Analysis of Daily Taxi Trips in Chicago Communities in 2022

In this project, we aim to conduct a comprehensive analysis of historical taxi trips in Chicago during 2022. The primary goal of this analysis is to identify communities with high demand for taxi services, enabling our company to make informed decisions regarding resource allocation and strategic planning. By doing so, we hope to foster growth and attract more customers in 2023.

Given that our dataset may not include every taxi trip taken in the city, it is crucial to ensure that our analysis is as representative as possible. To achieve this, we will employ statistical sampling methods, specifically stratified sampling. This approach allows us to account for potential discrepancies in the data and to generate reliable insights into the distribution of taxi trips across Chicago's communities.

In this project, we will be using the bigquery-public-data.chicago_taxi_trips.taxi_trips dataset, which is publicly available on Google BigQuery. This dataset provides a comprehensive record of taxi trips in Chicago and serves as an invaluable resource for our analysis. Before diving into the analysis, we have already filtered the data to focus exclusively on taxi trips taken in 2022. This allows us to concentrate on the most recent and relevant information for our purposes.

Below it's our stratified sampling design for this project.

sampling design

1. Data Preparation

In [27]:
```
# import necessary libraries
import pandas as pd
import numpy as np
import os
import re
from scipy import stats
import math


import geopandas as gpd
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon

import matplotlib.pyplot as plt
import plotly.graph_objects as go
import json
```

In [2]:
```
# set filepath, load dataset, set time start as datetime index
filepath = "../dataset/preprocessed-all-taxi-trips.csv"

main_df = pd.read_csv(filepath, parse_dates=["trip_start_timestamp"], index_col=["trip_s
display(main_df.head(3), print("dataframe's shape:", main_df.shape))
```

```
dataframe's shape: (5866819, 13)
```

	unique_key	trip_end_timestamp	trip_seconds	trip_miles	pic
trip_start_timestamp					
2022-09-30	3696bb7f386062eb3f42a48ae2a7df55cb290853	2022-09-30 23:45:00	596.0	2.54	

2022-09-30 23:30:00+00:00		UTC		
2022-09-30 12:45:00+00:00	5b91f49428e134433e329c3b133b5a73fc466287	2022-09-30 13:15:00 UTC	1740.0	8.80
2022-09-30 04:30:00+00:00	646a2313684586b725de9618521a8bc359d1e91f	2022-09-30 04:45:00 UTC	682.0	10.52

```
None
```

In [3]:
```
# display date range to validate that we have one year period in the dataset
print(f"date range: {main_df.index.min()} to {main_df.index.max()}")
```

```
date range: 2022-01-01 00:00:00+00:00 to 2022-12-31 23:45:00+00:00
```

In [4]:
```
# copy dataframe, remove unnecessary columns, rearrange columns order
all_trips = main_df.copy()
all_trips = all_trips.drop(["trip_end_timestamp", "pickup_community_area", "dropoff_comm
                            "dropoff_location", "payment_type", "dropoff_community"], axi

columns_order = ["unique_key", "trip_seconds",  "trip_minutes", "trip_miles", "fare", "p
all_trips = all_trips[columns_order]

all_trips.head(3)
```

Out[4]:

trip_start_timestamp	unique_key	trip_seconds	trip_minutes	trip_miles	fare	pic
2022-09-30 23:30:00+00:00	3696bb7f386062eb3f42a48ae2a7df55cb290853	596.0	9.93	2.54	9.50	(-8 41
2022-09-30 12:45:00+00:00	5b91f49428e134433e329c3b133b5a73fc466287	1740.0	29.00	8.80	26.25	(-8 41
2022-09-30 04:30:00+00:00	646a2313684586b725de9618521a8bc359d1e91f	682.0	11.37	10.52	27.75	(-8 41

In [5]:
```
# is there any null values?
all_trips.isnull().sum()
```

Out[5]:
```
unique_key          0
trip_seconds        0
trip_minutes        0
trip_miles          0
fare                0
pickup_location     0
pickup_community    0
dtype: int64
```

In [6]:
```
# groupby pickup_community, resample to daily
grouped = main_df.groupby('pickup_community').resample('D')
```

In [7]:
```
# using grouped and resampled object data, calculate the total daily trip for each commu
# extract day name from the date
# rename the columns
daily_trip_counts = grouped["unique_key"].count().reset_index()
daily_trip_counts['trip_start_timestamp'] = daily_trip_counts['trip_start_timestamp'].dt
daily_trip_counts['trip_start_timestamp'] = pd.to_datetime(daily_trip_counts['trip_start
daily_trip_counts['day_name'] = daily_trip_counts['trip_start_timestamp'].dt.day_name()
```

```
daily_trip_counts = daily_trip_counts.rename(columns={'unique_key':'total_trip', 'trip_s

daily_trip_counts
```

Out[7]:

	pickup_community	date	total_trip	day_name
0	Albany Park	2022-01-01	37	Saturday
1	Albany Park	2022-01-02	34	Sunday
2	Albany Park	2022-01-03	46	Monday
3	Albany Park	2022-01-04	41	Tuesday
4	Albany Park	2022-01-05	36	Wednesday
...	...	...	...	...
28096	Woodlawn	2022-12-27	48	Tuesday
28097	Woodlawn	2022-12-28	45	Wednesday
28098	Woodlawn	2022-12-29	36	Thursday
28099	Woodlawn	2022-12-30	36	Friday
28100	Woodlawn	2022-12-31	82	Saturday

28101 rows × 4 columns

In [8]:

```
# using grouped and resampled object data, calculate the total daily fare, distance, and
# extract day name from the date
# rename the columns
daily_trip_features = grouped[['fare', 'trip_miles', 'trip_minutes']].sum().reset_index(
daily_trip_features['trip_start_timestamp'] = daily_trip_features['trip_start_timestamp'
daily_trip_features['trip_start_timestamp'] = pd.to_datetime(daily_trip_features['trip_s
daily_trip_features['day_name'] = daily_trip_features['trip_start_timestamp'].dt.day_nam
daily_trip_features = daily_trip_features.rename(columns={'fare':'sum_fare', 'trip_miles

daily_trip_features
```

Out[8]:

	pickup_community	date	sum_fare	sum_trip_miles	sum_trip_minutes	day_name
0	Albany Park	2022-01-01	645.67	154.59	812.18	Saturday
1	Albany Park	2022-01-02	629.09	165.42	1743.49	Sunday
2	Albany Park	2022-01-03	843.56	205.70	957.52	Monday
3	Albany Park	2022-01-04	738.50	195.25	860.92	Tuesday
4	Albany Park	2022-01-05	586.60	163.55	585.08	Wednesday
...	...	...	...	...	...	...
28096	Woodlawn	2022-12-27	1163.72	322.56	1127.63	Tuesday
28097	Woodlawn	2022-12-28	1065.52	322.06	1003.26	Wednesday
28098	Woodlawn	2022-12-29	853.18	263.72	825.89	Thursday
28099	Woodlawn	2022-12-30	926.60	276.22	974.05	Friday
28100	Woodlawn	2022-12-31	1952.00	610.68	2009.58	Saturday

28101 rows × 6 columns

In [9]:

```
# merge the two dataframes
```

```
# set the date as datetime index
merged_df = pd.merge(daily_trip_counts, daily_trip_features, on=['pickup_community', 'da
merged_df['date'] = merged_df['date'].dt.strftime('%Y-%m-%d')
merged_df.set_index("date", inplace=True)
merged_df.sort_index(ascending=True, inplace=True)


merged_df.shape
```

Out[9]:
```
(28101, 6)
```

In [10]:
```
# display the merged dataframe
merged_df
```

Out[10]:

date	pickup_community	total_trip	day_name	sum_fare	sum_trip_miles	sum_trip_minutes
2022-01-01	Albany Park	37	Saturday	645.67	154.59	812.18
2022-01-01	North Center	43	Saturday	675.79	130.06	531.68
2022-01-01	Avalon Park	3	Saturday	64.25	21.86	56.10
2022-01-01	West Garfield Park	2	Saturday	39.00	11.90	46.00
2022-01-01	Armour Square	19	Saturday	484.37	140.04	375.82
...	...	...	...	...	...	...
2022-12-31	Chatham	148	Saturday	3925.06	1273.30	2852.10
2022-12-31	Roseland	84	Saturday	3056.60	870.86	2411.94
2022-12-31	Calumet Heights	38	Saturday	1010.46	260.34	888.58
2022-12-31	Lake View	1413	Saturday	20059.92	4447.96	21189.78
2022-12-31	Woodlawn	82	Saturday	1952.00	610.68	2009.58

28101 rows × 6 columns

In [11]:
```
# take the community names from the merged dataframe
# create a seperate dataframe for each community using merged dataframe, and store it wi
pickup_communities = merged_df['pickup_community'].unique()

community_dfs = []

for community in pickup_communities:
    community_df = merged_df[merged_df['pickup_community'] == community]
    community_dfs.append(community_df)

# validate the result, it should be 77 community names in Chicago
print("Number of dataframes: ", len(community_dfs))
```

```
Number of dataframes:  77
```

In [12]:
```
# validate by display the first two objects within the list
community_dfs[0:2]
```

Out[12]:
```
[            pickup_community  total_trip   day_name  sum_fare  sum_trip_miles
 date
 2022-01-01       Albany Park          37   Saturday    645.67          154.59  \
 2022-01-02       Albany Park          34     Sunday    629.09          165.42
 2022-01-03       Albany Park          46     Monday    843.56          205.70
 2022-01-04       Albany Park          41    Tuesday    738.50          195.25
 2022-01-05       Albany Park          36  Wednesday    586.60          163.55
```

```
...                        ...         ...        ...       ...             ...
2022-12-27         Albany Park          44    Tuesday    901.67          254.78
2022-12-28         Albany Park          36  Wednesday    604.86          174.78
2022-12-29         Albany Park          39   Thursday    843.28          214.72
2022-12-30         Albany Park          32     Friday   1042.03          267.59
2022-12-31         Albany Park          90   Saturday   1407.36          410.34

            sum_trip_minutes
date
2022-01-01            812.18
2022-01-02           1743.49
2022-01-03            957.52
2022-01-04            860.92
2022-01-05            585.08
...                      ...
2022-12-27            907.89
2022-12-28            666.33
2022-12-29           1841.66
2022-12-30           1540.56
2022-12-31           1540.60

[365 rows x 6 columns],
           pickup_community  total_trip   day_name  sum_fare  sum_trip_miles  \
date
2022-01-01     North Center          43   Saturday    675.79          130.06
2022-01-02     North Center          25     Sunday    392.03           86.28
2022-01-03     North Center          37     Monday    521.19          103.44
2022-01-04     North Center          35    Tuesday    573.15          145.30
2022-01-05     North Center          30  Wednesday    414.81           95.78
...                     ...         ...        ...       ...             ...
2022-12-27     North Center          34    Tuesday    549.84          133.18
2022-12-28     North Center          34  Wednesday    731.46          193.11
2022-12-29     North Center          37   Thursday    592.00          102.36
2022-12-30     North Center          53     Friday    708.53          148.68
2022-12-31     North Center         123   Saturday   1790.56          377.26

            sum_trip_minutes
date
2022-01-01            531.68
2022-01-02            341.43
2022-01-03            509.61
2022-01-04            536.76
2022-01-05            628.42
...                      ...
2022-12-27            598.46
2022-12-28            786.81
2022-12-29            569.33
2022-12-30            736.29
2022-12-31           1690.45

[365 rows x 6 columns]]
```

Great, we already prepared the data for the analysis. We can move to the next step.

2. Taxi Trips Analysis Using Stratified Sampling

sampling design

In [13]:
```python
def is_weekend(day_name):
    """
    Returns a boolean value indicating whether the given day is a weekend day.

    Parameters:
```

```
    -----------
    day_name : str
        The name of the day of the week

    Returns:
    --------
    bool
        True if the day is a weekend day (i.e. Saturday or Sunday), False otherwise.
    """
    return day_name in ['Saturday', 'Sunday']
```

Optimum allocation for sample size for each stratum:

$$n_h = n \cdot \frac{N_h \sigma_h}{\sum_{k=1}^{L} N_k \sigma_k}$$

In [14]:

```
# set empty list
# for each community in community_dfs, stratify by weekday-weekend, calculate the reqire

optimum_stratified_dfs = []

for community_df in community_dfs:
    # separate the dataframe into weekend and weekday data
    weekend_df = community_df[community_df['day_name'].apply(is_weekend)]
    weekday_df = community_df[~community_df['day_name'].apply(is_weekend)]

    # calculate required sample size from the population
    population_std = community_df['total_trip'].std()
    margin_of_error = 0.05 * community_df['total_trip'].mean()
    confidence_level = 0.95
    Z = stats.norm.ppf((1 + confidence_level) / 2)
    population_size = len(community_df)

    n_sample = (Z**2 * population_std**2 * population_size) / ((margin_of_error**2 * (po
    n_sample = math.ceil(n_sample)

    # calculate optimum allocation sample size from each stratum
    weekend_std = weekend_df['total_trip'].std()
    weekday_std = weekday_df['total_trip'].std()
    nh_weekend_size = n_sample * (len(weekend_df) * weekend_std) / (len(weekend_df) * we
    nh_weekday_size = n_sample * (len(weekday_df) * weekday_std) / (len(weekend_df) * we


    # build dataframe
    stratified_df = pd.DataFrame({
        'pickup_community': [community_df.iloc[0]['pickup_community']],
        'population_size': [population_size],
        'population_std': [population_std],
        'population_moe': [margin_of_error],
        'required_sample': [n_sample],
        'weekend_size': [len(weekend_df)],
        'weekday_size': [len(weekday_df)],
        'weekend_std': [weekend_std],
        'weekday_std': [weekday_std],
        'nh_weekend_size': [round(nh_weekend_size)],
        'nh_weekday_size': [round(nh_weekday_size)],
```

```
    })

    # append the stratified_df to the optimum_stratified_dfs list
    optimum_stratified_dfs.append(stratified_df)

# concatenate all dataframes in the optimum_stratified_dfs list into a single dataframe
final_optimum_stratified = pd.concat(optimum_stratified_dfs, ignore_index=True)


# display the results
final_optimum_stratified
```

Out[14]:

	pickup_community	population_size	population_std	population_moe	required_sample	weekend_size	weekday
0	Albany Park	365	7.638385	1.796849	59	105	
1	North Center	365	13.689483	2.472466	90	105	
2	Avalon Park	365	5.422278	0.750000	130	105	
3	West Garfield Park	365	2.809706	0.318767	165	105	
4	Armour Square	365	10.413344	2.209315	70	105	
...	...	...	...	...	...	...	
72	Ashburn	365	4.585832	0.649041	126	105	
73	South Chicago	365	8.906410	1.572055	93	105	
74	Logan Square	365	24.491824	4.295616	94	105	
75	Lake View	365	192.911804	31.399315	104	105	
76	Mount Greenwood	361	1.555276	0.095291	268	102	

77 rows × 11 columns

Unbiased estimator for population means:

$$\bar{y}_{st} = \frac{1}{N}\sum_{h=1}^{L} N_h \bar{y}_h$$

In [15]:

```
def calculate_trip_mean_sample(community_dfs, final_optimum_stratified, n_iterations=100
    """
    Calculates the mean daily taxi trips for each community using stratified sampling wi

    Parameters:
    - community_dfs (list): A list of dataframes, each containing data for one community
    - final_optimum_stratified (DataFrame): A dataframe containing the optimum allocatio
    - n_iterations (int): The number of iterations to perform for the sampling process.

    Returns:
    - mean_samples (list): A list of mean daily taxi trips for each community, calculate
    """
    mean_samples = []

    for i, community_df in enumerate(community_dfs):
        # separate the dataframe into weekend and weekday data
        weekend_df = community_df[community_df['day_name'].apply(is_weekend)]
        weekday_df = community_df[~community_df['day_name'].apply(is_weekend)]
```

```
        # get the number of weekend and weekday samples from final_stratified_df
        nh_weekend_size = final_optimum_stratified.loc[i, 'nh_weekend_size']
        nh_weekday_size = final_optimum_stratified.loc[i, 'nh_weekday_size']

        sample_means = []

        for _ in range(n_iterations):
            # sample the weekend and weekday data proportionally
            weekend_samples = weekend_df.sample(n=nh_weekend_size, replace=True)['total_
            weekday_samples = weekday_df.sample(n=nh_weekday_size, replace=True)['total_

            # calculate the mean of the weekend and weekday samples
            weekend_mean = weekend_samples.mean()
            weekday_mean = weekday_samples.mean()

            # calculate the population size for each stratum
            N_weekend = len(weekend_df)
            N_weekday = len(weekday_df)

            # compute the weighted average using the unbiased estimator
            stratified_mean = (N_weekend * weekend_mean + N_weekday * weekday_mean) / (N
            sample_means.append(stratified_mean)

        # calculate the overall mean of the sample means
        overall_mean = round(np.mean(sample_means))
        mean_samples.append(overall_mean)

    return mean_samples
```

In [16]:
```
# calculate mean samples and mean populations for each community
# loop through the community_dfs and calculate mean population values, store in mean_pop
mean_samples = calculate_trip_mean_sample(community_dfs, final_optimum_stratified)

mean_populations = []

for community_df in community_dfs:
    mean_population = round(community_df['total_trip'].mean())
    mean_populations.append(mean_population)

# add the mean populations and mean samples to the final_optimum_stratified dataframe
final_optimum_stratified['mean_trip_population'] = mean_populations
final_optimum_stratified['mean_trip_samples'] = mean_samples

# sort the dataframe on mean_trip_sample_1000_times
final_optimum_stratified = final_optimum_stratified.sort_values(by="mean_trip_samples",
final_optimum_stratified
```

Out[16]:

	pickup_community	population_size	population_std	population_moe	required_sample	weekend_size	weekday
0	Near North Side	365	1119.096105	202.910274	89	105	
1	Loop	365	1047.088512	141.139589	135	105	
2	Ohare	365	899.777275	135.822466	116	105	
3	Near West Side	365	478.996869	72.729589	115	105	
4	Near South Side	365	420.917368	31.438493	239	105	
...	...	...	...	...	...	...	...
72	Gage Park	365	1.941520	0.142192	243	105	
73	Edison Park	365	1.719608	0.154247	208	105	
74	Hermosa	365	1.748434	0.160548	203	105	

75	Montclare	365	1.657833	0.103288	267	105
76	Mount Greenwood	361	1.555276	0.095291	268	102

77 rows × 13 columns

Estimate variance:

$$\widehat{\text{var}}(\bar{y}_{st}) = \sum_{h=1}^{L} \left(\frac{N_h}{N}\right)^2 \left(\frac{N_h - n_h}{N_h}\right) \frac{s_h^2}{n_h}$$

In [17]:
```
def estimate_variance(row):
    """
    Calculate the estimate variance for stratified sampling in each community.

    Args:
        row (pd.Series): A row from the final_optimum_stratified DataFrame.

    Returns:
        float: The estimate variance for the given community.
    """
    Nh = row['population_size']
    nh_weekend = row['nh_weekend_size']
    nh_weekday = row['nh_weekday_size']
    sh_weekend_sq = row['weekend_std'] ** 2
    sh_weekday_sq = row['weekday_std'] ** 2

    weekend_variance = ((Nh / nh_weekend) * ((Nh - nh_weekend) / Nh) * sh_weekend_sq) /
    weekday_variance = ((Nh / nh_weekday) * ((Nh - nh_weekday) / Nh) * sh_weekday_sq) /

    return weekend_variance + weekday_variance
```

Confidence Intervals:

$$\hat{\mu}_{st} \pm t_{\alpha/2}\sqrt{\widehat{\text{var}}(\hat{\mu}_{st})}$$

In [18]:
```
def calculate_confidence_interval(row):
    """
    Calculate the confidence interval for the mean daily taxi trips in each community.

    Args:
        row (pd.Series): A row from the final_optimum_stratified DataFrame.

    Returns:
        tuple: A tuple containing the lower and upper bounds of the confidence interval.
    """
    mean_population = row['mean_trip_samples']
    estimate_variance = row['estimate_variance']
    confidence_level = 0.95
    Z = stats.norm.ppf((1 + confidence_level) / 2)

    # Calculate the margin of error
    margin_of_error = Z * np.sqrt(estimate_variance)

    # Calculate the confidence interval
    lower_bound = round(mean_population - margin_of_error)
    upper_bound = round(mean_population + margin_of_error)

    return round(margin_of_error), lower_bound, upper_bound
```

In [19]:
```
# apply the functions
# calculate estimate variance and confidence intervals
# display the result after it finished
final_optimum_stratified['estimate_variance'] = final_optimum_stratified.apply(estimate_
final_optimum_stratified['margin_of_error'], final_optimum_stratified['CI_lower_bound'],

final_optimum_stratified
```

Out[19]:

	pickup_community	population_size	population_std	population_moe	required_sample	weekend_size	weekday
0	Near North Side	365	1119.096105	202.910274	89	105	
1	Loop	365	1047.088512	141.139589	135	105	
2	Ohare	365	899.777275	135.822466	116	105	
3	Near West Side	365	478.996869	72.729589	115	105	
4	Near South Side	365	420.917368	31.438493	239	105	
...	...	...	...	...	...	...	
72	Gage Park	365	1.941520	0.142192	243	105	
73	Edison Park	365	1.719608	0.154247	208	105	
74	Hermosa	365	1.748434	0.160548	203	105	
75	Montclare	365	1.657833	0.103288	267	105	
76	Mount Greenwood	361	1.555276	0.095291	268	102	

77 rows × 17 columns

In [20]:
```
final_optimum_stratified.head(10)
```

Out[20]:

	pickup_community	population_size	population_std	population_moe	required_sample	weekend_size	weekday_
0	Near North Side	365	1119.096105	202.910274	89	105	
1	Loop	365	1047.088512	141.139589	135	105	
2	Ohare	365	899.777275	135.822466	116	105	
3	Near West Side	365	478.996869	72.729589	115	105	
4	Near South Side	365	420.917368	31.438493	239	105	
5	Lake View	365	192.911804	31.399315	104	105	
6	Garfield Ridge	365	177.420728	25.121781	126	105	
7	Lincoln Park	365	127.361443	20.625753	105	105	
8	Uptown	365	50.421059	13.781507	46	105	
9	West Town	365	70.086265	10.384247	119	105	

Our analysis of the results shows that the stratified sampling method was effective in estimating the mean daily taxi trips for each community. The mean samples, calculated using unbiased estimators, are very close to the mean population values, indicating the accuracy of the method used.

Based on the top 10 communities with the highest mean daily trips, it's clear that `Near North Side`, `Loop`, and `Ohare` are the busiest communities, with significantly higher daily taxi trips compared to the others. These areas should be prioritized for additional resources and investment, as focusing on these communities could lead to more customers and increased growth for the company.

However, it's also essential not to neglect other communities in the top 10, such as `Near West Side`, `Near South Side`, `Lake View`, `Garfield Ridge`, `Lincoln Park`, `Uptown`, and `West Town`. While their daily taxi trips are not as high as the top three, they still represent significant demand and potential for growth. Allocating resources proportionally to these communities can help optimize the company's operations and ensure better coverage.

In addition to allocating resources, the company can also leverage the findings of this analysis to tailor its marketing strategies and promotional offers. For instance, offering targeted discounts or incentives in these high-demand communities could attract more customers and foster loyalty. By focusing resources and strategies on these communities, the company can effectively tap into the high demand, resulting in increased growth and customer satisfaction.

In [21]:
```
final_optimum_stratified.to_csv("../dataset/preprocessed-stratified-sampling-taxi.csv",
```

3. Visualize the Results

In [22]:
```
# read geojson chicago
# change the community name into a tittle case
chicago_path = "../dataset/Boundaries - Community Areas (current).geojson"
boundaries = gpd.read_file(chicago_path)
boundaries = boundaries[["community", "geometry"]]
boundaries["community"] = boundaries["community"].apply(lambda x: x.title())

boundaries.head()
```

Out[22]:

	community	geometry
0	Douglas	MULTIPOLYGON (((-87.60914 41.84469, -87.60915 ...
1	Oakland	MULTIPOLYGON (((-87.59215 41.81693, -87.59231 ...
2	Fuller Park	MULTIPOLYGON (((-87.62880 41.80189, -87.62879 ...
3	Grand Boulevard	MULTIPOLYGON (((-87.60671 41.81681, -87.60670 ...
4	Kenwood	MULTIPOLYGON (((-87.59215 41.81693, -87.59215 ...

In [23]:
```
boundaries
```

Out[23]:

	community	geometry
0	Douglas	MULTIPOLYGON (((-87.60914 41.84469, -87.60915 ...
1	Oakland	MULTIPOLYGON (((-87.59215 41.81693, -87.59231 ...
2	Fuller Park	MULTIPOLYGON (((-87.62880 41.80189, -87.62879 ...
3	Grand Boulevard	MULTIPOLYGON (((-87.60671 41.81681, -87.60670 ...
4	Kenwood	MULTIPOLYGON (((-87.59215 41.81693, -87.59215 ...
...	...	...
72	Mount Greenwood	MULTIPOLYGON (((-87.69646 41.70714, -87.69644 ...
73	Morgan Park	MULTIPOLYGON (((-87.64215 41.68508, -87.64249 ...
74	Ohare	MULTIPOLYGON (((-87.83658 41.98640, -87.83658 ...
75	Edgewater	MULTIPOLYGON (((-87.65456 41.99817, -87.65456 ...
76	Edison Park	MULTIPOLYGON (((-87.80676 42.00084, -87.80676 ...

77 rows × 2 columns

In [24]:
```
# merge the dataframes on the community name\
# convert the merged dataframe into a geodataframe
merged_df = final_optimum_stratified.merge(boundaries, left_on='pickup_community', right
merged_gdf = gpd.GeoDataFrame(merged_df, geometry='geometry')
merged_gdf.drop(["community"], axis=1, inplace=True)

# display the results
merged_gdf.head()
```

Out[24]:

	pickup_community	population_size	population_std	population_moe	required_sample	weekend_size	weekday_
0	Near North Side	365	1119.096105	202.910274	89	105	
1	Loop	365	1047.088512	141.139589	135	105	
2	Ohare	365	899.777275	135.822466	116	105	
3	Near West Side	365	478.996869	72.729589	115	105	
4	Near South Side	365	420.917368	31.438493	239	105	

In [49]:
```
merged_gdf_json = json.loads(merged_gdf.to_json())

fig = go.Figure(go.Choroplethmapbox(geojson=merged_gdf_json,
                                    locations=merged_gdf.index,
                                    z=merged_gdf['mean_trip_samples'],
                                    colorscale='YlOrRd',
                                    colorbar=dict(len=0.7, title="<b>Daily<br>Trip</b>",
                                    marker_line_width=1,
                                    marker_opacity=0.8,
                                    text=merged_gdf['pickup_community'],
                                    hovertemplate="<b>%{text}</b><br>Mean Daily Trips: %

fig.update_layout(mapbox_style="carto-positron",
                  mapbox_zoom=10,
                  mapbox_center={"lat": merged_gdf.geometry.centroid.y.mean(), "lon": me
                  margin={"r": 0, "t": 0, "l": 0, "b": 0},
                  width=900,
                  height=900,
                  title={
                      'text': "<b>Mean Daily Taxi Trips by Chicago Community</b>",
                      'y': 0.98,
                      'x': 0.5,
                      'xanchor': 'center',
                      'yanchor': 'top'})

fig.show()
```

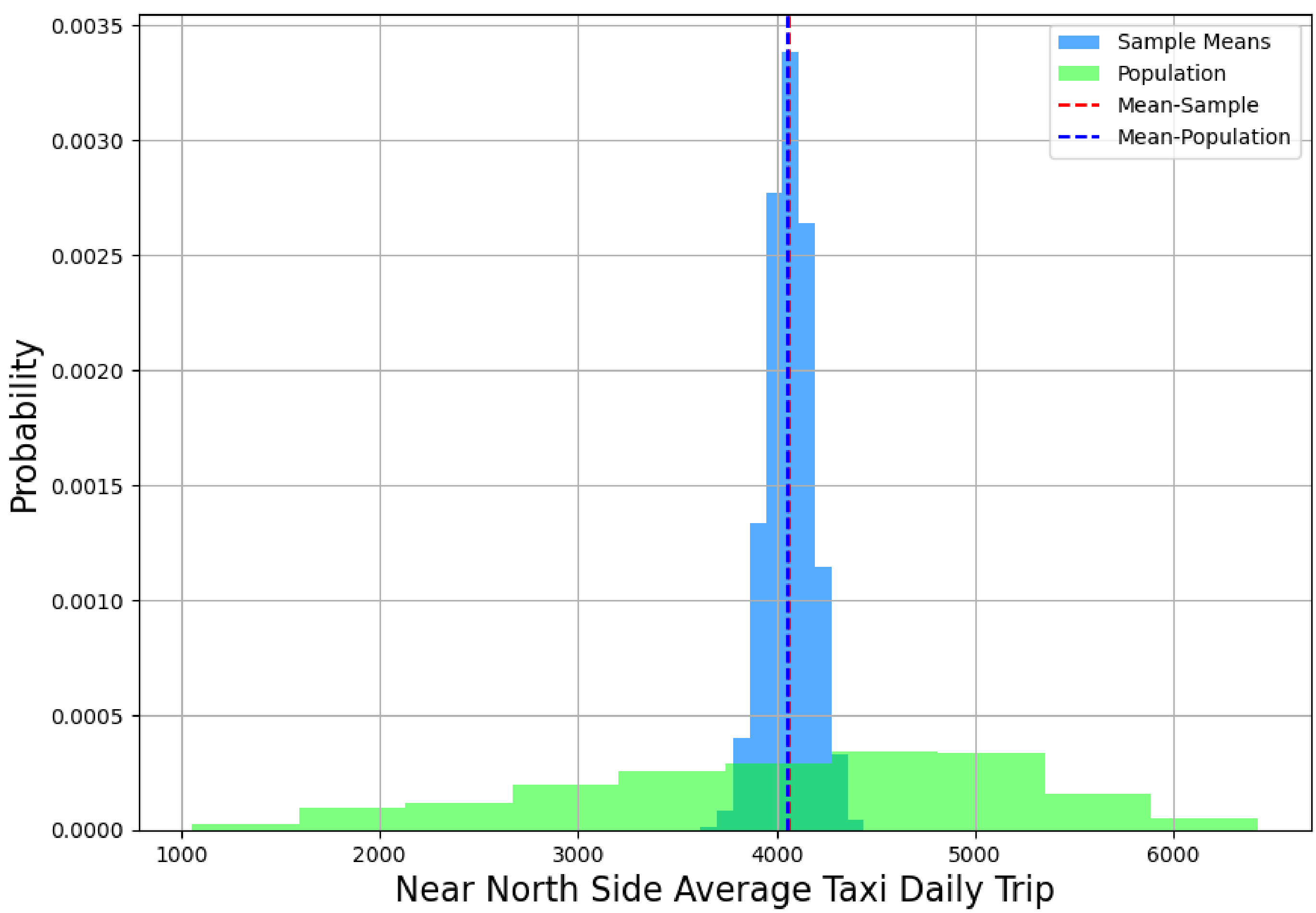
```
C:\Users\PF2L6BL6\AppData\Local\Temp\ipykernel_29892\3693698261.py:15: UserWarning:

Geometry is in a geographic CRS. Results from 'centroid' are likely incorrect. Use 'GeoS
eries.to_crs()' to re-project geometries to a projected CRS before this operation.


C:\Users\PF2L6BL6\AppData\Local\Temp\ipykernel_29892\3693698261.py:15: UserWarning:

Geometry is in a geographic CRS. Results from 'centroid' are likely incorrect. Use 'GeoS
eries.to_crs()' to re-project geometries to a projected CRS before this operation.
```

In [50]:

```
def get_community_distributions(community_name, final_optimum_stratified, n_iterations=1
    """
    Generates a histogram of the mean daily taxi trips for a specific community using st

    Parameters:
    - community_name (str): The name of the community for which the histogram is to be g
    - final_optimum_stratified (DataFrame): A dataframe containing the optimum allocatio
    - n_iterations (int): The number of iterations to perform for the sampling process.

    Returns:
    - The function displays the histogram plots.
    """
    name = community_name
    community_df = [df for df in community_dfs if df['pickup_community'].iloc[0] == name
    community_df = community_df[0]

    stratified = final_optimum_stratified[final_optimum_stratified["pickup_community"]==

    weekend_df = community_df[community_df['day_name'].apply(is_weekend)]
    weekday_df = community_df[~community_df['day_name'].apply(is_weekend)]

    nh_weekend_size = stratified["nh_weekend_size"][0]
    nh_weekday_size = stratified["nh_weekday_size"][0]

    sample_means = []

    for _ in range(n_iterations):
        weekend_samples = weekend_df.sample(n=nh_weekend_size, replace=True)['total_trip
        weekday_samples = weekday_df.sample(n=nh_weekday_size, replace=True)['total_trip

        # combine the weekend and weekday samples
        combined_samples = pd.concat([weekend_samples, weekday_samples], ignore_index=Tr

        # calculate the mean of the combined samples
        sample_mean = combined_samples.mean()
        sample_means.append(sample_mean)

    overall_mean = np.mean(sample_means)

    fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10, 7))

    # create histogram for samples
    ax.hist(sample_means,
            density=True,
            facecolor="dodgerblue",
            alpha=0.75,
            label='Sample Means')

    # create histogram for population
    ax.hist(community_df['total_trip'],
            density=True,
```

```
            facecolor="lime",
            alpha=0.5,
            label='Population')

    ax.axvline(overall_mean,
            color="red",
            linestyle="dashed",
            label="Mean-Sample")

    ax.axvline(stratified["mean_trip_population"][0],
            color="blue",
            linestyle="dashed",
            label="Mean-Population")

    # tidy up plot
    ax.set_xlabel(f"{name} Average Taxi Daily Trip", fontsize=16)
    ax.set_ylabel("Probability", fontsize=16)

    plt.legend()
    plt.grid(True)

    return plt.show()
```

In [51]:
```
get_community_distributions("Near North Side", final_optimum_stratified, 10000)
```



In [ ]: