Maximizing Revenue for Taxi Drivers through Payment Type Analysis

Problem Statement

In the fast-paced taxi booking sector, making the most of revenue is essential for long-term success and driver happiness. Our goal is to use data-driven insights to maximize revenue streams for taxi drivers in order to meet this need. Our research aims to determine whether payment methods have an impact on fare pricing by focusing on the relationship between payment type and fare amount.

Objective

This project's main goal is to run an A/B test to examine the relationship between the total fare and the method of payment. We use Python hypothesis testing and descriptive statistics to extract useful information that can help taxi drivers maximize their revenue. In particular, we want to find out if there is a big difference in the fares for those who pay with credit cards versus those who pay with cash.

Research Question

Is there a relationship between total fare amount and payment type, and can we nudge customers towards payment methods that generate higher revenue for drivers without negatively impacting customer experience?

Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as st
import statsmodels.api as sm
from scipy.stats import mannwhitneyu
import warnings
warnings.filterwarnings('ignore')

#loading the dataset
df = pd.read_parquet(r"C:\Users\ASUS 1\Desktop\yellow_tripdata_2023-01.parquet")
df
```

```
         VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
0               2  2023-01-01 00:32:10   2023-01-01 00:40:36
```

```
1.0000000000
1                2  2023-01-01 00:55:08   2023-01-01 01:01:27
1.0000000000
2                2  2023-01-01 00:25:04   2023-01-01 00:37:49
1.0000000000
3                1  2023-01-01 00:03:48   2023-01-01 00:13:25
0.0000000000
4                2  2023-01-01 00:10:29   2023-01-01 00:21:19
1.0000000000
...            ...                 ...                   ...
...
3066761          2  2023-01-31 23:58:34   2023-02-01 00:12:33
NaN
3066762          2  2023-01-31 23:31:09   2023-01-31 23:50:36
NaN
3066763          2  2023-01-31 23:01:05   2023-01-31 23:25:36
NaN
3066764          2  2023-01-31 23:40:00   2023-01-31 23:53:00
NaN
3066765          2  2023-01-31 23:07:32   2023-01-31 23:21:56
NaN

         trip_distance   RatecodeID store_and_fwd_flag
PULocationID  \
0         0.9700000000 1.0000000000                  N           161

1         1.1000000000 1.0000000000                  N            43

2         2.5100000000 1.0000000000                  N            48

3         1.9000000000 1.0000000000                  N           138

4         1.4300000000 1.0000000000                  N           107

...                ...          ...                ...           ...

3066761   3.0500000000          NaN               None           107

3066762   5.8000000000          NaN               None           112

3066763   4.6700000000          NaN               None           114

3066764   3.1500000000          NaN               None           230

3066765   2.8500000000          NaN               None           262


         DOLocationID  payment_type   fare_amount       extra
mta_tax  \
0                 141             2  9.3000000000 1.0000000000
```

```
0.5000000000
1                 237              1  7.9000000000 1.0000000000
0.5000000000
2                 238              1 14.9000000000 1.0000000000
0.5000000000
3                   7              1 12.1000000000 7.2500000000
0.5000000000
4                  79              1 11.4000000000 1.0000000000
0.5000000000
...               ...            ...           ...          ...
...
3066761            48              0 15.8000000000 0.0000000000
0.5000000000
3066762            75              0 22.4300000000 0.0000000000
0.5000000000
3066763           239              0 17.6100000000 0.0000000000
0.5000000000
3066764            79              0 18.1500000000 0.0000000000
0.5000000000
3066765           143              0 15.9700000000 0.0000000000
0.5000000000

           tip_amount  tolls_amount  improvement_surcharge
total_amount  \
0        0.0000000000  0.0000000000           1.0000000000
14.3000000000
1        4.0000000000  0.0000000000           1.0000000000
16.9000000000
2       15.0000000000  0.0000000000           1.0000000000
34.9000000000
3        0.0000000000  0.0000000000           1.0000000000
20.8500000000
4        3.2800000000  0.0000000000           1.0000000000
19.6800000000
...               ...           ...                    ...             .
..
3066761  3.9600000000  0.0000000000           1.0000000000
23.7600000000
3066762  2.6400000000  0.0000000000           1.0000000000
29.0700000000
3066763  5.3200000000  0.0000000000           1.0000000000
26.9300000000
3066764  4.4300000000  0.0000000000           1.0000000000
26.5800000000
3066765  2.0000000000  0.0000000000           1.0000000000
21.9700000000

         congestion_surcharge  airport_fee
0                2.5000000000 0.0000000000
```

```
1                2.5000000000 0.0000000000
2                2.5000000000 0.0000000000
3                0.0000000000 1.2500000000
4                2.5000000000 0.0000000000
...                       ...          ...
3066761                   NaN          NaN
3066762                   NaN          NaN
3066763                   NaN          NaN
3066764                   NaN          NaN
3066765                   NaN          NaN

[3066766 rows x 19 columns]
```

Exploratory Data Analysis:

```
df.info() #overview of the data

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3066766 entries, 0 to 3066765
Data columns (total 19 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int64
 1   tpep_pickup_datetime   datetime64[us]
 2   tpep_dropoff_datetime  datetime64[us]
 3   passenger_count        float64
 4   trip_distance          float64
 5   RatecodeID             float64
 6   store_and_fwd_flag     object
 7   PULocationID           int64
 8   DOLocationID           int64
 9   payment_type           int64
 10  fare_amount            float64
 11  extra                  float64
 12  mta_tax                float64
 13  tip_amount             float64
 14  tolls_amount           float64
 15  improvement_surcharge  float64
 16  total_amount           float64
 17  congestion_surcharge   float64
 18  airport_fee            float64
dtypes: datetime64[us](2), float64(12), int64(4), object(1)
memory usage: 444.6+ MB

#making a column which can help in our analysis
df['Duration']= (df['tpep_dropoff_datetime']-
df['tpep_pickup_datetime']).dt.total_seconds()/60
df['Duration']
```

```
0           8.4333333333
1           6.3166666667
2          12.7500000000
3           9.6166666667
4          10.8333333333
                ...
3066761    13.9833333333
3066762    19.4500000000
3066763    24.5166666667
3066764    13.0000000000
3066765    14.4000000000
Name: Duration, Length: 3066766, dtype: float64
```

```
#taking only those column which can helpfull in this analysis
df=df[['passenger_count','trip_distance','payment_type','fare_amount',
'Duration']]
df
```

```
         passenger_count  trip_distance  payment_type   fare_amount  \
0           1.0000000000   0.9700000000             2  9.3000000000
1           1.0000000000   1.1000000000             1  7.9000000000
2           1.0000000000   2.5100000000             1 14.9000000000
3           0.0000000000   1.9000000000             1 12.1000000000
4           1.0000000000   1.4300000000             1 11.4000000000
...                  ...            ...           ...           ...
3066761              NaN   3.0500000000             0 15.8000000000
3066762              NaN   5.8000000000             0 22.4300000000
3066763              NaN   4.6700000000             0 17.6100000000
3066764              NaN   3.1500000000             0 18.1500000000
3066765              NaN   2.8500000000             0 15.9700000000

             Duration
0        8.4333333333
1        6.3166666667
2       12.7500000000
3        9.6166666667
4       10.8333333333
...               ...
3066761 13.9833333333
3066762 19.4500000000
3066763 24.5166666667
3066764 13.0000000000
3066765 14.4000000000

[3066766 rows x 5 columns]
```

```
#changing the data type
df['passenger_count'] = df['passenger_count'].astype('Int64')
df['payment_type'] = df['payment_type'].astype('Int64')
```

```
#checking total data
df.shape

(3066766, 5)

#checking for null values
df.isnull().sum()

passenger_count    71743
trip_distance          0
payment_type           0
fare_amount            0
Duration               0
dtype: int64

null_values_percentage =
df['passenger_count'].isnull().sum()/len(df)*100
print(f"total percentage of null values in passenger count columns
is :{null_values_percentage:.2f}")

total percentage of null values in passenger count columns is :2.34

#removing duplicates
df = df.dropna(subset=['passenger_count'])
after_removing=df.shape
print(f"after removing the null values total data
left{after_removing}")

after removing the null values total data left(2995023, 5)

#check for duplicates value
duplicates = df.duplicated().sum()
print(f"duplicate value are:{duplicates}")
duplicates_percentage = df.duplicated().sum()/len(df)*100
print(f'Total duplicate percentage is:{duplicates_percentage:.2f} %')

duplicate value are:1191207
Total duplicate percentage is:39.77 %

#drop duplicate values
df.drop_duplicates(inplace=True)

new_shape = df.shape
print(f'after droping the duplicate values:{new_shape}')

after droping the duplicate values:(1803816, 5)

#checking distribution of specific columns:
# Set Pandas to display full decimal values instead of scientific
notation
pd.options.display.float_format = '{:.10f}'.format
df['passenger_count'].value_counts(normalize=True)
```

```
passenger_count
1   0.6633780829
2   0.1934260479
3   0.0540337817
4   0.0286326322
5   0.0227933448
0   0.0225139371
6   0.0152110858
8   0.0000072069
7   0.0000033263
9   0.0000005544
Name: proportion, dtype: Float64
```

```
df['payment_type'].value_counts(normalize=True)
```

```
payment_type
1   0.7469836170
2   0.2281945609
4   0.0169335453
3   0.0078882769
Name: proportion, dtype: Float64
```

Passenger Count:

Since typical NYC taxi rides are not expected to have more than 6 passengers—and rides with more than 6 passengers contribute minimally to our analysis—we have filtered out any records where passenger_count exceeds 6.

Payment Type:

For our analysis, we are focusing exclusively on the two primary payment methods: cash and credit card. Therefore, we have filtered the dataset to include only rides where the payment type is either cash or credit card.

```
df = df.query('passenger_count >0 & passenger_count<6')
df = df.query('payment_type<3')
after_filtering = df.shape
print(f'after filtering :{after_filtering}')
```

```
after filtering :(1692444, 5)
```

```
# Convert the column to string and then replace values
df['payment_type'] = df['payment_type'].astype(str).replace({'1':
'card', '2': 'cash'})
df[['payment_type']].head(5)
```

```
  payment_type
0         cash
1         card
2         card
```

```
4          card
5          card
```

Statistical Summary

```
df.describe()
```

```
         passenger_count       trip_distance        fare_amount  \
count 1692444.0000000000 1692444.0000000000 1692444.0000000000
mean        1.5019451161        4.8760649392      24.2009338330
std         0.9125023812       55.9302572553      20.4781648876
min         1.0000000000        0.0000000000    -495.1000000000
25%         1.0000000000        1.5000000000      11.4000000000
50%         1.0000000000        2.6800000000      17.0000000000
75%         2.0000000000        5.6000000000      28.9000000000
max         5.0000000000    62359.5200000000    1160.1000000000

                Duration
count 1692444.0000000000
mean       20.5432628889
std        55.5370555513
min       -29.2000000000
25%         9.9666666667
50%        15.6833333333
75%        23.4500000000
max     10029.1833333333
```

Observations-

Passenger_count: Ranges from 1 to 5, which seems reasonable.

Trip_distance: Median is 2.68, 75th percentile is 5.6, but the max is 62,359.52 miles. This indicates extreme high values that are likely errors or rare events.

Fare_amount: The median is 17.00 and the 75th percentile is 28.90, but you have a negative minimum (-495.10) and a maximum of 1160.10. Negative fares are likely data errors, and the extremely high fare might be an outlier.

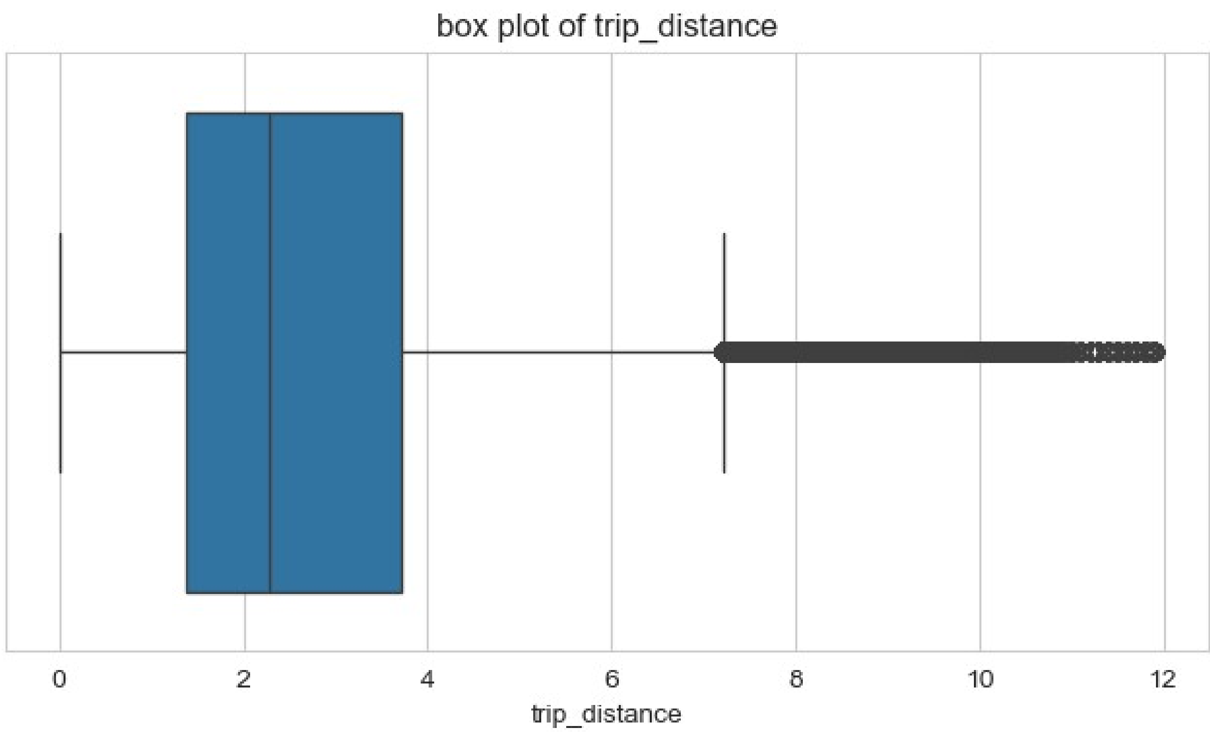
Duration: Median is 15.68 minutes and 75th percentile is 23.45 minutes, but the min is -29.20 and the max is over 10,000 minutes. Negative durations are clearly erroneous, and the extremely high duration values need further investigation.

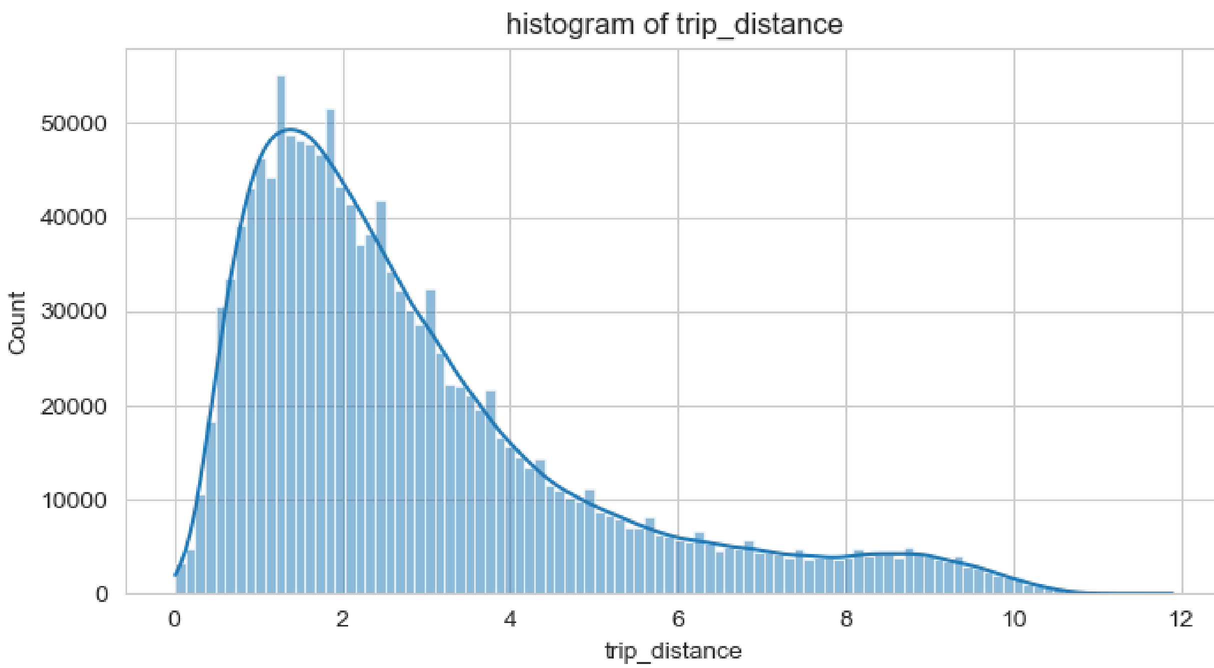
```
#droping negative values
df = df.query('fare_amount > 0 & Duration > 0 & trip_distance > 0')
```

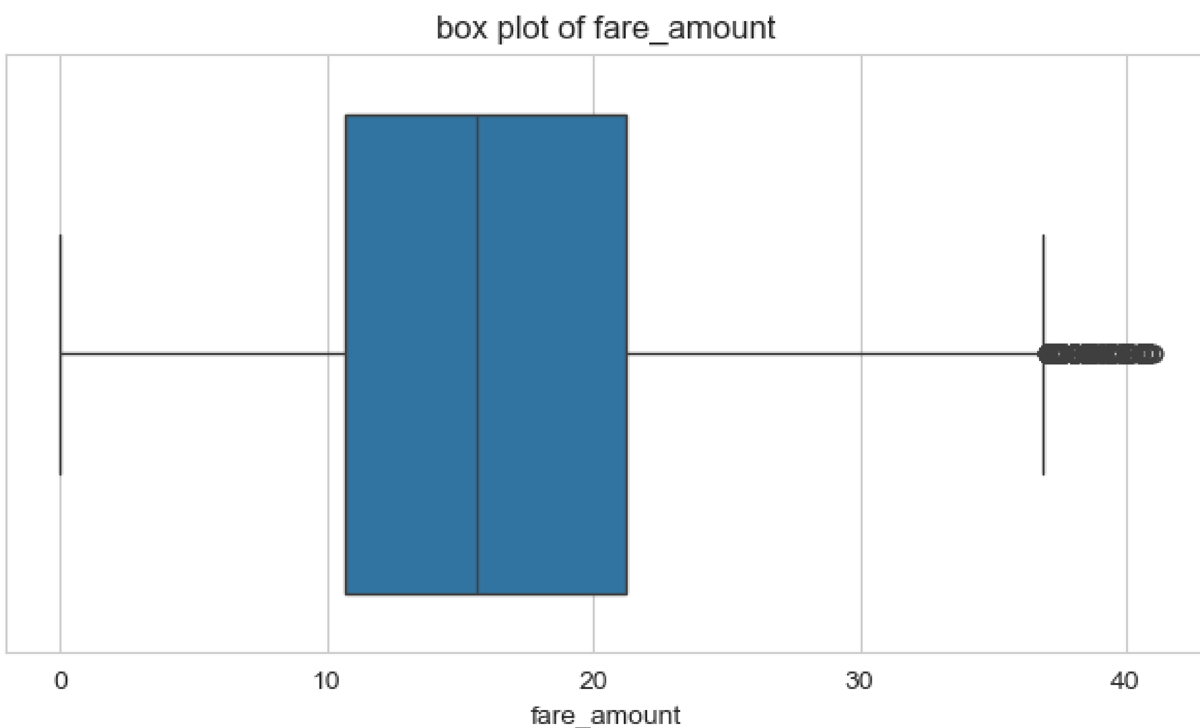
Checking outliers

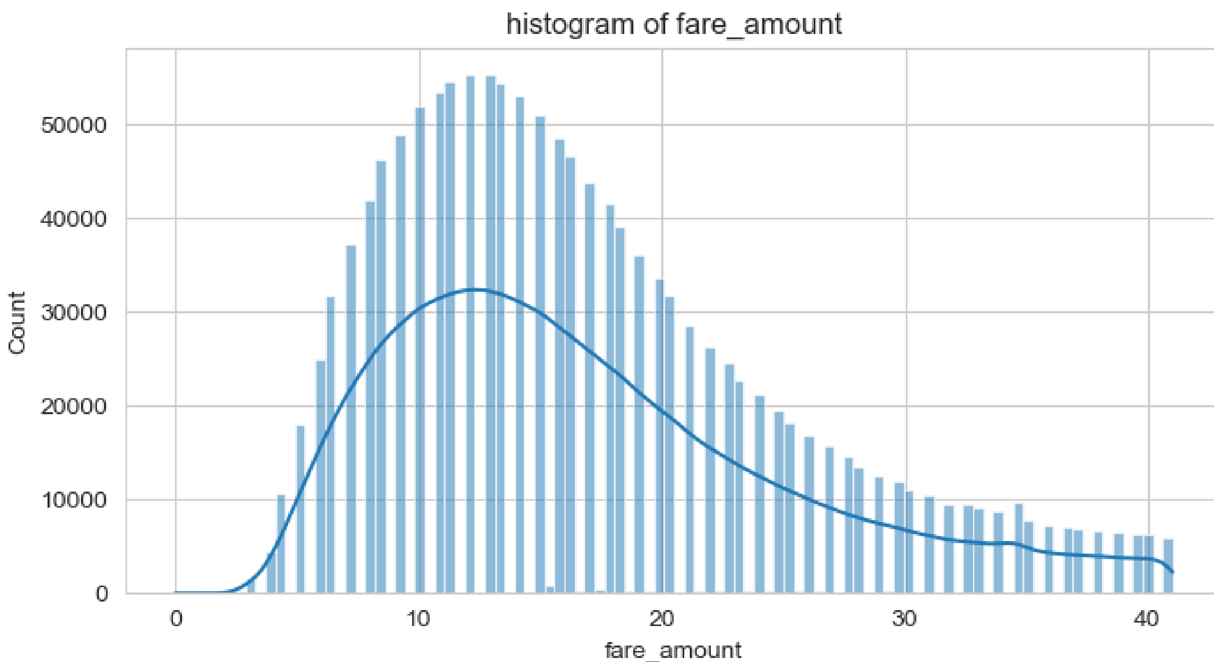
```
# List of columns to visualize
cols = ['trip_distance', 'fare_amount', 'Duration']
```

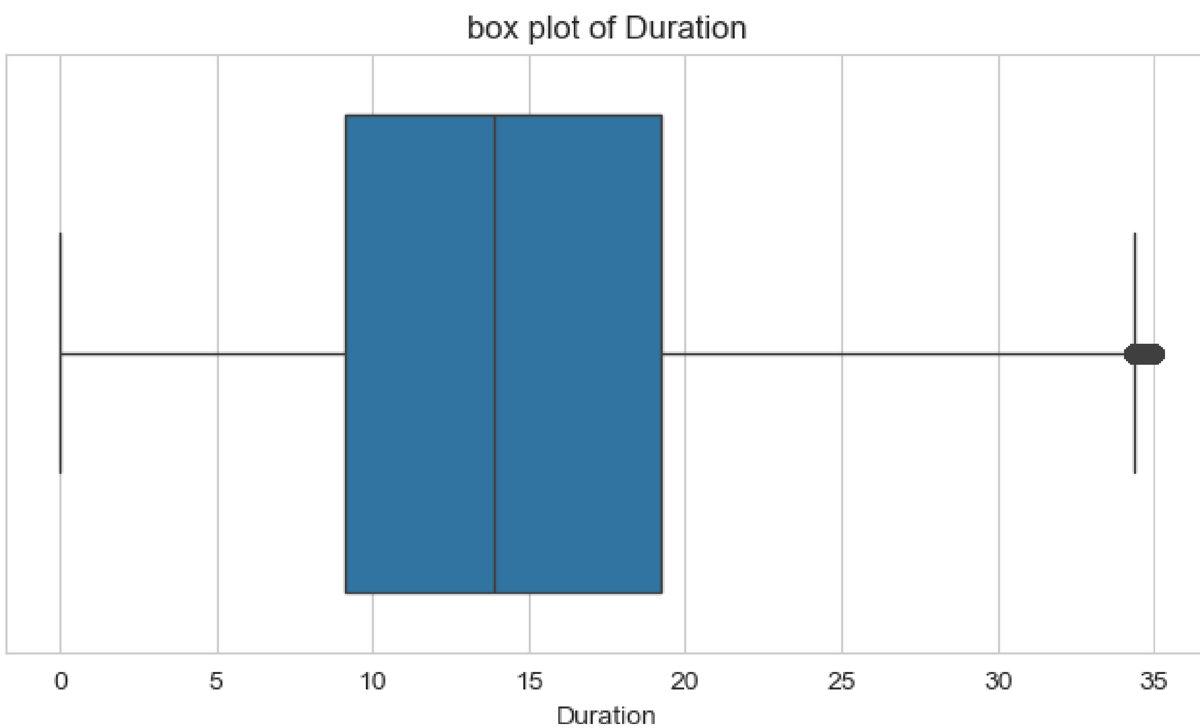
```
#using boxplot
for col in cols:
    plt.figure(figsize=(8,4))
    sns.boxplot(x=df[col])
    plt.title(f'box plot of {col}')
    plt.show()
#using histogram
    plt.figure(figsize=(8,4))
    sns.histplot(df[col],bins=100,kde=True)
    plt.title(f'histogram of {col}')
    plt.show()
```

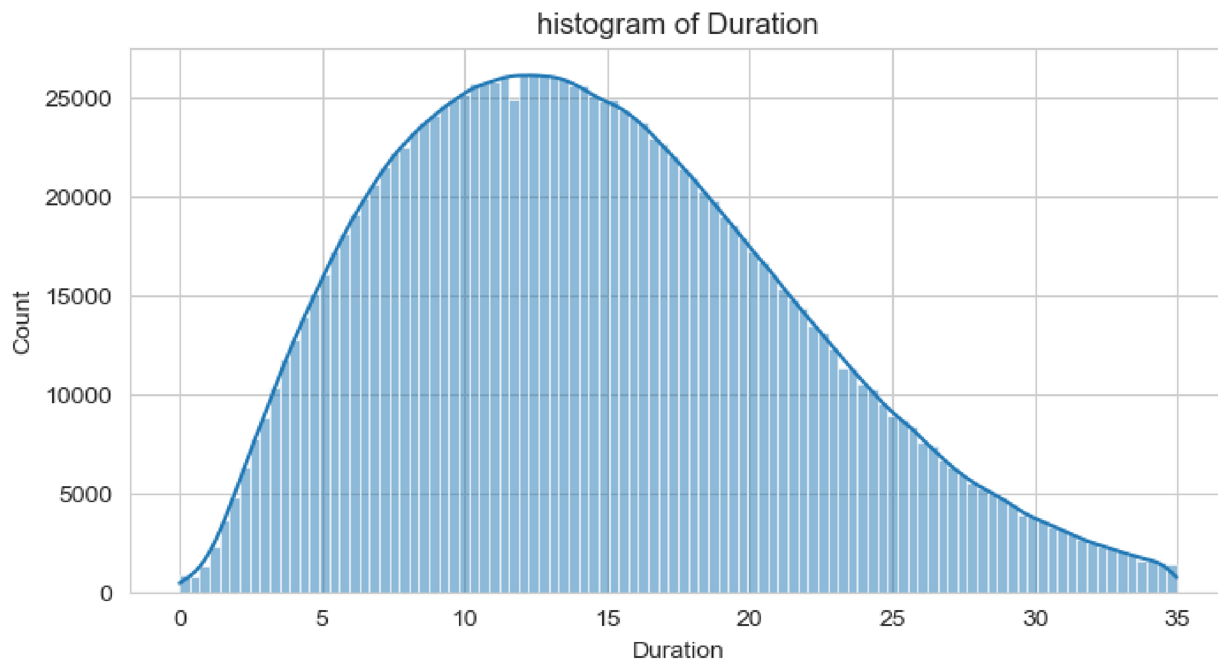



histogram of trip_distance
Count
50000
40000
30000
20000
10000
0
0
2
4
6
8
10
12
trip_distance


box plot of fare_amount
0
10
20
30
40
fare_amount


histogram of fare_amount
Count
0
10000
20000
30000
40000
50000
0
10
20
30
40
fare_amount


box plot of Duration
0
5
10
15
20
25
30
35
Duration



After plotting both boxplots and histograms for the trip_distance, fare_amount, and Duration columns, I observed that each variable contains a significant number of extreme values (outliers). These outliers cause heavy right-skew in the distributions, making it difficult to visualize the main cluster of data.

Next Step: I will proceed with a systematic approach (e.g., IQR method) to remove or cap these extreme values so that my analysis and statistical tests are not overly influenced by a relatively small number of anomalous points. By handling these outliers, I aim to improve the reliability of the insights drawn from the data.

Removing outliers

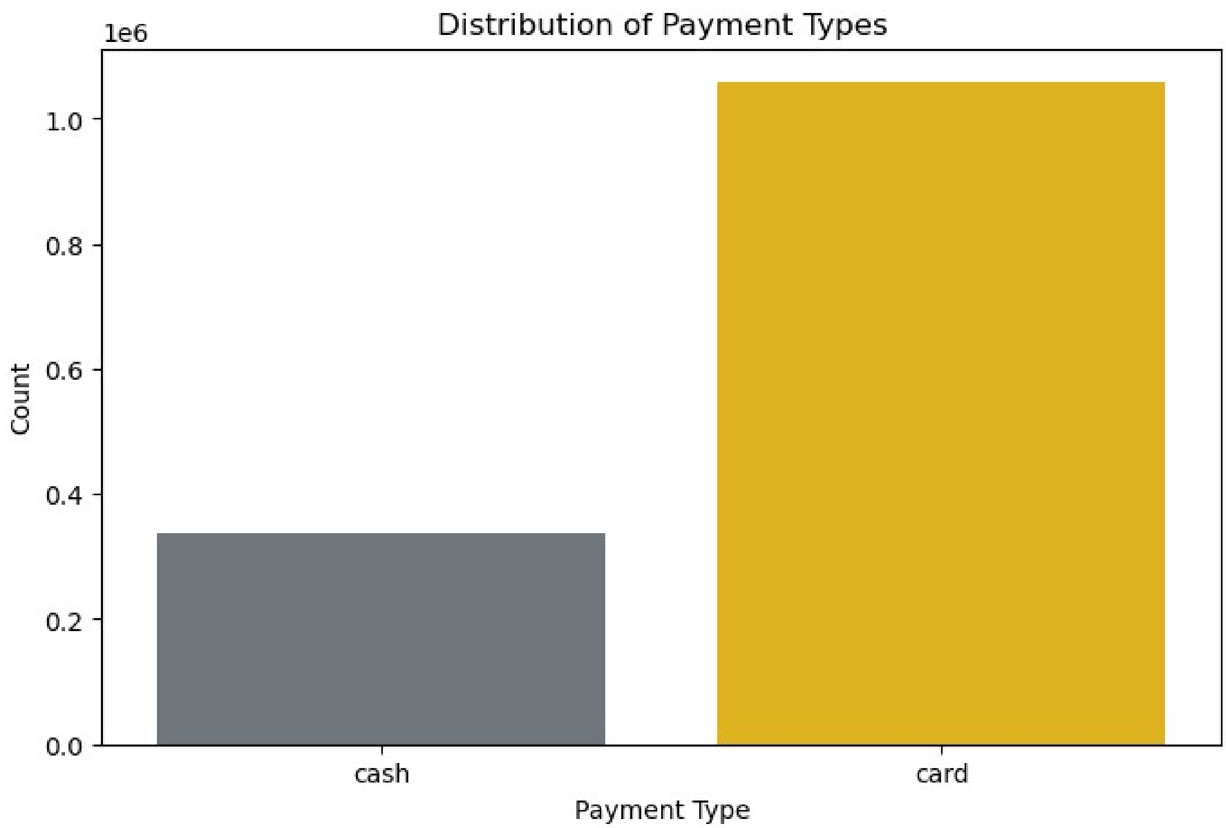
```
#using IQR method
cols = ['trip_distance', 'fare_amount', 'Duration']
for col in cols:
    q1= df[col].quantile(0.25)
    q3= df[col].quantile(0.75)
    iqr = q3-q1
    lower_bound = q1-1.5*iqr
    upper_bound = q3+1.5*iqr
    df = df.query(f"{col} >= @lower_bound and {col} <= @upper_bound")


df
```

```
      passenger_count  trip_distance payment_type   fare_amount  \
0                   1   0.9700000000         cash  9.3000000000
1                   1   1.1000000000         card  7.9000000000
2                   1   2.5100000000         card 14.9000000000
4                   1   1.4300000000         card 11.4000000000
```

```
5                       1   1.8400000000         card 12.8000000000
...                   ...            ...          ...           ...
2995014                 2   1.7400000000         card 11.4000000000
2995015                 1   2.8400000000         card 13.5000000000
2995016                 1   7.4000000000         card 29.6000000000
2995019                 1   3.3700000000         card 15.6000000000
2995021                 2   3.8000000000         card 17.7000000000

             Duration
0        8.4333333333
1        6.3166666667
2       12.7500000000
4       10.8333333333
5       12.3000000000
...               ...
2995014  9.9666666667
2995015  7.7500000000
2995016 12.7166666667
2995019 10.8666666667
2995021 10.7666666667

[1396856 rows x 5 columns]
```

Feature Analysis: Understanding Fare Amount,Passenger Count & Trip Distance by Payment Type

```
# Set a consistent color theme
custom_palette = {"cash": "#6C757D", "card":  "#FFC300"}  # Light peach for cash, deep red for card
plt.figure(figsize=(8, 5))
#using countplot
sns.countplot(x=df['payment_type'], palette=custom_palette)
plt.title("Distribution of Payment Types")
plt.xlabel("Payment Type")
plt.ylabel("Count")
plt.show()
```

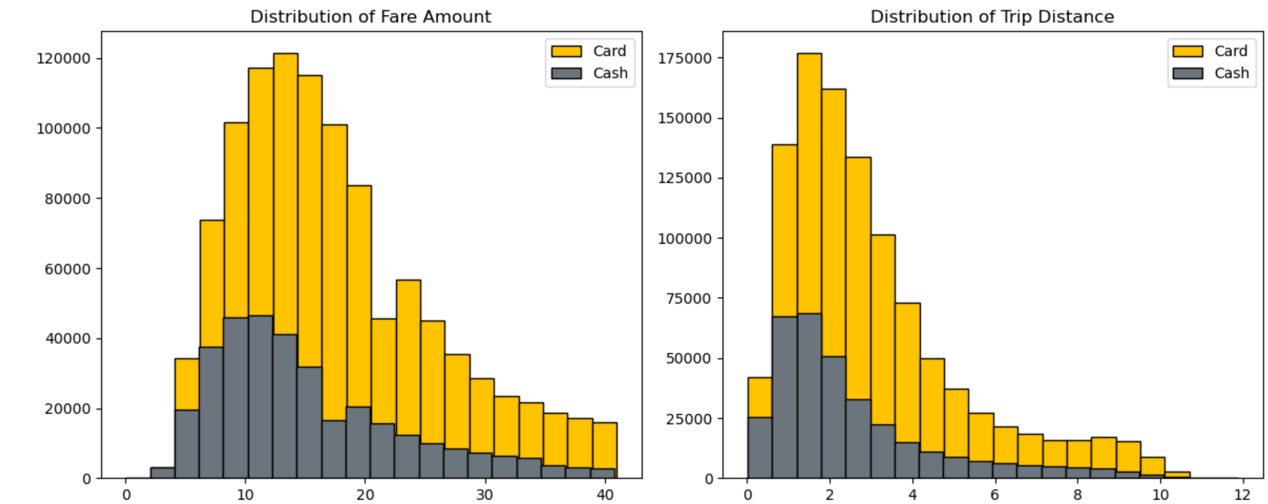


Observations from the Countplot:

Card payments are significantly higher than cash payments. passengers might prefer card payments due to convenience and speed.

```
plt.figure(figsize=(12, 5))

# Subplot 1: Distribution of fare amount
plt.subplot(1, 2, 1)
plt.title('Distribution of Fare Amount')
plt.hist(df[df['payment_type'] == 'card']['fare_amount'],
         histtype='barstacked', bins=20, edgecolor='k',
         color=custom_palette['card'], label='Card')
plt.hist(df[df['payment_type'] == 'cash']['fare_amount'],
         histtype='barstacked', bins=20, edgecolor='k',
         color=custom_palette['cash'], label='Cash')
plt.legend()

# Subplot 2: Distribution of trip distance
plt.subplot(1, 2, 2)
plt.title('Distribution of Trip Distance')
plt.hist(df[df['payment_type'] == 'card']['trip_distance'],
```

```
        histtype='barstacked', bins=20, edgecolor='k',
        color=custom_palette['card'], label='Card')
plt.hist(df[df['payment_type'] == 'cash']['trip_distance'],
        histtype='barstacked', bins=20, edgecolor='k',
        color=custom_palette['cash'], label='Cash')
plt.legend()

plt.tight_layout()  # Improves subplot spacing
plt.show()
```



Observations Card vs. Cash The yellow bars (Card) generally dominate at higher fare amounts, suggesting card payments might be more common for pricier trips. The gray bars (Cash) peak somewhat lower but still overlap significantly.

Skewed Distribution Even after outlier removal, fares can remain right-skewed—some longer, costlier trips are valid.

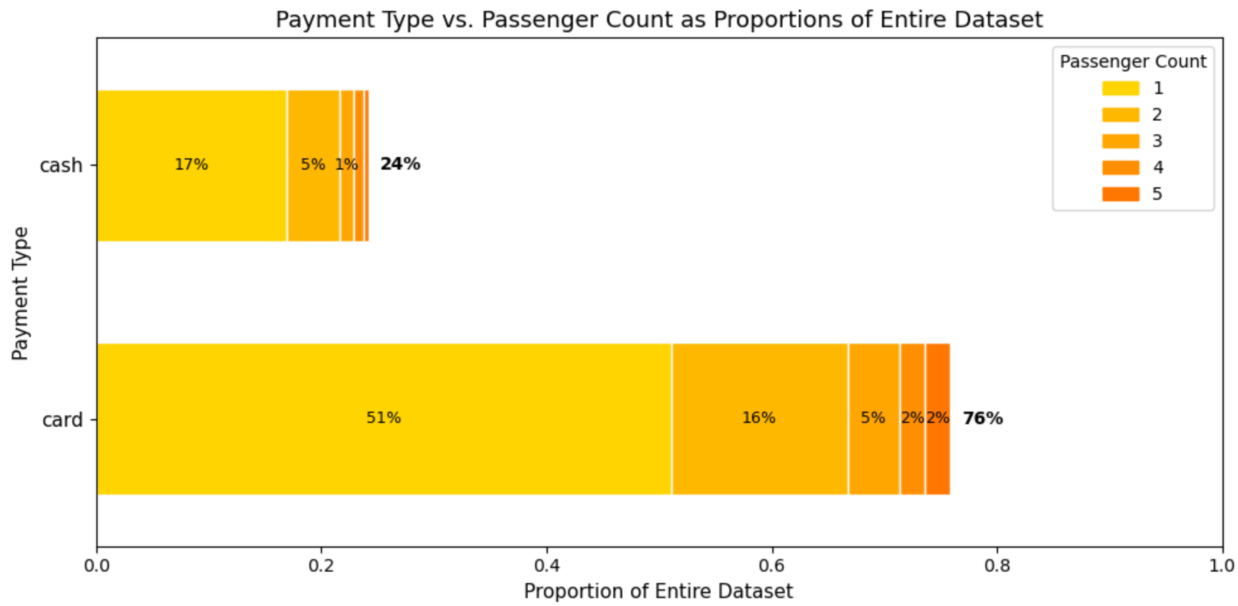
Multiple Peaks The distribution looks somewhat “multi-modal” (multiple small peaks). This might reflect different rate zones or typical trip patterns (e.g., short local rides vs. airport trips).

```
#using stack bar chart we are analyzing passenger count and payment
type
# 1) Count how many rides belong to each (payment_type,
passenger_count)
df_group = df.groupby(['payment_type',
'passenger_count']).size().reset_index(name='count')

# 2) Also find how many rides belong to each payment_type overall
df_payment = df_group.groupby('payment_type')
['count'].sum().reset_index(name='payment_count')

# 3) Calculate total rides
total_rides = df_payment['payment_count'].sum()
```

```
# 4) Merge data so each row has both:
#    - passenger_count
#    - payment_type
#    - count (rides in that group)
#    - payment_count (total rides for that payment type)
df_merged = pd.merge(df_group, df_payment, on='payment_type')

# 5) Calculate proportion_of_total for each segment (passenger_count
within payment_type)
#    relative to the ENTIRE dataset
df_merged['prop_of_total'] = df_merged['count'] / total_rides

# 6) Pivot so rows = payment_type, columns = passenger_count, values =
prop_of_total
df_pivot = df_merged.pivot(index='payment_type',
columns='passenger_count', values='prop_of_total').fillna(0)

# 7) Sort passenger_count columns if needed
#    (e.g., passenger_count from 1 to 5)
df_pivot = df_pivot[sorted(df_pivot.columns)]

# 8) We'll plot two horizontal bars (one for each payment_type),
#    each subdivided by passenger_count, summing to the
proportion_of_total
#    for that payment type.
fig, ax = plt.subplots(figsize=(10, 5))

# We'll track the left edge of each segment
y_positions = [0, 1]  # top bar = 0, bottom bar = 1
bar_height = 0.6

# Define a custom palette for passenger_count (5 categories)
passenger_colors = ["#FFD400", "#FFB800", "#FFA600", "#FF8F00",
"#FF7700"]

# We'll plot from left to right. Each row is a bar: "cash" or "card"
payment_types = df_pivot.index.tolist()

for i, ptype in enumerate(payment_types):
    # This row is a Series with passenger_count=1..5 proportions
    row_data = df_pivot.loc[ptype]
    left_edge = 0.0

    for j, (pcount, val) in enumerate(row_data.items()):
        if val > 0:
            # Plot a rectangle from left_edge to left_edge+val
            ax.barh(
                y=y_positions[i],
                width=val,
                left=left_edge,
```

```
                height=bar_height,
                color=passenger_colors[j % len(passenger_colors)],
                edgecolor='white'
            )

            # Add percentage label if segment > 1%
            if val > 0.01:
                ax.text(
                    left_edge + val/2,
                    y_positions[i],
                    f"{val*100:.0f}%",
                    ha='center', va='center', color='black',
fontsize=9
                )

            left_edge += val

    # Also label the entire bar with the total proportion for that
payment_type
    # sum of row_data is proportion_of_dataset for that payment_type
    total_prop = row_data.sum()
    ax.text(
        total_prop + 0.01,  # place text slightly to the right
        y_positions[i],
        f"{(total_prop*100):.0f}%",
        ha='left', va='center', color='black', fontsize=10,
fontweight='bold'
    )

# Format y-axis ticks with the payment_type labels
ax.set_yticks(y_positions)
ax.set_yticklabels(payment_types, fontsize=11)

ax.set_xlim(0, 1)  # 0% to 100%
ax.set_ylim(-0.5, 1.5)

ax.set_xlabel("Proportion of Entire Dataset", fontsize=11)
ax.set_ylabel("Payment Type", fontsize=11)
ax.set_title("Payment Type vs. Passenger Count as Proportions of
Entire Dataset", fontsize=13)

# Create a custom legend for passenger_count
# (just show squares for 1..5)
handles = []
labels = []
for idx, col in enumerate(df_pivot.columns):
    patch = plt.Rectangle((0, 0), 1, 1, color=passenger_colors[idx],
edgecolor='white')
    handles.append(patch)
    labels.append(str(col))
```

```
legend = ax.legend(handles, labels, title="Passenger Count",
bbox_to_anchor=(1.0, 1.0))
plt.tight_layout()
plt.show()
```



Observations Overall Dataset Distribution

Card payments represent about 76% of all rides, while cash payments make up the remaining 24%. This indicates that card usage dominates in the dataset. Passenger Count Breakdown

For card rides: 51% of the entire dataset consists of single-passenger trips paid by card. The remaining 25% (within card) are multi-passenger trips of 2 or more people. For cash rides: 17% of the entire dataset are single-passenger cash rides. The remaining 7% are multi-passenger cash rides. Single-Passenger Dominance

Combining both payment types, single-passenger trips constitute the majority of rides (over two-thirds of the dataset). Multi-passenger trips (2–5) form a smaller fraction overall, regardless of payment method. Practical Takeaway

Taxis see a high reliance on card payments, especially for single-passenger rides. Cash transactions remain relevant, but they represent a smaller share, mostly single-passenger as well. This distribution highlights the convenience preference (card) and the rarity of larger group rides.

Hypotheses Testing

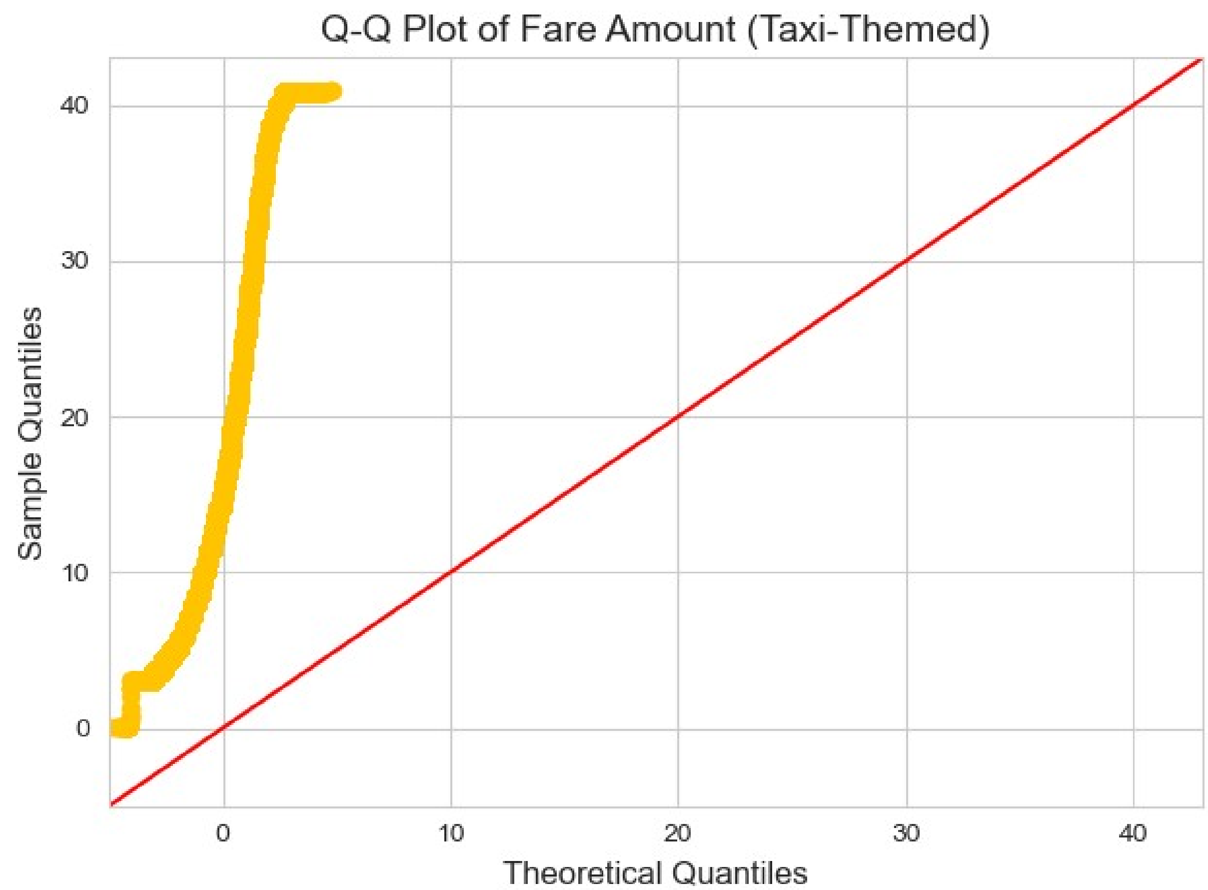
Objective:

To test whether there is a statistically significant difference in fare amounts between rides paid by card and rides paid by cash.

Null Hypothesis: The mean fare is the same for both payment types (card vs. cash).

Alternative Hypothesis: The mean fare differs between the two payment types.

```python
# 1. Seaborn style for a clean background
sns.set_style("whitegrid")

# 2. Create the Q-Q plot figure
fig = sm.qqplot(df['fare_amount'], line='45')

# 3. Customize the plot
ax = fig.axes[0]

# The Q-Q plot typically has two line objects:
#   - line[0]: the data points
#   - line[1]: the 45° reference line
# Depending on statsmodels version, the data might be a scatter
object.

# Try to color the data points in a taxi-yellow
points = ax.get_lines()[0]
points.set_markerfacecolor("#FFC300")  # bright taxi yellow
points.set_markeredgecolor("#FFC300")

# Optionally, change the reference line color to black or red
ref_line = ax.get_lines()[1]
ref_line.set_color("red")
ref_line.set_linewidth(1.5)

# 4. Add a title & labels
ax.set_title("Q-Q Plot of Fare Amount (Taxi-Themed)", fontsize=14)
ax.set_xlabel("Theoretical Quantiles", fontsize=12)
ax.set_ylabel("Sample Quantiles", fontsize=12)

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



Q-Q Plot Purpose: Provides a formal visual check of normality beyond histograms. Points near the line suggest normality; large deviations (particularly in tails) indicate skewness or heavy tails.

Statistical Test Recommendation: Since fare_amount is skewed and not normal, a two-sample t-test is not ideal (unless I log-transform this data). A nonparametric test like Mann–Whitney U (also known as Wilcoxon rank-sum) is more appropriate for comparing fare amounts between two groups (e.g., card vs. cash).

Methodology

Since our fare data is highly skewed (as confirmed by our histograms and Q-Q plots), we opted for a nonparametric test. We used the Mann–Whitney U test because:

It does not assume normality. It is appropriate for comparing two independent groups (card vs. cash).

```
# 1. Extract fare data for each payment type
card_fares = df.loc[df['payment_type'] == 'card',
'fare_amount'].dropna()
cash_fares = df.loc[df['payment_type'] == 'cash',
'fare_amount'].dropna()

# 2. Perform Mann–Whitney U test (two-sided)
```

```
stat, p_value = mannwhitneyu(card_fares, cash_fares, alternative='two-sided')

# 3. Significance level
alpha = 0.05

print("Mann–Whitney U Statistic:", stat)
print("p-value:", p_value)

if p_value < alpha:
    print("Reject H0: There's a significant difference in the fare amounts between card & cash.")
else:
    print("Fail to reject H0: No significant difference in fare amounts.")

Mann–Whitney U Statistic: 211555609179.5
p-value: 0.0
Reject H0: There's a significant difference in the fare amounts
between card & cash.
```

Additionally, I computed the median fare for each group to understand the direction of any difference:

```
median_card = card_fares.median()
median_cash = cash_fares.median()

print(f"Median Fare (Card): {median_card:.2f}")
print(f"Median Fare (Cash): {median_cash:.2f}")

Median Fare (Card): 15.60
Median Fare (Cash): 13.50
```

Based on the results:

median_card > median_cash: It suggests that rides paid by card tend to have higher fares.

Regression Analysis: Modeling Fare Amount Based on Trip Duration and Payment Type

```
import statsmodels.formula.api as smf
# linear regression formula:
model = smf.ols("fare_amount ~ Duration + C(payment_type)", data=df).fit()
print(model.summary())

                            OLS Regression Results

==============================================================================
```

```
Dep. Variable:             fare_amount   R-squared:                       0.693
Model:                             OLS   Adj. R-squared:                  0.693
Method:                  Least Squares   F-statistic:                 1.579e+06
Date:                 Sun, 02 Mar 2025   Prob (F-statistic):               0.00
Time:                         18:44:04   Log-Likelihood:            -4.1068e+06
No. Observations:              1396856   AIC:                         8.214e+06
Df Residuals:                  1396853   BIC:                         8.214e+06
Df Model:                            2

Covariance Type:             nonrobust

=====================================================================================
                              coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
Intercept                   2.8773      0.009    304.508      0.000       2.859       2.896
C(payment_type)[T.cash]    -0.2782      0.009    -30.516      0.000      -0.296      -0.260
Duration                    0.9755      0.001   1758.636      0.000       0.974       0.977
==============================================================================
Omnibus:                482601.046   Durbin-Watson:                   1.678
Prob(Omnibus):               0.000   Jarque-Bera (JB):          1737563.933
Skew:                        1.739   Prob(JB):                         0.00
Kurtosis:                    7.214   Cond. No.                         44.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Conclusion

linear regression suggests:

Trip Duration is the biggest driver of fare (≈ $ 0.98 ≈$0.98 per minute).

Cash rides are $0.28 cheaper than card rides, on average, controlling for duration.

Model Overview

In a regression model, I predict fare_amount using two predictors:

Duration: The trip duration (in minutes)

Payment Type: Categorical variable (card vs. cash)—where the model includes a dummy variable for cash rides.

The model's output shows:

Intercept:

$2.88 (the estimated fare for a card ride when Duration = 0).

Coefficient for Duration (0.9755):

For every additional minute of the ride, the fare increases by roughly $0.98.

Coefficient for Payment Type [Cash] (-0.2782): Controlling for trip duration, rides where cash is used tend to have fares that are, on average, $0.28 lower than those where a card is used.

What Does "Controlling for Duration" Mean?

It means that when comparing two trips of equal duration, the trip paid by cash is estimated to be $0.28 cheaper than the one paid by card. This is an average effect found by the model, not a direct measurement of frequency. So even if cash rides have a lower fare on average, if card rides occur much more frequently (which your univariate analyses indicate), then overall revenue might still be driven by card transactions.

Model Performance

R-squared: About 0.693, which means roughly 69% of the variance in fare_amount is explained by trip duration and payment type. Statistical Significance: The predictors (duration and payment type) are highly significant, suggesting these are important factors.

Limitations & Suggestions

Residual Non-Normality: The Q-Q plot indicates that the residuals from the regression model are not normally distributed. In future analyses, consider applying a transformation (e.g., log-transforming fare_amount) or using robust regression techniques to better handle the skewed data.

Additional Predictors: While the current model includes Duration and Payment Type, incorporating further predictors—such as trip_distance, time-of-day, or passenger_count—could refine the model and provide deeper insights into the factors influencing fare amounts.

Recommendation

Based on our comprehensive analysis—including hypothesis testing and regression modeling—we observe the following:

Hypothesis Testing Results: Our Mann–Whitney U test indicated that the difference in fare amounts between card and cash payments is statistically significant. This confirms that the observed difference is unlikely due to chance.

Regression Analysis Findings: The regression model shows that, controlling for trip duration, cash rides are on average about $0.28 cheaper than card rides. Furthermore, the model's high R-squared suggests that factors like trip duration and payment type play a substantial role in determining fare amounts.

Actionable Recommendations:

Incentivize Card Payments:

Given that card payments are associated with higher fares and are used overwhelmingly in the dataset, the company should consider promoting cashless transactions through loyalty programs, targeted promotions, or small discounts on digital payments.

Further Analyze Ride Characteristics:

Investigate additional variables such as trip distance, time-of-day, and passenger count to understand if card rides are linked to longer trips or occur during peak times, which could explain the higher fares.

Invest in Digital Infrastructure:

With the majority of rides using card payments, enhancing digital transaction systems could improve operational efficiency and customer satisfaction, leading to long-term revenue growth.

In summary, both our hypothesis testing and regression analysis support the strategy of promoting card payments as a means to boost revenue. This targeted approach, combined with further exploration of ride characteristics, can help the taxi company optimize pricing and service delivery.