

Maximizing Revenue for Taxi Cab 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 maximise 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 generate more cash. 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 matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')
```

Loading the data

```
taxi_data = pd.read_csv("Taxi.csv")
taxi_data.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime
passenger_count \			
0	1.0	2020-01-01 00:28:15	2020-01-01 00:33:03
1.0			
1	1.0	2020-01-01 00:35:39	2020-01-01 00:43:04
1.0			
2	1.0	2020-01-01 00:47:41	2020-01-01 00:53:52
1.0			
3	1.0	2020-01-01 00:55:23	2020-01-01 01:00:14
1.0			
4	2.0	2020-01-01 00:01:58	2020-01-01 00:04:16
1.0			

	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
DOLocationID \				
0	1.2	1.0	N	238
239				
1	1.2	1.0	N	239
238				
2	0.6	1.0	N	238
238				
3	0.8	1.0	N	238
151				
4	0.0	1.0	N	193
193				

	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount
\						
0	1.0	6.0	3.0	0.5	1.47	0.0
1	1.0	7.0	3.0	0.5	1.50	0.0
2	1.0	6.0	3.0	0.5	1.00	0.0
3	1.0	5.5	0.5	0.5	1.36	0.0
4	2.0	3.5	0.5	0.5	0.00	0.0

	improvement_surcharge	total_amount	congestion_surcharge
0	0.3	11.27	2.5
1	0.3	12.30	2.5
2	0.3	10.80	2.5
3	0.3	8.16	0.0
4	0.3	4.80	0.0

taxi_data.tail()

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime
passenger_count \			

6405003	NaN	2020-01-31	22:51:00	2020-01-31	23:22:00
NaN					
6405004	NaN	2020-01-31	22:10:00	2020-01-31	23:26:00
NaN					
6405005	NaN	2020-01-31	22:50:07	2020-01-31	23:17:57
NaN					
6405006	NaN	2020-01-31	22:25:53	2020-01-31	22:48:32
NaN					
6405007	NaN	2020-01-31	22:44:00	2020-01-31	23:06:00
NaN					

	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	\
6405003	3.24	NaN	NaN	237	
6405004	22.13	NaN	NaN	259	
6405005	10.51	NaN	NaN	137	
6405006	5.49	NaN	NaN	50	
6405007	11.60	NaN	NaN	179	

	DOLocationID	payment_type	fare_amount	extra	mta_tax
tip_amount \					
6405003	234	NaN	17.59	2.75	0.5
0.0					
6405004	45	NaN	46.67	2.75	0.5
0.0					
6405005	169	NaN	48.85	2.75	0.0
0.0					
6405006	42	NaN	27.17	2.75	0.0
0.0					
6405007	205	NaN	54.56	2.75	0.5
0.0					

	tolls_amount	improvement_surcharge	total_amount	\
6405003	0.00	0.3	21.14	
6405004	12.24	0.3	62.46	
6405005	0.00	0.3	51.90	
6405006	0.00	0.3	30.22	
6405007	0.00	0.3	58.11	

	congestion_surcharge
6405003	0.0
6405004	0.0
6405005	0.0
6405006	0.0
6405007	0.0

Explortatory Data Analysis

Dataset Description

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

```

# rows and columns of the dataset
taxi_data.shape

(6405008, 18)

# Calculating duration from the pickup and dropoff datetime in minutes

# converting pickup and dropoff to datetime
taxi_data['tpep_pickup_datetime'] =
pd.to_datetime(taxi_data['tpep_pickup_datetime'])
taxi_data['tpep_dropoff_datetime'] =
pd.to_datetime(taxi_data['tpep_dropoff_datetime'])

# subtracting the pickup time from dropoff time to get duration
taxi_data['duration'] = taxi_data['tpep_dropoff_datetime'] -
taxi_data['tpep_pickup_datetime']

# converting into minutes
taxi_data['duration'] = taxi_data['duration'].dt.total_seconds()/60

# datatypes of the data
taxi_data.dtypes

```

VendorID	float64
tpep_pickup_datetime	datetime64[ns]
tpep_dropoff_datetime	datetime64[ns]
passenger_count	float64
trip_distance	float64
RatecodeID	float64
store_and_fwd_flag	object
PULocationID	int64
DOLocationID	int64
payment_type	float64
fare_amount	float64
extra	float64
mta_tax	float64
tip_amount	float64
tolls_amount	float64
improvement_surcharge	float64
total_amount	float64
congestion_surcharge	float64
duration	float64
dtype:	object

There are so many columns in the dataset, but as per our problem statement, we only require some fields from the original data. Rest columns are nothing but the unwanted columns for this study. So we will simply remove those columns.

As the problem statement is revolving around the payment type, fare amount, and any other factor influencing the fare amount, we will filter the data to have only passenger count, trip distance, payment type, fare amount and duration of the trip.

```
# removing unwanted columns
taxi_data.drop(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
               'RatecodeID', 'store_and_fwd_flag', 'PULocationID', 'DOLocationID', 'extra',
               'mta_tax', 'tip_amount', 'tolls_amount',
               'improvement_surcharge',
               'total_amount', 'congestion_surcharge'],
               axis = 1, inplace = True)
```

```
# filtered data with relevant columns essential for the analysis
taxi_data.head()
```

	passenger_count	trip_distance	payment_type	fare_amount	duration
0	1.0	1.2	1.0	6.0	4.800000
1	1.0	1.2	1.0	7.0	7.416667
2	1.0	0.6	1.0	6.0	6.183333
3	1.0	0.8	1.0	5.5	4.850000
4	1.0	0.0	2.0	3.5	2.300000

```
# check for missing values
taxi_data.isnull().sum()
```

```
passenger_count    65441
trip_distance        0
payment_type       65441
fare_amount         0
duration            0
dtype: int64
```

```
# percentage of missing data to the total record of the data
print('Missing data %', round(65441/taxi_data.shape[0]*100,2))
```

```
Missing data % 1.02
```

```
# as the data has only 1% of data missing, we can simply drop the
records with missing values.
```

```
taxi_data.dropna(inplace = True)
```

```
# changing the passenger count and payment type data type to integer
as they are present in float
```

```
taxi_data['passenger_count'] =
taxi_data['passenger_count'].astype('int64')
taxi_data['payment_type'] = taxi_data['payment_type'].astype('int64')
```

```
# check for the duplicate rows
```

```
taxi_data[taxi_data.duplicated()]
```

	passenger_count	trip_distance	payment_type	fare_amount	duration
--	-----------------	---------------	--------------	-------------	----------

2056	1	0.00	2	7.0
0.000000				
2441	1	0.00	1	52.0
0.200000				
2446	2	1.70	1	9.5
13.066667				
2465	1	0.40	1	4.0
3.083333				
3344	1	1.20	1	6.0
5.350000				
...
...				
6339558	1	1.63	2	8.0
8.800000				
6339559	1	1.81	1	8.5
8.016667				
6339560	1	0.98	2	6.5
6.900000				
6339562	1	2.10	1	11.0
14.233333				
6339565	1	1.61	2	8.5
9.633333				

[3331706 rows x 5 columns]

removing duplicate rows as they will not contribute in analysis
taxi_data.drop_duplicates(inplace = True)

after removing missing values and duplicate rows, now we have this much records left

taxi_data.shape

(3007861, 5)

passenger count distribution

taxi_data['passenger_count'].value_counts(normalize = True)

passenger_count

1	0.581981
2	0.190350
3	0.066360
5	0.062937
6	0.039272
4	0.036046
0	0.023033
7	0.000009
9	0.000006
8	0.000006

Name: proportion, dtype: float64


```
# payment type distribution
taxi_data['payment_type'].value_counts()

payment_type
1      2040133
2       925137
3       26233
4       16357
5           1
Name: count, dtype: int64
```

We will focus solely on payment types 'card' and 'cash,' denoted by 1 and 2 in the dataset. To ensure our analysis centers on these payment methods, we'll filter the data accordingly, excluding all other types.

Moreover, examining the distribution of passenger counts reveals that trips with more than 5 passengers are rare. Additionally, trips with 0 passengers are impossible, as we expect at least one passenger to pay the fare amount. Therefore, we will filter the passenger count to include only values ranging from 1 to 5.

```
# filtering for payment type 1 and 2
taxi_data = taxi_data[taxi_data['payment_type']<3]

# filtering for passenger count from 1 to 2
taxi_data =
taxi_data[(taxi_data['passenger_count']>0)&(taxi_data['passenger_count']<6)]

# replacing the payment type encoded value 1 and 2 to Card and Cash
taxi_data['payment_type'].replace([1,2],['Card','Cash'], inplace =
True)

# descriptive statistics for data
taxi_data.describe()
```

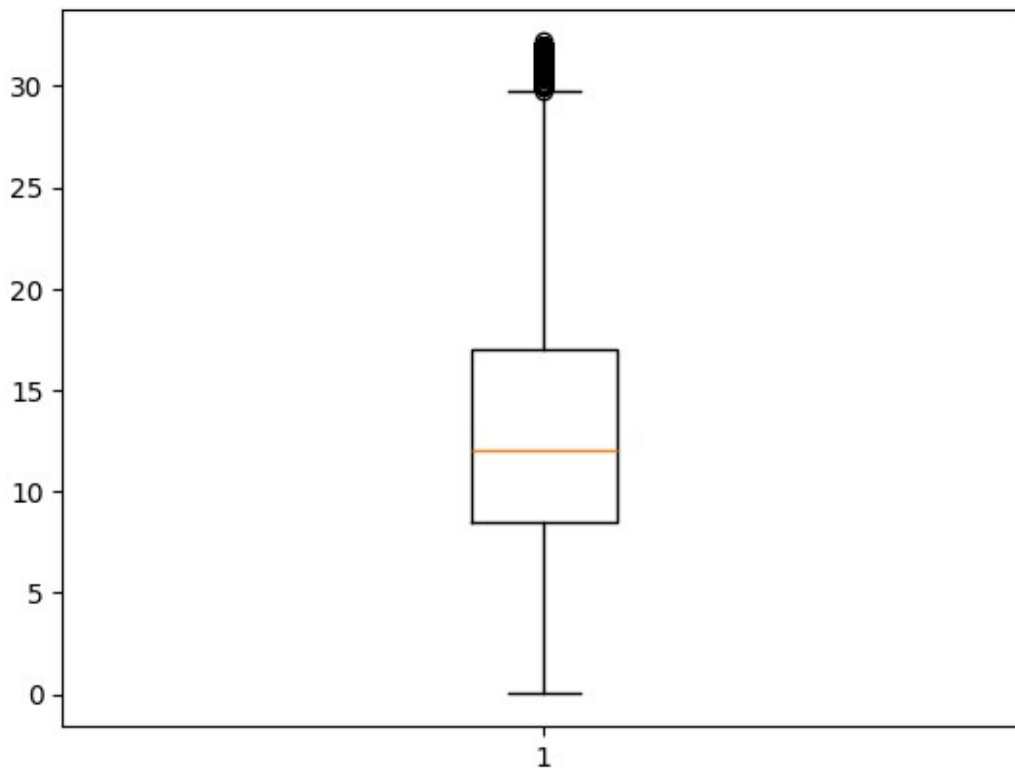
	passenger_count	trip_distance	fare_amount	duration
count	2.780283e+06	2.780283e+06	2.780283e+06	2.780283e+06
mean	1.733386e+00	4.536729e+00	1.780567e+01	2.415478e+01
std	1.176652e+00	4.895890e+00	1.506997e+01	9.260031e+01
min	1.000000e+00	-2.218000e+01	-5.000000e+02	-2.770367e+03
25%	1.000000e+00	1.500000e+00	9.000000e+00	9.883333e+00
50%	1.000000e+00	2.730000e+00	1.300000e+01	1.573333e+01
75%	2.000000e+00	5.470000e+00	2.100000e+01	2.336667e+01
max	5.000000e+00	2.628800e+02	4.265000e+03	8.525117e+03

Upon reviewing the provided statistics, it's evident that the minimum values for trip distance, fare amount, and duration are negative, which is unrealistic and invalid for further analysis. Consequently, we will eliminate these negative values from the dataset.

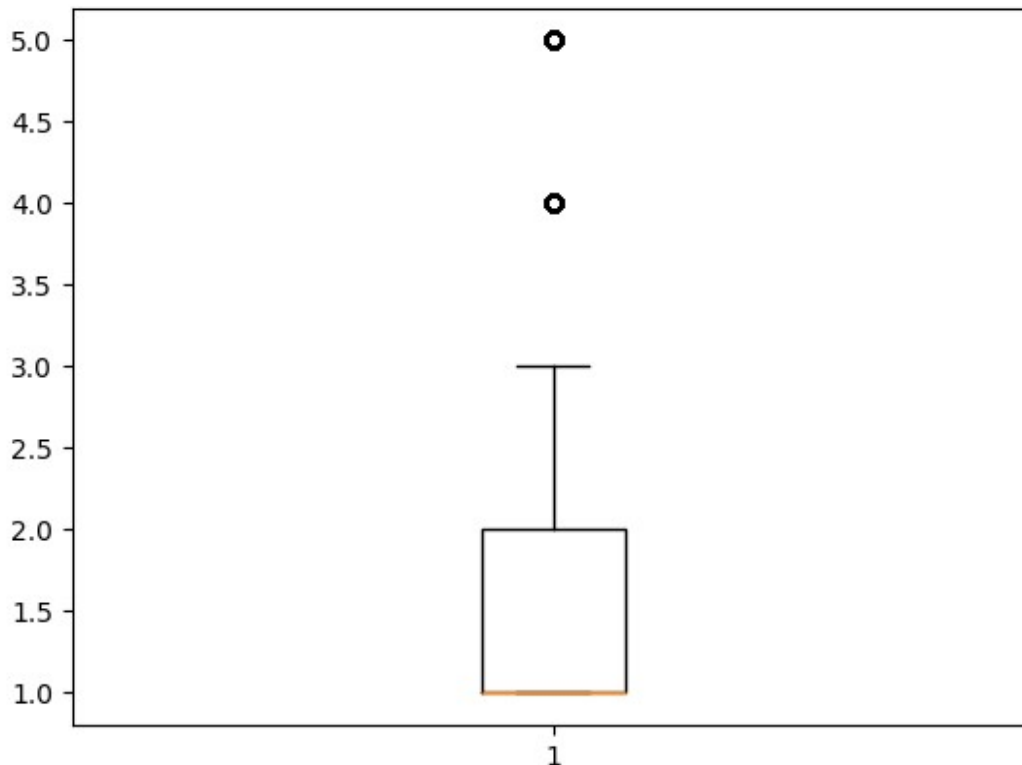
Furthermore, observing the maximum and 50th percentile values, it's possible that the data contains significant outliers, particularly high values. These outliers need to be addressed and removed to ensure the integrity of the analysis.

```
# filtering the records for only positive values
taxi_data = taxi_data[taxi_data['fare_amount']>0]
taxi_data = taxi_data[taxi_data['trip_distance']>0]
taxi_data = taxi_data[taxi_data['duration']>0]

# check for the outliers
plt.boxplot(taxi_data['fare_amount'])
plt.show()
```



```
# check for the outliers
plt.boxplot(taxi_data['passenger_count'])
plt.show()
```



```
# removing outliers using interquartile range for the numerical variables
for col in ['trip_distance', 'fare_amount', 'duration']:
    Q1 = taxi_data[col].quantile(0.25)
    Q3 = taxi_data[col].quantile(0.75)
    IQR = Q3 - Q1

    # Define lower and upper bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Filter out outliers
    taxi_data = taxi_data[(taxi_data[col] >= lower_bound) &
(taxi_data[col] <= upper_bound)]
```

We're interested on exploring the relationship between payment type and passenger behavior concerning trip distance and fare amount. Are there variations in the distribution of payment types concerning different fare amounts or trip distances?

To investigate this, we'll plot histograms to visualize the distribution of passenger counts paying with either card or cash. This will also provide stakeholders with insight into fare amount ranges associated with different payment methods.

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
```

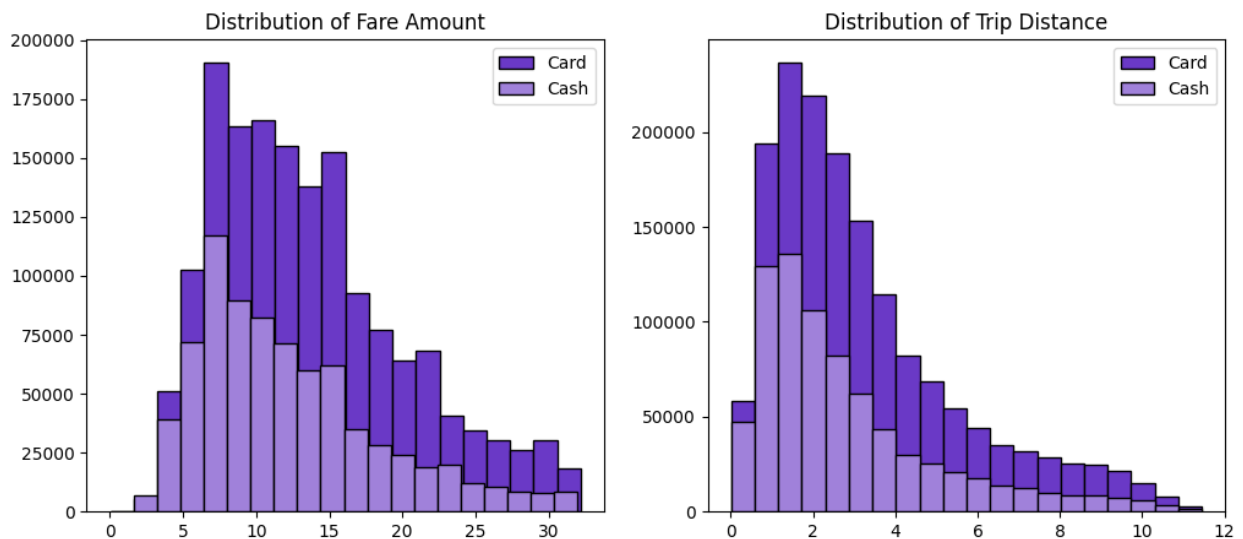
```

plt.title('Distribution of Fare Amount')
plt.hist(taxi_data[taxi_data['payment_type']=='Card']['fare_amount'],
histtype='barstacked', bins = 20, edgecolor = 'k', color = '#6A39C6',
label = 'Card')
plt.hist(taxi_data[taxi_data['payment_type']=='Cash']['fare_amount'],
histtype='barstacked',bins = 20, edgecolor = 'k', color =
'#A081DA',label = 'Cash')
plt.legend()

plt.subplot(1,2,2)
plt.title('Distribution of Trip Distance')
plt.hist(taxi_data[taxi_data['payment_type']=='Card']
['trip_distance'], histtype='barstacked', bins = 20, edgecolor =
'k',color = '#6A39C6',label = 'Card')
plt.hist(taxi_data[taxi_data['payment_type']=='Cash']
['trip_distance'], histtype='barstacked',bins = 20, edgecolor = 'k',
color = '#A081DA',label = 'Cash')
plt.legend()
plt.show()

# calculating the mean and standard deviation group by on payment type
taxi_data.groupby('payment_type').agg({'fare_amount': ['mean',
'std'], 'trip_distance': ['mean', 'std'],})

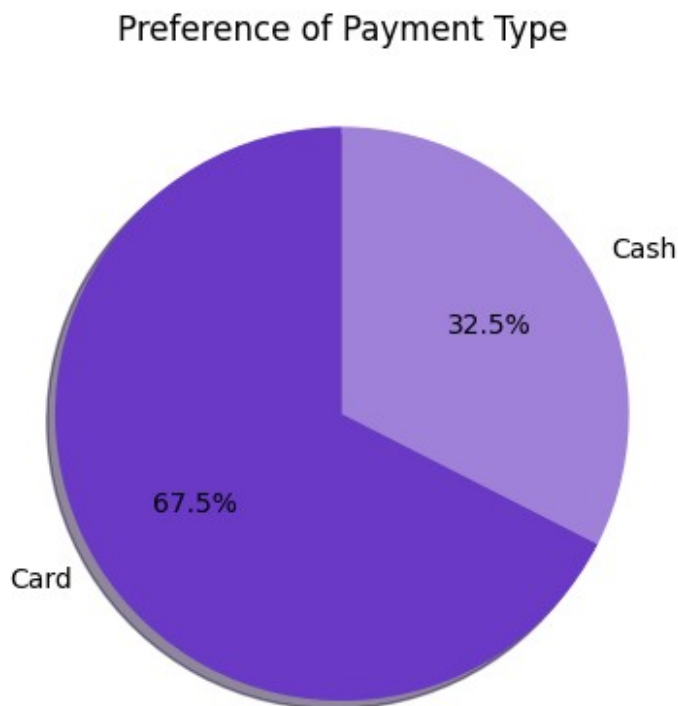
```



	fare_amount		trip_distance	
	mean	std	mean	std
payment_type				
Card	13.701903	6.506356	3.230729	2.320528
Cash	12.250209	6.246104	2.803716	2.231871

Now, in order to examine the passenger's preference regarding their choice of payment method, we will assess the proportion of the two payment types. To provide a visual representation, we have opted to utilize a pie chart. This graphical depiction will offer a clear and intuitive understanding of the distribution between the two payment methods chosen by passengers.

```
plt.title('Preference of Payment Type')
plt.pie(taxi_data['payment_type'].value_counts(normalize = True),
labels = taxi_data['payment_type'].value_counts().index,
        startangle = 90, shadow = True, autopct = '%1.1f%%', colors =
        ['#6A39C6', '#A081DA'])
plt.show()
```



Subsequently, we aim to conduct an analysis of the payment types in relation to the passenger count. Our objective is to investigate if there are any changes in preference contingent upon the number of passengers traveling in the cab.

To facilitate this examination, we have employed a visualization technique known as a stacked bar plot. This method is particularly advantageous for comparing the percentage distribution of each passenger count based on the payment method selected. Through this graphical representation, we can gain insights into potential variations in payment preferences across different passenger counts.

```
# calculating the total passenger count distribution based on the
different payment type
passenger_count =
```

```

taxi_data.groupby(['payment_type', 'passenger_count'])
[['passenger_count']].count()

# renaming the passenger_count to count to reset the index
passenger_count.rename(columns = {'passenger_count': 'count'}, inplace =
True)
passenger_count.reset_index(inplace = True)

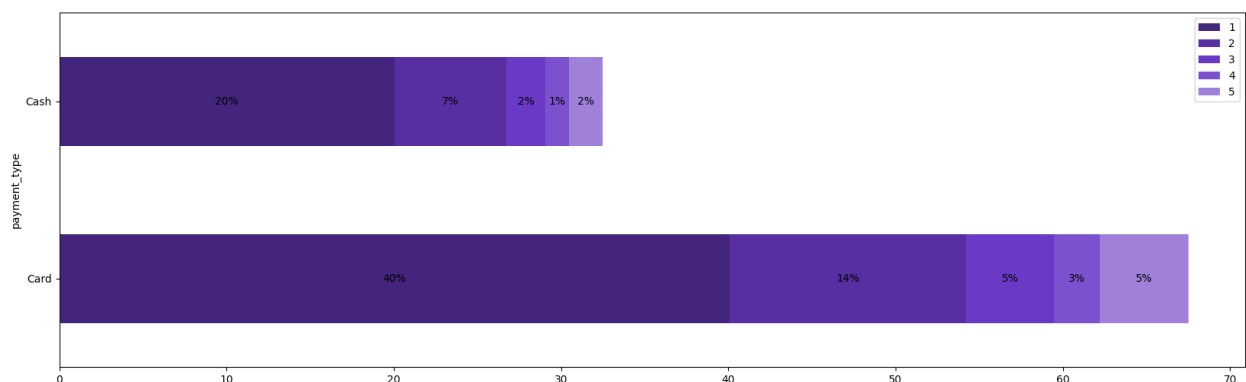
# calculating the percentage of the each passenger count
passenger_count['perc'] =
(passenger_count['count']/passenger_count['count'].sum())*100

# creating a new empty dataframe to store the distribution of each
payment type (useful for the visualization)
df = pd.DataFrame(columns = ['payment_type', 1, 2, 3, 4, 5])
df['payment_type'] = ['Card', 'Cash']
df.iloc[0, 1:] = passenger_count.iloc[:5, -1]
df.iloc[1, 1:] = passenger_count.iloc[5:, -1]

fig, ax = plt.subplots(figsize=(20, 6))
df.plot(x='payment_type', kind='barh', stacked=True, title=' ', ax=ax,
color = ['#44257E', '#572FA2', '#6A39C6', '#7C51CD', '#A081DA'] )

# Add percentage text
for p in ax.patches: # ax.patches → contains all the rectangle objects
(bars) drawn on the axes ax.
    width = p.get_width() # width = p.get_width().
    height = p.get_height() # p.get_height().
    x, y = p.get_xy() # bottom-left corner (x, y) of the bar.
    ax.text(x + width / 2,
            y + height / 2,
            '{:.0f}%'.format(width),
            horizontalalignment='center',
            verticalalignment='center')

```

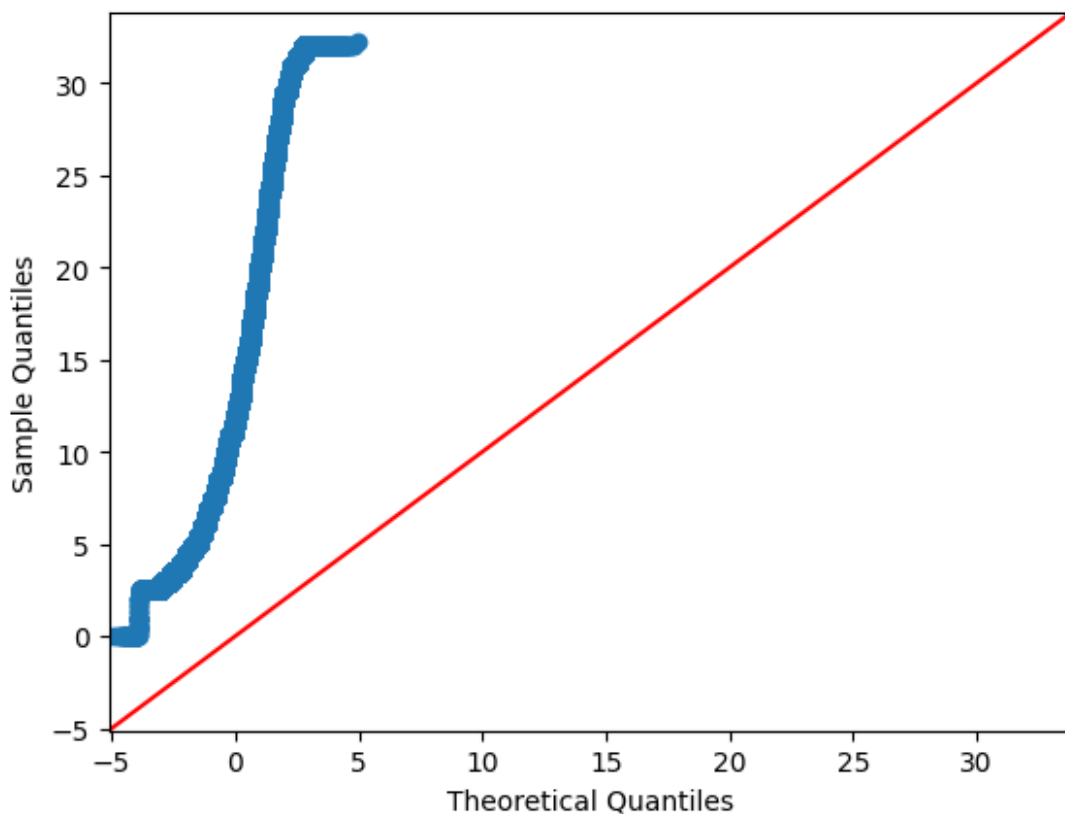


Hypothesis Testing

In order to select the most suitable test for our scenario, our initial step involves evaluating whether the distribution of fare amounts adheres to a normal distribution. While the histogram depicted above suggests otherwise, we will further confirm this by generating a QQ plot.

Quantile-quantile (QQ) plots can be used to assess whether the fare amount distributions for each payment type are approximately normally distributed. If the data points closely align with the diagonal line in the plot, it suggests that the data follows a normal distribution.

```
#create Q-Q plot with 45-degree line added to plot  
fig = sm.qqplot(taxi_data['fare_amount'], line='45')  
plt.show()
```



The data values clearly do not follow the red 45-degree line, which is an indication that they do not follow a normal distribution. So, z distribution will not be good for this. That's why we will use T test.

Given that the T-test can be applied to both small and large samples and does not require the population standard deviation, it is a more universally applicable approach for hypothesis testing in many practical research scenarios, including analyses of taxi trip data.

In the analysis of NYC Yellow Taxi Trip Records, where you're likely dealing with an unknown population standard deviation and potentially large datasets, the T-test offers a more

appropriate and flexible method for comparing means between two groups (e.g., fare amounts by payment type). It provides a reliable way to infer about the population, accommodating the uncertainty that comes with estimating population parameters from sample data.

Null hypothesis: There is no difference in average fare between customers who use credit cards and customers who use cash.

Alternative hypothesis: There is a difference in average fare between customers who use credit cards and customers who use cash

```
# sample 1
credit_card = taxi_data[taxi_data['payment_type'] == 'Card']
['fare_amount']

# sample 2
cash = taxi_data[taxi_data['payment_type'] == 'Cash']['fare_amount']

# performing t test on both the different sample
t_stat, p_value = stats.ttest_ind(a=credit_card, b=cash,
equal_var=False)
print(f"T-statistic: {t_stat}, P-value: {p_value}")

# comparing the p value with the significance of 5% or 0.05
if p_value < 0.05:
    print("\nReject the null hypothesis")
else:
    print("\nAccept the null hypothesis")

T-statistic: 165.59915491544626, P-value: 0.0

Reject the null hypothesis
```

Since the p-value is significantly smaller than the significance level of 5%, we will reject the null hypothesis.

You conclude that there is a statistically significant difference in the average fare amount between customers who use credit cards and customers who use cash.

The key business insight is that encouraging customers to pay with credit cards can generate more revenue for taxi cab drivers.