passenger counts to gain a deeper understanding of each zone's characteristics and dynamics.

Customization

1. **Real-time Updates**: Visualizations and statistics are updated in real-time based on user selections, ensuring that users always have access to the latest insights and information.
2. **Interactive Map**: Users can interact with the map to select specific zones, zoom in/out for detailed exploration, and pan across different areas of NYC. This interactive feature enhances user engagement and facilitates intuitive navigation.

FARE ANALYSIS:

Purpose

The "Fare Analysis" section aims to provide users with insights into the fare distribution and factors influencing fare amounts for taxi trips within the selected timeframe. By analyzing fare-related metrics, users can understand fare trends, identify outliers, and explore potential correlations between fare amounts and trip attributes.

Components

1. **Fare Distribution Histogram**: This visualization presents the distribution of fare amounts for taxi trips within the selected timeframe. By examining the histogram, users can identify common fare ranges, outliers, and potential patterns in fare distribution.
2. **Fare by Trip Distance**: This scatter plot illustrates the relationship between fare amounts and trip distances. By plotting fare amounts against trip distances, users can assess whether fare amounts vary based on the length of the trip and identify any anomalies or unexpected trends.
3. **Fare by Time of Day**: This line chart displays the average fare amounts at different times of the day. Users can observe variations in fare amounts throughout the day, helping them identify peak fare periods, potential surges, and other temporal patterns.
4. **Fare by Passenger Count**: The bar chart visualizes the average fare amounts based on the number of passengers in a taxi trip. Users can analyze how fare amounts correlate with passenger counts, providing insights into fare structures, ride-sharing behavior, and fare dynamics based on group size.

Usage

1. **Select Month**: Users can choose a specific month for fare analysis using a dropdown menu. This allows users to focus their analysis on a particular timeframe and explore fare trends within that period.

2. **Interact with Visualizations**: Users can interact with each visualization to explore fare-related metrics in detail. They can zoom in/out, pan across different data points, and hover over elements to view specific fare amounts and related information.

3. **Analyze Fare Trends**: By examining the visualizations, users can identify patterns, anomalies, and trends in fare amounts. They can compare fare distributions, explore relationships between fare amounts and trip attributes, and derive insights into fare dynamics within the selected timeframe.

TRIP PLANNER SECTION:

The Trip Planner section of the dashboard is a powerful tool designed to help users explore and plan taxi trips within New York City. By leveraging historical taxi trip data, this section enables users to select a pickup location (taxi zone) and identify the top five most frequent drop-off locations from that pickup zone. Additionally, users can visualize the selected pickup zone and the top drop-off zones on an interactive map for better geographical context.

Purpose

The purpose of the Trip Planner section is to empower users to:

- Select a specific pickup location (taxi zone) within NYC.
- Explore the top five most frequent drop-off locations from the selected pickup zone.
- Visualize the spatial distribution of the selected pickup and top drop-off zones on an interactive map.

Components

1. **Pickup Location Selection**:
    - Users can choose a pickup location from a dropdown menu populated with the names of NYC taxi zones.

- This interactive component allows users to specify their starting point for the trip.

2. **Top Drop-off Locations Display**:
    - Upon selecting a pickup location, the dashboard presents a table listing the top five drop-off locations.
    - The table includes the drop-off zones and the frequency of trips ending in each zone, offering valuable insights into common destinations.
3. **Selected Zones Map**:
    - An interactive map is displayed alongside the table, highlighting the selected pickup zone in red and the top drop-off zones in sequential colors.
    - This map provides users with a visual representation of the selected zones, aiding in understanding the geographic distribution of common drop-off locations.

Usage

1. **Interacting with the Section**:
    - **Step 1**: Select a pickup location from the dropdown menu, choosing from the available NYC taxi zones.
    - **Step 2**: View the top five drop-off locations displayed in a table, which shows the zones and the number of trips ending in each zone.
    - **Step 3**: Examine the map to visualize the selected pickup zone (highlighted in red) and the top drop-off zones (highlighted in sequential colors).
2. **Interpreting the Visualizations**:
    - **Table**: Quickly grasp the most common destinations from the selected pickup location, aiding in understanding travel patterns.
    - **Map**: Visualize the spatial distribution of the selected zones, providing a geographic context that complements the data in the table.

Customization

- **Dynamic Updates**: Information dynamically updates based on the selected pickup location, ensuring real-time insights.

- **Color Coding**: Different sequential colors are used on the map to distinguish between the selected pickup zone and the top drop-off zones, enhancing clarity and visual appeal.

The Trip Planner section enhances the Taxi Data Analysis Dashboard by providing an interactive and user-friendly tool for exploring and planning taxi trips within NYC. By combining tabular data with interactive visualizations, users can gain valuable insights into travel patterns and make informed decisions when planning their trips.

CODE:

```
import streamlit as st

import pandas as pd

import plotly.express as px

from streamlit_option_menu import option_menu

import pydeck as pdk

import geopandas as gpd

import matplotlib.pyplot as plt

import seaborn as sns


#########################################

st.set_page_config(

    page_title="Dashboard",

    page_icon="🏂 ",

    layout="wide",

    initial_sidebar_state="expanded")

#########################################

MONTHS = {
```

```
    "January":1,
    "February":2,
    "March":3,
    "April":4,
    "May":5,
    "June":6
}

def read_df(month_number):
    return pd.read_csv(r"D:\dev\taxi\yellow_trip_data_2023-0{}.csv".format(month_number))

data = pd.DataFrame({
    'x': range(10),
    'y': range(10),
    'category': ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
})

##########################################
def format_number(num):
    neg = 0
    if num < 0:
        neg = 1
    if num < 0:
        num = -1 * num
```

```
    if num > 1000000:
        if not num % 1000000:
            if neg:
                return f'-{num // 1000000} M'
            else:
                return f'{num // 1000000} M'
        if neg:
            return f'-{round(num / 1000000, 1)} M'
        else:
            return f'{round(num / 1000000, 1)} M'
    if neg:
        return f'-{num // 1000} K'
    else:
        return f'{num // 1000} K'


 ##########################################


def overview_create_line_chart(df, month_number, line_color):
    df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])


    df_month = df[(df['tpep_pickup_datetime'] >= '2023-0{}-01'.format(month_number)) &
(df['tpep_pickup_datetime'] < '2023-0{}-01'.format(month_number+1))]
     df_month = df_month['tpep_pickup_datetime']
```

```
    trips_per_day = df_month.groupby(df_month.dt.date).size().reset_index(name='Trip Count')

    trips_per_day = trips_per_day.rename(columns={'tpep_pickup_datetime': 'Date'})

    fig = px.line(trips_per_day, x='Date', y='Trip Count', color_discrete_sequence=[line_color])
    fig.update_layout(width=650, height = 400)
    st.plotly_chart(fig)

def create_donut_chart_overview(df):
    df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])

    df['day_of_week'] = df['tpep_pickup_datetime'].dt.day_name()

    days_order = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']

    trip_distribution = df['day_of_week'].value_counts().reindex(days_order)

    day_of_week_to_explode = trip_distribution.idxmax()

    fig = px.pie(trip_distribution,
                 names=trip_distribution.index,
                 values=trip_distribution.values,
                 hole=0.6,
                 labels={'label': 'Day of Week', 'value': 'Trip Count'},
```

```
            )

    fig.update_traces(pull=[0.1 if day == day_of_week_to_explode else 0 for day in
trip_distribution.index],
                      textinfo='percent+label',
                      )

    fig.update_layout(showlegend=False, title="", width = 410, height = 410)

    st.plotly_chart(fig)

def plot_comparison_graphs(month1):
    month_no1 = MONTHS[month1]

    df = read_df(month_no1)

    df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])

    # Extract day of the week and hour of the day from the pickup datetime
    df['pickup_day_of_week'] = df['tpep_pickup_datetime'].dt.day_name()
    df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour

    counts_by_hour = df.groupby(['pickup_day_of_week',
'pickup_hour']).size().unstack(fill_value=0)
```

```
    counts_by_hour = counts_by_hour.reindex(columns=range(24), fill_value=0)


    days_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
    counts_by_hour = counts_by_hour.reindex(index=days_order, fill_value=0)


    plt.figure(figsize=(12, 8))


    for day in days_order:
        plt.plot(counts_by_hour.loc[day].index.astype(str), counts_by_hour.loc[day], label=day)


    plt.xlabel('Hour of Day')
    plt.ylabel('Count of Pickups')
    plt.xticks([i for i in range(24)], [i for i in range(24)])
    plt.legend()
    plt.grid(True)
    st.pyplot(plt)


    plt.figure(figsize=(10, 20))
    sns.heatmap(counts_by_hour.T, cmap='YlGnBu', annot = True)  # annot=True to display
counts, fmt='d' to format counts as integers
    plt.title('Pickup Counts by Day of Week and Hour of Day')
    plt.xlabel('Hour of Day')
    plt.ylabel('Day of Week')
```

```
    st.pyplot(plt)


def plot_ratecode_barchart(selected_month):
    df = read_df(MONTHS[selected_month])

    filtered_df = df[df['RatecodeID'].between(1, 6)]

    ratecode_counts = filtered_df['RatecodeID'].value_counts().reset_index()
    ratecode_counts.columns = ['RatecodeID', 'Count']

    x_axis_labels = ['Standard rate', 'JFK', 'Newark', 'Nassau or Westchester', 'Negotiated fare',
'Group ride']

    fig = px.bar(ratecode_counts, x='RatecodeID', y='Count',
                 color='Count', color_continuous_scale='viridis')
    fig.update_layout(xaxis_title='RatecodeID', yaxis_title='Count', yaxis_type='log')
    fig.update_xaxes(tickvals=ratecode_counts['RatecodeID'], ticktext=x_axis_labels)
    fig.update_traces(marker_line_color='black', marker_line_width=1,
text=ratecode_counts['Count'])
    fig.update_layout(bargap=0.2, bargroupgap=0.1)

    st.plotly_chart(fig)


def plot_pie_rates(selected_month):
```

```
    df = read_df(MONTHS[selected_month])

    components = ['fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
'improvement_surcharge', 'congestion_surcharge', 'airport_fee']

    # Calculate the sum of each component
    fare_breakdown = df[components].sum()

    # Plot interactive pie chart using Plotly Express
    fig = px.pie(names=fare_breakdown.index, values=fare_breakdown.values,
                 labels={'names': 'Components', 'values': 'Amount'},
                 hole=0.4)

    st.plotly_chart(fig)

def plot_daywise_fare(selected_month):
    df = read_df(MONTHS[selected_month])

    df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
    df['tpep_dropoff_datetime'] = pd.to_datetime(df['tpep_dropoff_datetime'])

    df['trip_duration'] = (df['tpep_dropoff_datetime'] -
df['tpep_pickup_datetime']).dt.total_seconds() / 60
```

```
    filtered_df = df[df['trip_distance'] > 1]

    filtered_df.loc[:, 'pickup_day_of_week'] = filtered_df['tpep_pickup_datetime'].dt.dayofweek

    filtered_df.loc[:, 'cost_per_mile'] = filtered_df['total_amount'] / filtered_df['trip_distance']

    median_cost_per_mile_per_day =
filtered_df.groupby('pickup_day_of_week')['cost_per_mile'].median()

    average_duration_per_day =
filtered_df.groupby('pickup_day_of_week')['trip_duration'].mean()

    days_of_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
    median_cost_per_mile_per_day.index = days_of_week
    average_duration_per_day.index = days_of_week

    min_duration = average_duration_per_day.min()
    max_duration = average_duration_per_day.max()
    scaled_sizes = 5 + ((average_duration_per_day - min_duration) / (max_duration -
min_duration)) * (20 - 5)

    fig = px.scatter(x=median_cost_per_mile_per_day.index,
y=median_cost_per_mile_per_day.values,
                     size=scaled_sizes, hover_name=average_duration_per_day.values)
```

```
    fig.update_traces(marker=dict(color='orange', symbol='circle'), line=dict(color='teal'), mode='markers+lines')


    hover_text = [f"Median Cost Per Mile: ${y:.2f}<br>Average Trip Duration: {avg_duration:.2f} minutes"
                  for y, avg_duration in zip(median_cost_per_mile_per_day.values, average_duration_per_day.values)]
    fig.update_traces(hovertext=hover_text, hoverinfo='text')


    fig.update_layout(xaxis_title='Day of the Week', yaxis_title='Median Cost Per Mile', showlegend=False)


    st.plotly_chart(fig)


def plot_histograms(selected_month):
    df = read_df(MONTHS[selected_month])


    filtered_df_total_amount = df[(df['total_amount'] > 0) & (df['total_amount'] < 200)]


    fig_total_amount = px.histogram(filtered_df_total_amount, x='total_amount',  nbins = 100)
    fig_total_amount.update_layout(xaxis_title='Total Amount', yaxis_title='Count')
    fig_total_amount.update_traces(marker=dict(color='#2eb853'))


    st.plotly_chart(fig_total_amount)
```

```
def show_overview():
    col = st.columns((1, 3.5, 2), gap='medium')

    line_color = "#20a829"
    with col[0]:
        st.markdown('#### Summary')

        months = MONTHS.keys()

        selected_month = st.selectbox('Select a Month', months)

        if MONTHS[selected_month] == 1:
            df = read_df(MONTHS[selected_month])

            total_trips = format_number(df.shape[0])
            total_miles = format_number(sum(df['trip_distance']))
            total_fare = format_number(sum(df['total_amount']))

            st.metric(label="Total Trips", value=total_trips, delta="")

            st.metric(label="Total Miles", value=total_miles)

            st.metric(label="Total Fare Collected", value="$" + total_fare)
        else:
```

```
        df_curr = read_df(MONTHS[selected_month])
        df_prev = read_df(MONTHS[selected_month] - 1)

        total_trips = format_number(df_curr.shape[0])
        total_miles = format_number(sum(df_curr['trip_distance']))
        total_fare = format_number(sum(df_curr['total_amount']))

        total_trips_delta = format_number(df_curr.shape[0] - df_prev.shape[0])
        total_miles_delta = format_number(sum(df_curr['trip_distance']) -
sum(df_prev['trip_distance']))
        total_fare_delta = format_number(sum(df_curr['total_amount']) -
sum(df_prev['total_amount']))

        if (df_curr.shape[0] - df_prev.shape[0]) < 0:
            line_color = "#f74f4f"

        st.metric(label="Total Trips", value=total_trips, delta=total_trips_delta)

        st.metric(label="Total Miles", value=total_miles, delta=total_miles_delta)

        st.metric(label="Total Fare Collected", value="$" + total_fare, delta=total_fare_delta)

    with col[1]:
        st.markdown('#### Trips Per Day in {} 2023'.format(selected_month))
```

```
        if MONTHS[selected_month] == 1:
            overview_create_line_chart(df, MONTHS[selected_month], line_color)
        else:
            overview_create_line_chart(df_curr, MONTHS[selected_month], line_color)



    with col[2]:
        st.markdown('#### Distribtion of Trips in {} 2023'.format(selected_month))
        if MONTHS[selected_month] == 1:
            create_donut_chart_overview(df)
        else:
            create_donut_chart_overview(df_curr)


def show_comparison():
    col = st.columns((0.5, 2, 2), gap='medium')


    with col[0]:
        months = MONTHS.keys()


        selected_comparison_month1 = st.selectbox('Select Month 1', months)
        selected_comparison_month2 = st.selectbox('Select Month 2', months)


    with col[1]:
        st.markdown('#### {} 2023'.format(selected_comparison_month1))
```

```
        plot_comparison_graphs(selected_comparison_month1)

    with col[2]:
        st.markdown('#### {} 2023'.format(selected_comparison_month2))
        plot_comparison_graphs(selected_comparison_month2)

def show_fare_analysis():
    col = st.columns((0.5, 2, 2), gap='medium')


    with col[0]:

        months = MONTHS.keys()
        selected_month = st.selectbox('Select Month', months)

    with col[1]:
        st.markdown("# Fare Analysis")
        st.markdown('#### Count Plot of RatecodeID')
        plot_ratecode_barchart(selected_month)

        st.markdown('#### Median Cost Per Mile')
        plot_daywise_fare(selected_month)

    with col[2]:
```

```
        for _ in range(5):
            st.markdown("")
        st.markdown("#### Fare Breakdown")
        plot_pie_rates(selected_month)


        st.markdown("#### Distribution of Total Amount")
        plot_histograms(selected_month)


def show_trip_planner():
    col = st.columns((0.5, 2, 2), gap='medium')


########################################
def create_heatmap(df):
    df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])


    # Extract day of the week and hour
    df['day_of_week'] = df['tpep_pickup_datetime'].dt.day_name()
    df['hour'] = df['tpep_pickup_datetime'].dt.hour


    # Calculate total passenger count for each combination of day of the week and hour
    heatmap_data = df.groupby(['day_of_week', 'hour'])['passenger_count'].sum().reset_index()


    # Pivot the dataframe to create the heatmap data
```

```
    heatmap_data_pivot = heatmap_data.pivot_table(index='day_of_week', columns='hour', values='passenger_count', aggfunc='sum')


    # Reorder the columns to start from Monday
    heatmap_data_pivot = heatmap_data_pivot.reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])


    # Create the heatmap
    plt.figure(figsize=(20, 6))
    sns.heatmap(heatmap_data_pivot, cmap='viridis', annot=True, fmt='g')
    plt.title('Heatmap of Total Passenger Count by Day of Week and Hour')
    plt.xlabel('Hour')
    plt.ylabel('Day of Week')
    st.pyplot(plt)
############################################################
def categorize_time_of_day(hour):
    if 0 <= hour < 6:
        return 'Midnight'
    elif 6 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 18:
        return 'Noon'
    else:
        return 'Evening'
```

```
def create_donut_chart_time_of_day(df):
    df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
    df['time_of_day'] = df['tpep_pickup_datetime'].dt.hour.apply(categorize_time_of_day)

    time_of_day_distribution = df['time_of_day'].value_counts()

    fig = px.pie(time_of_day_distribution,
                 names=time_of_day_distribution.index,
                 values=time_of_day_distribution.values,
                 hole=0.6,
                 labels={'label': 'Time of Day', 'value': 'Trip Count'},
                 title="Trips by Time of Day",
                 template="plotly_white",
                 color_discrete_sequence=px.colors.sequential.Viridis)

    fig.update_traces(textinfo='percent+label')
    fig.update_layout(showlegend=False, title_x=0.5)

    return fig

def show_zones_map():
    # Load the taxi zone dataset
    taxi_zones = gpd.read_file('D:/dev/taxinyc/taxi_zones.shp')
```

```
    # Calculate centroids
    taxi_zones['centroid'] = taxi_zones.geometry.centroid
    taxi_zones['lat'] = taxi_zones['centroid'].apply(lambda p: p.y)
    taxi_zones['lon'] = taxi_zones['centroid'].apply(lambda p: p.x)

    # Prepare a mapping DataFrame
    location_mapping = taxi_zones[['LocationID', 'lat', 'lon']]

    # Plot the taxi zones on a map
    st.markdown('# NYC Taxi Zones')

    col1, col2 = st.columns([1.2, 1])  # Make the first column wider

    with col1:
        for _ in range(2):
            st.markdown("")
        selected_month = st.selectbox("Select a month for zones map", list(MONTHS.keys()), key="zones_month")
        selected_zone_display = st.selectbox("Select a zone", taxi_zones['zone'])
        selected_zone_id = taxi_zones[taxi_zones['zone'] == selected_zone_display]['LocationID'].values[0]
        selected_zone_data = taxi_zones[taxi_zones['LocationID'] == selected_zone_id]
```

```
    # Load the taxi dataset
    df = read_df(MONTHS[selected_month])

    # Merge pickup coordinates
    df = df.merge(location_mapping, left_on='PULocationID', right_on='LocationID', suffixes=('', '_pickup'))
    df.rename(columns={'lat': 'pickup_latitude', 'lon': 'pickup_longitude'}, inplace=True)

    # Merge dropoff coordinates
    df = df.merge(location_mapping, left_on='DOLocationID', right_on='LocationID', suffixes=('', '_dropoff'))
    df.rename(columns={'lat': 'dropoff_latitude', 'lon': 'dropoff_longitude'}, inplace=True)

    # Filter the dataset based on the selected taxi zone
    df_filtered = df[(df['PULocationID'] == selected_zone_id) | (df['DOLocationID'] == selected_zone_id)]

    # Calculate total pickups and dropoffs
    total_pickups = df_filtered[df_filtered['PULocationID'] == selected_zone_id].shape[0]
    total_dropoffs = df_filtered[df_filtered['DOLocationID'] == selected_zone_id].shape[0]

    # Add total pickups and dropoffs to the DataFrame
    selected_zone_data['Total Pickups'] = total_pickups
    selected_zone_data['Total Dropoffs'] = total_dropoffs
```

```
    st.write(f"Selected Zone: {selected_zone_display}")

    st.dataframe(selected_zone_data[['LocationID', 'zone', 'borough', 'Total Pickups', 'Total Dropoffs']], width=2000)  # Adjust width here


    for _ in range(15):
        st.markdown("")


    # Hourly Pickup and Dropoff Patterns
    st.markdown('#### Hourly Pickup and Dropoff Patterns')
    df_filtered['pickup_hour'] = pd.to_datetime(df_filtered['tpep_pickup_datetime']).dt.hour
    df_filtered['dropoff_hour'] = pd.to_datetime(df_filtered['tpep_dropoff_datetime']).dt.hour

    hourly_pickup = df_filtered.groupby('pickup_hour').size().reset_index(name='Pickup Count')
    hourly_dropoff = df_filtered.groupby('dropoff_hour').size().reset_index(name='Dropoff Count')

    fig_hourly_pickup = px.line(hourly_pickup, x='pickup_hour', y='Pickup Count',
                                title='Hourly Pickup Count',
                                labels={'pickup_hour': 'Hour of Day', 'Pickup Count': 'Number of Pickups'},
                                template="plotly_white",
                                color_discrete_sequence=["#2ca02c"])
    fig_hourly_pickup.update_layout(title_x=0.5)
    st.plotly_chart(fig_hourly_pickup)
```

```
        st.markdown('#### Trips by Time of Day')

        fig_time_of_day = create_donut_chart_time_of_day(df_filtered)

        st.plotly_chart(fig_time_of_day)


        st.markdown('#### Trip Duration Distribution')

        df_filtered['trip_duration'] = (pd.to_datetime(df_filtered['tpep_dropoff_datetime']) - pd.to_datetime(df_filtered['tpep_pickup_datetime'])).dt.total_seconds() / 60

        fig3 = px.histogram(df_filtered[df_filtered['trip_duration'] < 200], x='trip_duration', nbins=50, title='Trip Duration Distribution (minutes)', template="plotly_white", color_discrete_sequence=["#d62728"])

        fig3.update_layout(title_x=0.5)

        st.plotly_chart(fig3)


        # Plot 6: Passenger Count Distribution

        st.markdown('#### Passenger Count Distribution')

        fig6 = px.histogram(df_filtered, x='passenger_count', nbins=6, title='Passenger Count Distribution', template="plotly_white", color_discrete_sequence=["#9467bd"])

        fig6.update_layout(title_x=0.5)

        st.plotly_chart(fig6)


        st.markdown('#### Passenger Count by Day of Week and Hour of Day')

        create_heatmap(df_filtered)


    with col2:
```

```
    plt.figure(figsize=(8,8))
    taxi_zones.plot(color='blue', edgecolor='gray', alpha=0.1)
    plt.scatter(selected_zone_data['lon'], selected_zone_data['lat'], color='red', s=10,
label='Selected Zone')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.title('NYC Taxi Zone Map')
    plt.legend()
    st.pyplot(plt)

    fig_hourly_dropoff = px.line(hourly_dropoff, x='dropoff_hour', y='Dropoff Count',
                                 title='Hourly Dropoff Count',
                                 labels={'dropoff_hour': 'Hour of Day', 'Dropoff Count': 'Number of
Dropoffs'},
                                 template="plotly_white",
                                 color_discrete_sequence=["#ff7f0e"])
    fig_hourly_dropoff.update_layout(title_x=0.5)

    for _ in range(3):
       st.markdown("")
    st.plotly_chart(fig_hourly_dropoff)

    st.markdown('#### Total Pick Up and Drop Offs')
```

```
# Plot 1: Total Pickups and Drop-offs per Zone
total_counts = pd.DataFrame({
    'Type': ['Pickups', 'Drop-offs'],
    'Count': [total_pickups, total_dropoffs]
})
fig1 = px.bar(total_counts, x='Type', y='Count', title='Total Pickups and Drop-offs', template="plotly_white", color_discrete_sequence=["#8c564b"])
fig1.update_layout(title_x=0.5)
st.plotly_chart(fig1)

# Plot 2: Trip Distance Distribution
st.markdown('#### Trip Distance Distribution')
fig2 = px.histogram(df_filtered, x='trip_distance', nbins=50, title='Trip Distance Distribution', template="plotly_white", color_discrete_sequence=["#e377c2"])
fig2.update_layout(title_x=0.5)
st.plotly_chart(fig2)

# Plot 4: Average Fare by Time of Day
st.markdown('#### Average Fare by Time of Day')
df_filtered['pickup_hour'] = pd.to_datetime(df_filtered['tpep_pickup_datetime']).dt.hour
avg_fare_time_of_day = df_filtered.groupby('pickup_hour')['total_amount'].mean().reset_index()
fig4 = px.line(avg_fare_time_of_day, x='pickup_hour', y='total_amount', title='Average Fare by Time of Day', template="plotly_white", color_discrete_sequence=["#7f7f7f"])
fig4.update_layout(title_x=0.5)
```

```
        st.plotly_chart(fig4)
##########################################################################
contingency_df=pd.read_csv(r"D:\dev\frequency_table.csv")
contingency_df.drop(columns=['Unnamed: 0'], inplace=True)
location_id_to_zone = taxi_zones.set_index('LocationID')['zone'].to_dict()


def trip_planner():
    st.markdown("# NYC Taxi Zones with Top Dropoff Locations")


    # Get the zone names for the select box
    zone_names = taxi_zones['zone'].tolist()


    # Function to generate dropdown menu for selecting pickup locations
    def select_pickup_location():
        selected_pickup_location = st.selectbox("Select Pickup Location", zone_names)
        return selected_pickup_location


    # Function to display top 5 dropoff locations for the selected pickup location
    def display_top_dropoff_locations(selected_pickup_location):
        st.write(f"Top 5 Dropoff Locations for Pickup Location {selected_pickup_location}:")
        pickup_location_id = taxi_zones[taxi_zones['zone'] ==
selected_pickup_location]['LocationID'].iloc[0]
        top_dropoff_locations = contingency_df[str(pickup_location_id)].nlargest(5)
```

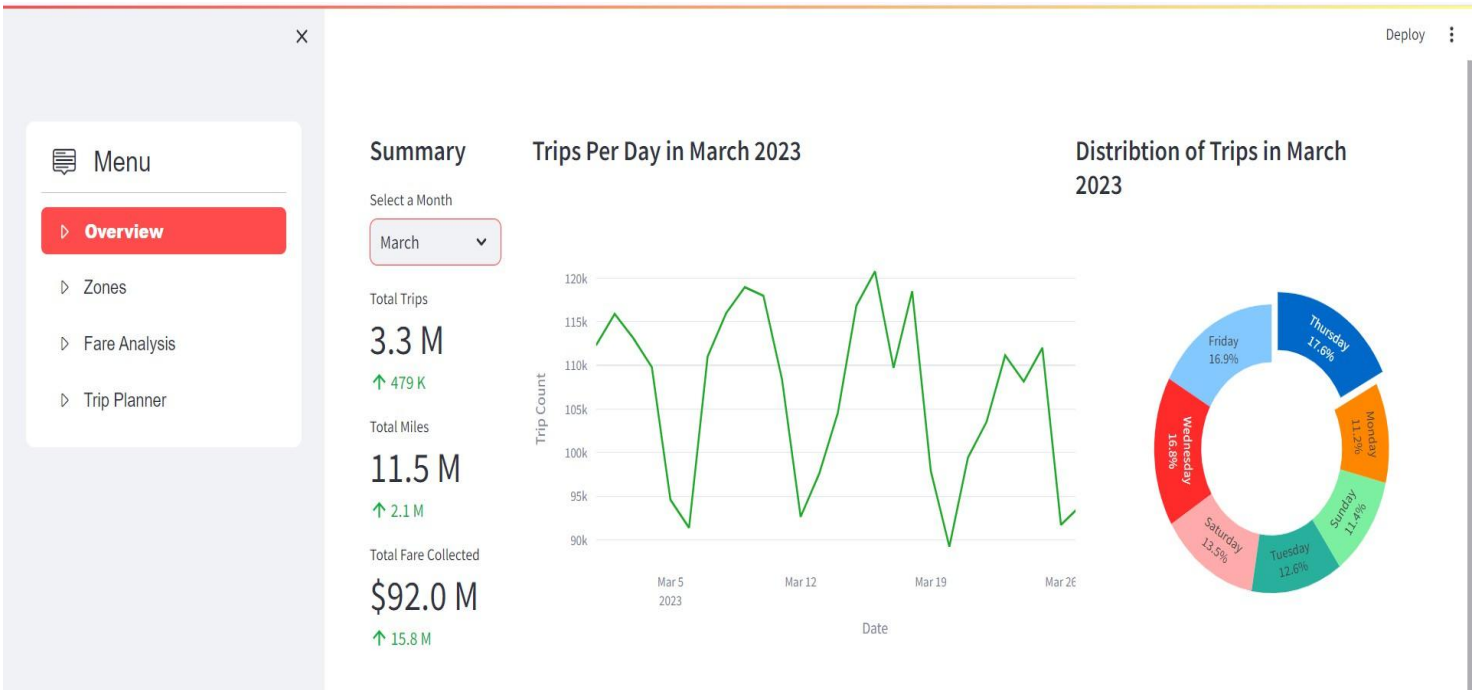
```
        top_dropoff_zones = [location_id_to_zone[int(loc_id)] for loc_id in
top_dropoff_locations.index]
        top_dropoff_df = pd.DataFrame({
            'Dropoff Zone': top_dropoff_zones,
            'Frequency': top_dropoff_locations.values
        })

        st.table(top_dropoff_df)

        colors = cm.viridis(np.linspace(0, 1, len(top_dropoff_df)))  # Use Viridis colormap
        plt.figure(figsize=(8, 5))
        plt.bar(top_dropoff_df['Dropoff Zone'], top_dropoff_df['Frequency'], color=colors)
        plt.xlabel('Dropoff Zone')
        plt.ylabel('Frequency')
        plt.title('Top 5 Dropoff Zones based on Frequency')
        plt.xticks(rotation=45, ha='right')
        st.pyplot(plt)
        return top_dropoff_df['Dropoff Zone'].tolist()  # Return top 5 dropoff locations as a list

    # Create columns for layout
    col1, col2 = st.columns([1, 2])

    # Select pickup location
    with col1:
```

```
        selected_pickup_location = select_pickup_location()


        # Display top 5 dropoff locations
        top_dropoff_zones = display_top_dropoff_locations(selected_pickup_location)


    # Get the LocationID of the selected pickup zone
    pickup_location_id = taxi_zones[taxi_zones['zone'] ==
selected_pickup_location]['LocationID'].iloc[0]


    # Filter taxi zones dataframe to only include the selected pickup and dropoff locations
    selected_zones = taxi_zones[(taxi_zones['LocationID'] == pickup_location_id) |
                                (taxi_zones['zone'].isin(top_dropoff_zones))]


    # Plot the selected zones on a map using matplotlib
    with col2:
        st.markdown("## Selected Zones Map")
        plt.figure(figsize=(5, 5))  # Adjusted size to be more compact
        taxi_zones.plot(color='orange', edgecolor='gray', alpha=0.1)
        plt.scatter(selected_zones['centroid'].x, selected_zones['centroid'].y, color='blue', s=10,
label='Top 5 Zones')
        plt.scatter(selected_zones[selected_zones['zone'] == selected_pickup_location]['centroid'].x,
                    selected_zones[selected_zones['zone'] == selected_pickup_location]['centroid'].y,
                    color='red', s=30, label='Selected Zone: ' + selected_pickup_location)  # Display
selected zone name
        plt.xlabel('Longitude')
```

```
        plt.ylabel('Latitude')
        plt.title('NYC Taxi Zone Map')
        plt.legend()
        st.pyplot(plt)

with st.sidebar:
    option = option_menu(
        menu_title="Menu",
        options=["Overview", "Zones", "Fare Analysis", "Trip Planner"],
        default_index=0
    )

if option == "Overview":
    show_overview()

if option == "Zones":
    show_zones_map()

if option == "Fare Analysis":
    show_fare_analysis()

if option == "Trip Planner":
    trip_planner()
```

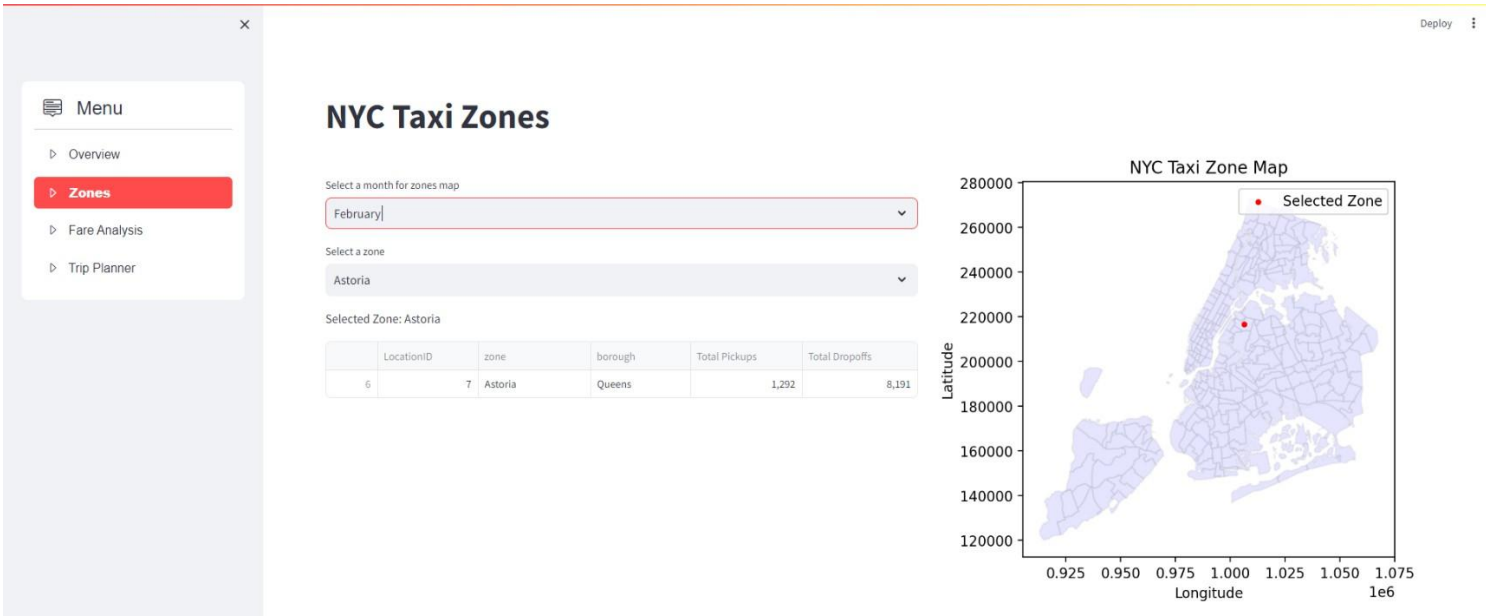
SCREENSHOTS:

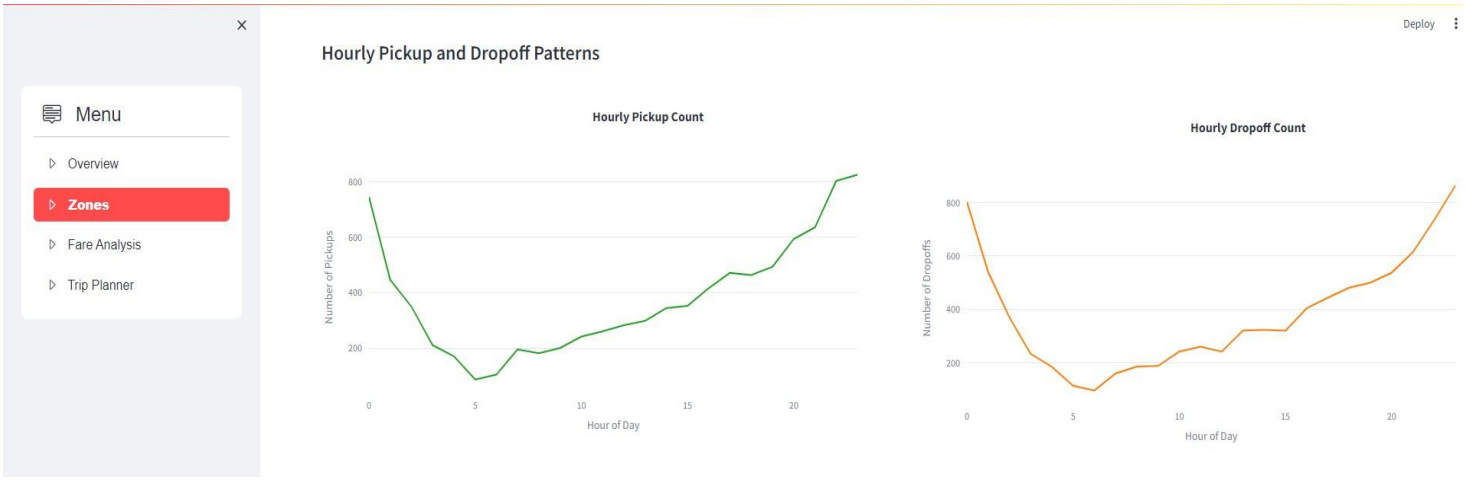
OVERVIEW :

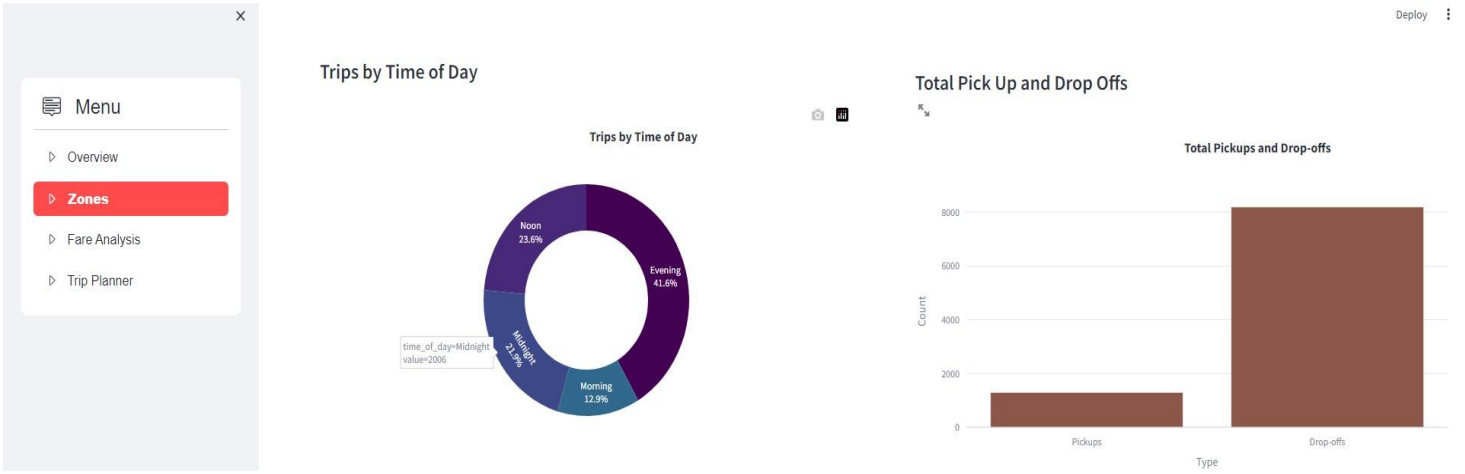


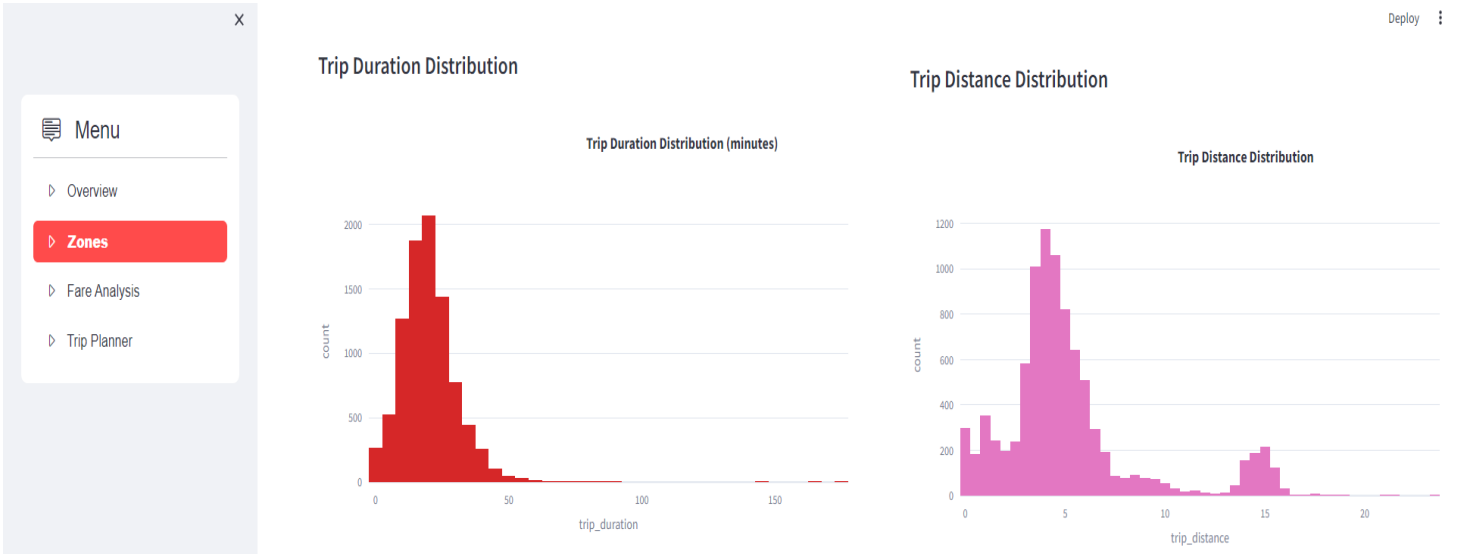
ZONE SECTION:

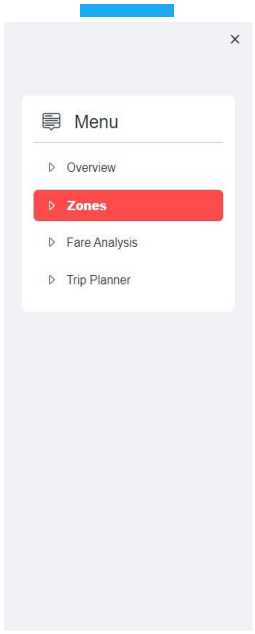
1)SELECTION OF ZONES AND MONTH:

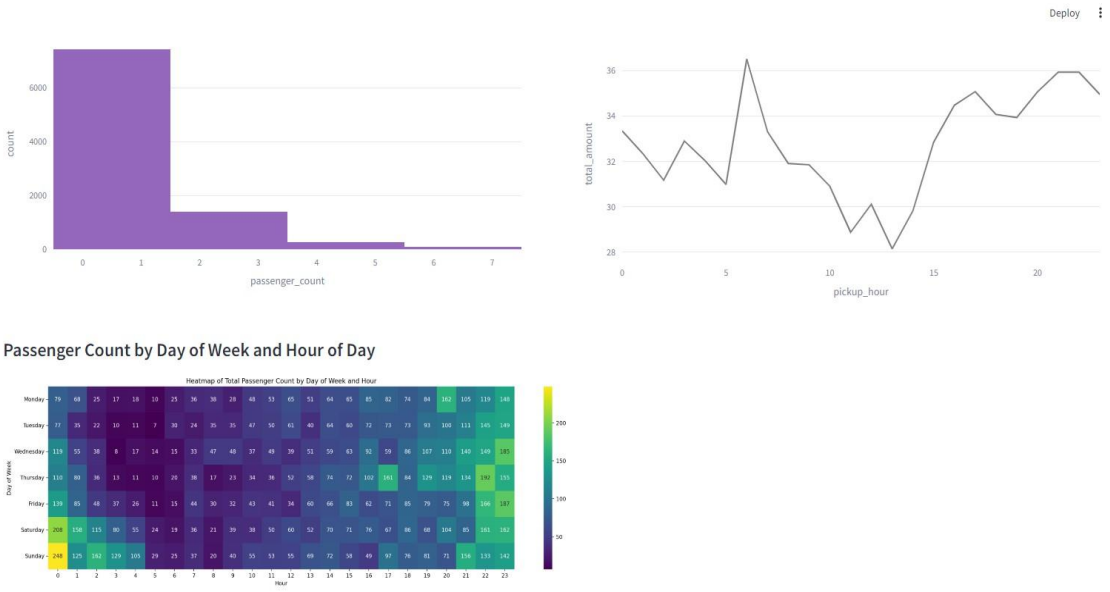



Deploy
Hourly Pickup and Dropoff Patterns
Menu
Overview
Zones
Fare Analysis
Trip Planner
Hourly Pickup Count
Number of Pickups
Hour of Day
Hourly Dropoff Count
Number of Dropoffs
Hour of Day

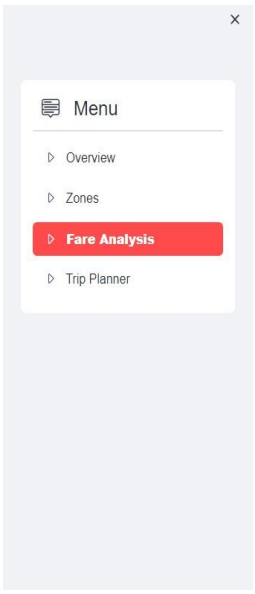

Deploy
Trips by Time of Day
Menu
Overview
Zones
Fare Analysis
Trip Planner
Trips by Time of Day
Noon 23.6%
Evening 41.6%
Midnight 21.9%
Morning 12.9%
time_of_day=Midnight
value=2006
Total Pick Up and Drop Offs
Total Pickups and Drop-offs
Count
Pickups
Drop-offs
Type

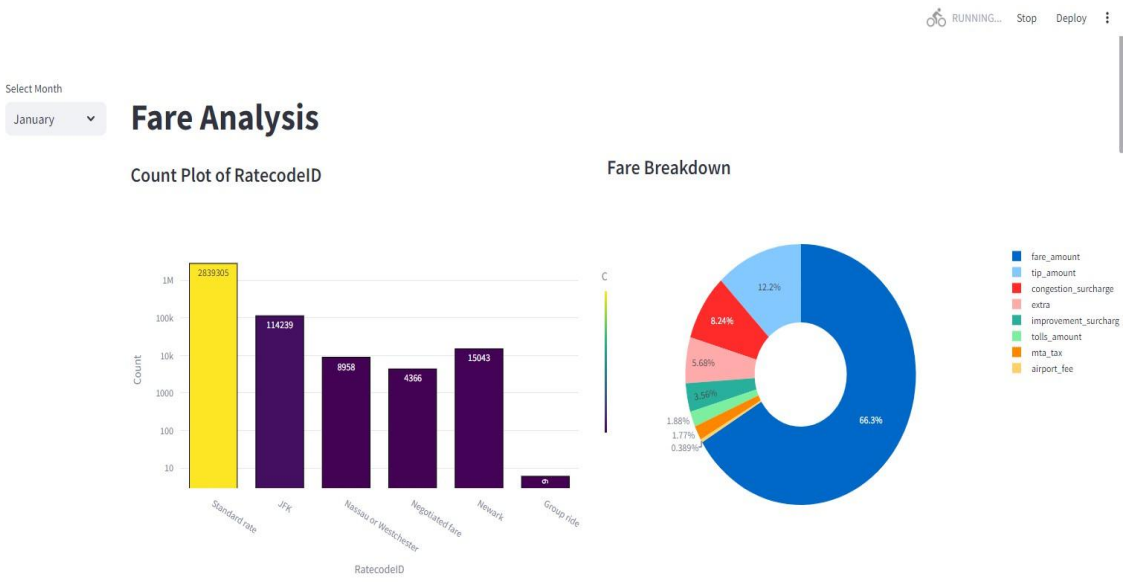

Deploy
Trip Duration Distribution
Menu
Overview
Zones
Fare Analysis
Trip Planner
Trip Duration Distribution (minutes)
count
trip_duration
Trip Distance Distribution
Trip Distance Distribution
count
trip_distance

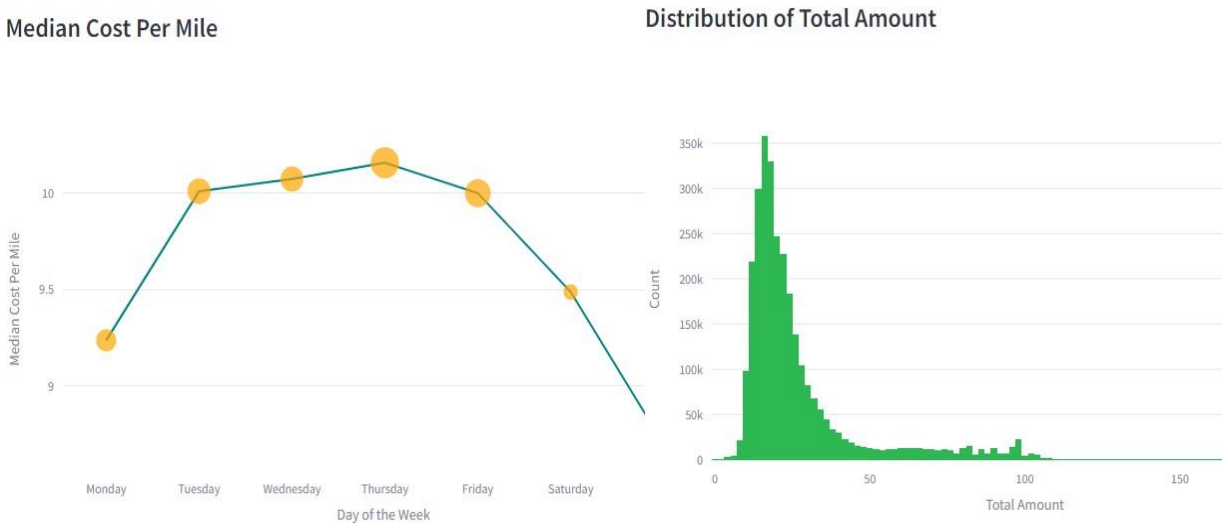




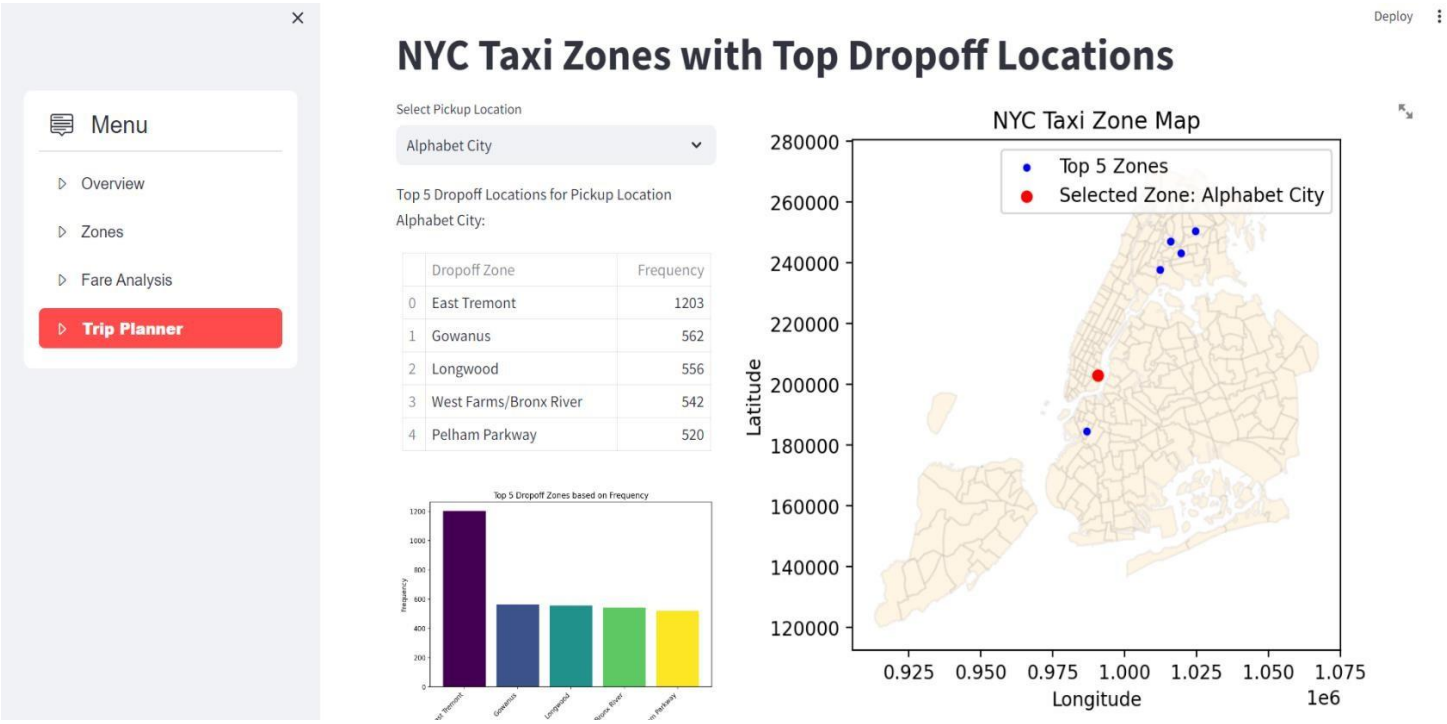
FARE ANALYSIS:







TRIP PLANNER SECTION:



Key Achievements:

- **Comprehensive Data Exploration**: The dashboard provides users with an intuitive interface to explore key statistics, trends, and patterns in NYC taxi trip data. From summary statistics to detailed visualizations, users can gain a deep understanding of taxi trip dynamics.
- **Zone-specific Insights**: By integrating an interactive map and zone-specific analytics, users can delve into taxi trip patterns within specific geographic areas. This feature enables stakeholders to identify hotspots, assess demand-supply dynamics, and optimize service offerings accordingly.
- **Fare Analysis**: The fare analysis section offers users a detailed examination of fare distribution, factors influencing fare amounts, and temporal variations in fare dynamics. By analyzing fare-related metrics, users can make informed decisions related to pricing strategies, fare optimizations, and revenue management.

Future Directions:

- **Real-time Data Integration**: Incorporating real-time data feeds would enable users to access up-to-the-minute insights and respond promptly to emerging trends and events.
- **Predictive Analytics**: Integrating predictive models could enable forecastings of taxi demand, fare fluctuations, and other key metrics, empowering stakeholders to proactively manage operations and resources.
- **User Feedback and Iterative Improvement**: Soliciting user feedback and continuously iterating based on user needs and preferences will ensure that the dashboard remains relevant, user-friendly, and impactful.

CONCLUSION:

In conclusion, The Taxi Data Analysis Dashboard represents a significant milestone in understanding and harnessing the wealth of information contained within New York City's taxi trip data. Through this project, we've created a comprehensive tool that empowers users to explore and derive insights from various facets of taxi trip data, ranging from overall trends to zone-specific patterns and fare analysis.By providing stakeholders with the means to explore, analyze, and derive actionable insights, the dashboard contributes to more informed decision-making, optimized operations, and enhanced service delivery in the taxi industry.