NYC Taxi Trips – Data Pipeline and Analysis

“NYC Taxi Trips” is a team project I did for my Data Programming course in the Big Data Analytics program at Georgian College. In this project, I developed and implemented a data pipeline to collect, process and store taxi trip data from the New York City Open Data platform. I set up an AWS EC2 instance to host a MongoDB database and use Python to fetch JSON data via an API, enabling real-time retrieval progress reports every 50,000 records. The pipeline is designed to ensure seamless and efficient cloud server data storage for team members to access for further data cleaning and preparation for data analysis. Following data preparation, we collaborate as a team to explore key trends, including peak trip hours, preferred payment methods and high-traffic pickup zones to generate actionable insights.

Table of Contents

Code Base

In [44]:
```
pip install requests
```

```
Requirement already satisfied: requests in /Users/avsapkota/anaconda3/lib/python3.10/site-packages (2.28.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/avsapkota/anaconda3/lib/python3.10/site-packages (from requests) (1.26.14)
Requirement already satisfied: charset-normalizer<3,>=2 in /Users/avsapkota/anaconda3/lib/python3.10/site-packages (from requests) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/avsapkota/anaconda3/lib/python3.10/site-packages (from requests) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in /Users/avsapkota/anaconda3/lib/python3.10/site-packages (from requests) (2022.12.7)
Note: you may need to restart the kernel to use updated packages.
```

In [45]:
```
pip install pymongo
```

```
Requirement already satisfied: pymongo in /Users/avsapkota/anaconda3/lib/python3.10/site-packages (4.4.1)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in /Users/avsapkota/anaconda3/lib/python3.10/site-packages (from pymongo) (2.4.1)
Note: you may need to restart the kernel to use updated packages.
```

In [1]:
```
import requests

#Defines a function 'getData' to retrieve taxi trip data for a given year and month.
def getData(year, month):
    url = "https://data.cityofnewyork.us/resource/kxp8-n2sj.json"

    chunkSize = 1000
    offset = 0
    allData = []
    progressStep = 50000

    while True:
        params = {
            '$limit': chunkSize,
            '$offset': offset,
            '$where': f"tpep_pickup_datetime >= '{year}-{month:02d}-01T00:00:00.000' AND tpep_pickup_datetime < '{year}-{month+1:02d}-01T00:00:00.000'"
        }

        response = requests.get(url, params=params)
        if response.status_code == 200:
            data = response.json()

            if not data:
                break

            allData.extend(data)
            offset += chunkSize


            if offset % progressStep == 0:
                print(f"Retrieved {offset} rows of data.")

        else:
            print("Failed to get data from the NYC TLC API.")
            return None

    return allData


year = 2020
month = 4
taxitripData = getData(year, month)

if taxitripData:
```

```
    print("Taxi trip data acquired successfully!\n")
    print("Total records found:", len(taxitripData))
    print("\n")
    print("Sample data:", taxitripData[0])
```

```
Retrieved 50000 rows of data.
Retrieved 100000 rows of data.
Retrieved 150000 rows of data.
Retrieved 200000 rows of data.
Taxi trip data acquired successfully!

Total records found: 237883


Sample data: {'tpep_pickup_datetime': '2020-04-01T00:00:00.000', 'tpep_dropoff_datetime': '2020-04-01T00:27:00.000', 'trip_distance': '8.68', 'pulocationid': '137', 'dolocationid': '61', 'fare_amoun
t': '30.45', 'extra': '0', 'mta_tax': '0.5', 'tip_amount': '0', 'tolls_amount': '0', 'improvement_surcharge': '0.3', 'total_amount': '33.75', 'congestion_surcharge': '2.5'}
```

In [2]:

```
from pymongo import MongoClient

MongoDBConnectionLink = 'mongodb://BDAT:BDAT@ec2-3-144-229-9.us-east-2.compute.amazonaws.com:27017/admin'

def insertDataDBS(data):
    try:
        client = MongoClient(MongoDBConnectionLink)
        db = client['ProjectDatabase']
        collection = db['TaxiData']
        result = collection.insert_many(data)

        if result.acknowledged:
            print("Data inserted into MongoDB successfully!")
        else:
            print("Failed to insert data into MongoDB.")
    except Exception as e:
        print("Error while connecting to MongoDB:", str(e))

if taxitripData:
    insertDataDBS(taxitripData)
```

```
Data inserted into MongoDB successfully!
```

In [4]:

```
from pymongo import MongoClient
import pandas as pd

MongoDBConnectionLink = 'mongodb://BDAT:BDAT@ec2-3-144-229-9.us-east-2.compute.amazonaws.com:27017/admin'

try:
    client = MongoClient(MongoDBConnectionLink)
    db = client['ProjectDatabase']
    collection = db['TaxiData']

    data_from_mongo = list(collection.find())
    client.close()

    taxi_df = pd.DataFrame(data_from_mongo)

    print("Data Overview:")
    print(taxi_df.head())

    print("\nData Types and Missing Values:")
    print(taxi_df.info())
```

```
    print("\nSummary Statistics:")
    print(taxi_df.describe())

    print("\nUnique Values in Categorical Column: Payment Type")
    print(taxi_df['payment_type'].unique())

except Exception as e:
    print("Error while connecting to MongoDB:", str(e))
```

```
Data Overview:
                       _id     tpep_pickup_datetime    tpep_dropoff_datetime  \
0  64d2a4dc414c6ca20b9210f1  2020-04-01T00:00:00.000  2020-04-01T00:27:00.000
1  64d2a4dc414c6ca20b9210f2  2020-04-01T00:00:15.000  2020-04-01T00:05:04.000
2  64d2a4dc414c6ca20b9210f3  2020-04-01T00:00:26.000  2020-04-01T00:09:25.000
3  64d2a4dc414c6ca20b9210f4  2020-04-01T00:00:28.000  2020-04-01T00:07:59.000
4  64d2a4dc414c6ca20b9210f5  2020-04-01T00:00:36.000  2020-04-01T00:03:32.000

  trip_distance pulocationid dolocationid fare_amount extra mta_tax  \
0          8.68          137           61       30.45     0     0.5
1          2.08          141           74         7.5   0.5     0.5
2          2.80          237          137          10     3     0.5
3          3.25          107          236        10.5   0.5     0.5
4          0.51          263           75           4   0.5     0.5

  tip_amount tolls_amount improvement_surcharge total_amount  \
0          0            0                   0.3        33.75
1          0            0                   0.3         11.3
2          1            0                   0.3         14.8
3          0            0                   0.3         14.3
4          0            0                   0.3          7.8

  congestion_surcharge vendorid passenger_count ratecodeid store_and_fwd_flag  \
0                  2.5      NaN             NaN        NaN                NaN
1                  2.5        2               1          1                  N
2                  2.5        1               1          1                  N
3                  2.5        2               1          1                  N
4                  2.5        2               3          1                  N

  payment_type
0          NaN
1            2
2            1
3            2
4            2

Data Types and Missing Values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 237883 entries, 0 to 237882
Data columns (total 19 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   _id                    237883 non-null  object
 1   tpep_pickup_datetime   237883 non-null  object
 2   tpep_dropoff_datetime  237883 non-null  object
 3   trip_distance          237883 non-null  object
 4   pulocationid           237883 non-null  object
 5   dolocationid           237883 non-null  object
 6   fare_amount            237883 non-null  object
 7   extra                  237883 non-null  object
 8   mta_tax                237883 non-null  object
 9   tip_amount             237883 non-null  object
 10  tolls_amount           237883 non-null  object
 11  improvement_surcharge  237883 non-null  object
 12  total_amount           237883 non-null  object
 13  congestion_surcharge   237883 non-null  object
 14  vendorid               218370 non-null  object
 15  passenger_count        218370 non-null  object
 16  ratecodeid             218370 non-null  object
 17  store_and_fwd_flag     218370 non-null  object
```

```
 18  payment_type           218370 non-null  object
dtypes: object(19)
memory usage: 34.5+ MB
None

Summary Statistics:
                            _id     tpep_pickup_datetime  \
count                    237883                   237883
unique                   237883                   217940
top     64d2a4dc414c6ca20b9210f1  2020-04-29T17:01:00.000
freq                          1                        8

         tpep_dropoff_datetime trip_distance pulocationid dolocationid  \
count                   237883        237883       237883       237883
unique                  218070          2952          250          259
top     2020-04-29T17:27:00.000          0.00          137           75
freq                          8          6221        12006        10792

       fare_amount   extra mta_tax tip_amount tolls_amount  \
count       237883  237883  237883     237883       237883
unique        5111      18       4       1445          102
top              5       0     0.5          0            0
freq         14399  112221  232959     107416       230168

       improvement_surcharge total_amount congestion_surcharge vendorid  \
count                 237883       237883               237883   218370
unique                     3         6672                    3        2
top                      0.3          8.8                  2.5        2
freq                  236263         6920               184302   123320

      passenger_count ratecodeid store_and_fwd_flag payment_type
count          218370     218370             218370       218370
unique              8          7                  2            4
top                 1          1                  N            1
freq           177675     215973             216519       131069

Unique Values in Categorical Column: Payment Type
[nan '2' '1' '4' '3']
```

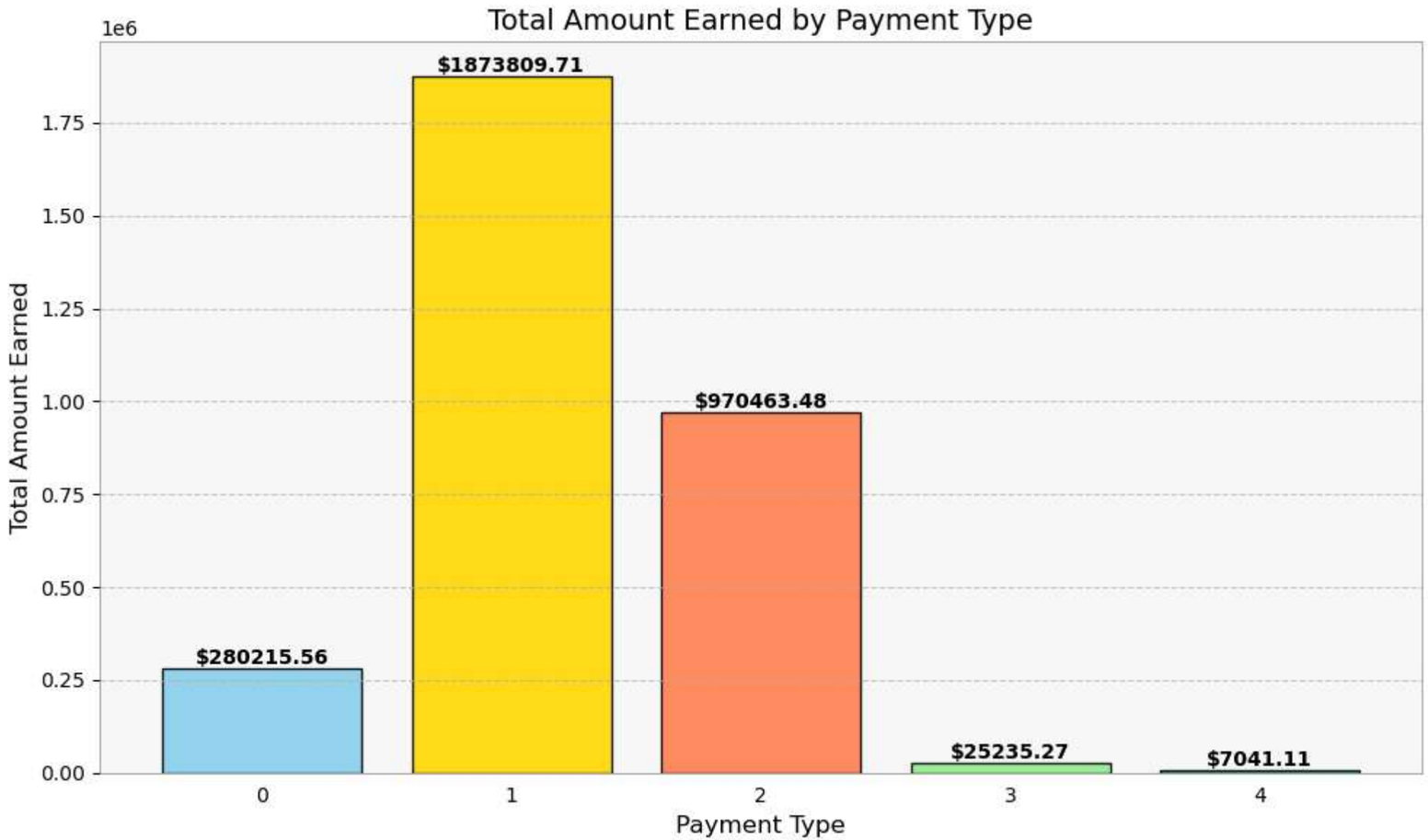
In [5]:
```python
#Data Cleaning and Preparation
try:
    taxi_df['tpep_pickup_datetime'] = pd.to_datetime(taxi_df['tpep_pickup_datetime'])
    taxi_df['tpep_dropoff_datetime'] = pd.to_datetime(taxi_df['tpep_dropoff_datetime'])

    columns_to_convert = ['passenger_count', 'payment_type']
    taxi_df[columns_to_convert] = taxi_df[columns_to_convert].fillna(0).astype('int64')
    convert_cols = [
        'trip_distance', 'pulocationid', 'dolocationid', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
        'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'vendorid',
        'passenger_count', 'ratecodeid', 'store_and_fwd_flag', 'payment_type'
    ]
    taxi_df[convert_cols] = taxi_df[convert_cols].apply(pd.to_numeric, errors='coerce')


    def remove_outliers(column):
        z_scores = (column - column.mean()) / column.std()
        return column[abs(z_scores) <= 3]

    taxi_df['trip_distance'] = remove_outliers(taxi_df['trip_distance'])
    taxi_df['total_amount'] = remove_outliers(taxi_df['total_amount'])
```

```
    taxi_df['trip_duration_minutes'] = (taxi_df['tpep_dropoff_datetime'] - taxi_df['tpep_pickup_datetime']).dt.total_seconds() / 60

    print("Data Overview:")
    print(taxi_df.head())

    print("\nData Types and Missing Values:")
    print(taxi_df.info())

    print("\nSummary Statistics:")
    print(taxi_df.describe())

    print("\nUnique Values in Categorical Columns:")
    print(taxi_df['payment_type'].unique())
    if 'ratecodeid' in taxi_df.columns:
        print(taxi_df['ratecodeid'].unique())
    else:
        print("Column 'ratecodeid' does not exist in the dataset.")

except Exception as e:
    print("An error occured:", str(e))
```

```
Data Overview:
                       _id tpep_pickup_datetime tpep_dropoff_datetime  \
0  64d2a4dc414c6ca20b9210f1  2020-04-01 00:00:00   2020-04-01 00:27:00
1  64d2a4dc414c6ca20b9210f2  2020-04-01 00:00:15   2020-04-01 00:05:04
2  64d2a4dc414c6ca20b9210f3  2020-04-01 00:00:26   2020-04-01 00:09:25
3  64d2a4dc414c6ca20b9210f4  2020-04-01 00:00:28   2020-04-01 00:07:59
4  64d2a4dc414c6ca20b9210f5  2020-04-01 00:00:36   2020-04-01 00:03:32

   trip_distance  pulocationid  dolocationid  fare_amount  extra  mta_tax  \
0           8.68           137            61        30.45    0.0      0.5
1           2.08           141            74         7.50    0.5      0.5
2           2.80           237           137        10.00    3.0      0.5
3           3.25           107           236        10.50    0.5      0.5
4           0.51           263            75         4.00    0.5      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount  \
0         0.0           0.0                    0.3         33.75
1         0.0           0.0                    0.3         11.30
2         1.0           0.0                    0.3         14.80
3         0.0           0.0                    0.3         14.30
4         0.0           0.0                    0.3          7.80

   congestion_surcharge  vendorid  passenger_count  ratecodeid  \
0                   2.5       NaN                0         NaN
1                   2.5       2.0                1         1.0
2                   2.5       1.0                1         1.0
3                   2.5       2.0                1         1.0
4                   2.5       2.0                3         1.0

   store_and_fwd_flag  payment_type  trip_duration_minutes
0                 NaN             0              27.000000
1                 NaN             2               4.816667
2                 NaN             1               8.983333
3                 NaN             2               7.516667
4                 NaN             2               2.933333

Data Types and Missing Values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 237883 entries, 0 to 237882
Data columns (total 20 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   _id                    237883 non-null  object
 1   tpep_pickup_datetime   237883 non-null  datetime64[ns]
 2   tpep_dropoff_datetime  237883 non-null  datetime64[ns]
 3   trip_distance          237879 non-null  float64
 4   pulocationid           237883 non-null  int64
 5   dolocationid           237883 non-null  int64
 6   fare_amount            237883 non-null  float64
 7   extra                  237883 non-null  float64
 8   mta_tax                237883 non-null  float64
 9   tip_amount             237883 non-null  float64
 10  tolls_amount           237883 non-null  float64
 11  improvement_surcharge  237883 non-null  float64
 12  total_amount           233473 non-null  float64
 13  congestion_surcharge   237883 non-null  float64
 14  vendorid               218370 non-null  float64
 15  passenger_count        237883 non-null  int64
 16  ratecodeid             218370 non-null  float64
 17  store_and_fwd_flag     0 non-null       float64
```

```
 18  payment_type           237883 non-null  int64
 19  trip_duration_minutes  237883 non-null  float64
dtypes: datetime64[ns](2), float64(13), int64(4), object(1)
memory usage: 36.3+ MB
None

Summary Statistics:
       trip_distance   pulocationid   dolocationid    fare_amount  \
count  237879.000000  237883.000000  237883.000000  237883.000000
mean        3.010223     154.909115     150.358638      11.665936
std         3.897117      70.752046      74.476382      11.729592
min         0.000000       1.000000       1.000000    -118.000000
25%         0.950000      97.000000      75.000000       5.500000
50%         1.740000     143.000000     143.000000       8.000000
75%         3.400000     234.000000     233.000000      13.000000
max       177.400000     265.000000     265.000000     903.020000

              extra        mta_tax     tip_amount   tolls_amount  \
count  237883.000000  237883.000000  237883.000000  237883.000000
mean        1.066982       0.486994       1.529985       0.220426
std         1.260374       0.095015       2.295684       1.342272
min        -4.500000      -0.500000      -5.000000     -19.870000
25%         0.000000       0.500000       0.000000       0.000000
50%         0.500000       0.500000       1.000000       0.000000
75%         2.500000       0.500000       2.460000       0.000000
max         7.000000       1.100000     117.280000      98.750000

       improvement_surcharge   total_amount  congestion_surcharge  \
count          237883.000000  233473.000000         237883.000000
mean                0.296329      15.365056              1.927387
std                 0.045439       9.055289              1.072943
min                -0.300000     -22.800000             -2.500000
25%                 0.300000       9.800000              2.500000
50%                 0.300000      12.800000              2.500000
75%                 0.300000      17.890000              2.500000
max                 0.300000      55.870000              2.500000

            vendorid  passenger_count     ratecodeid  store_and_fwd_flag  \
count  218370.000000    237883.000000  218370.000000                 0.0
mean        1.564730         1.190619       1.034084                 NaN
std         0.495794         1.007683       0.865255                 NaN
min         1.000000         0.000000       1.000000                 NaN
25%         1.000000         1.000000       1.000000                 NaN
50%         2.000000         1.000000       1.000000                 NaN
75%         2.000000         1.000000       1.000000                 NaN
max         2.000000         7.000000      99.000000                 NaN

        payment_type  trip_duration_minutes
count  237883.000000          237883.000000
mean        1.308807              11.295633
std         0.660909              44.841211
min         0.000000               0.000000
25%         1.000000               4.450000
50%         1.000000               7.500000
75%         2.000000              12.550000
max         4.000000            3574.700000

Unique Values in Categorical Columns:
[0 2 1 4 3]
[nan  1.  5.  4.  2.  3. 99.  6.]
```

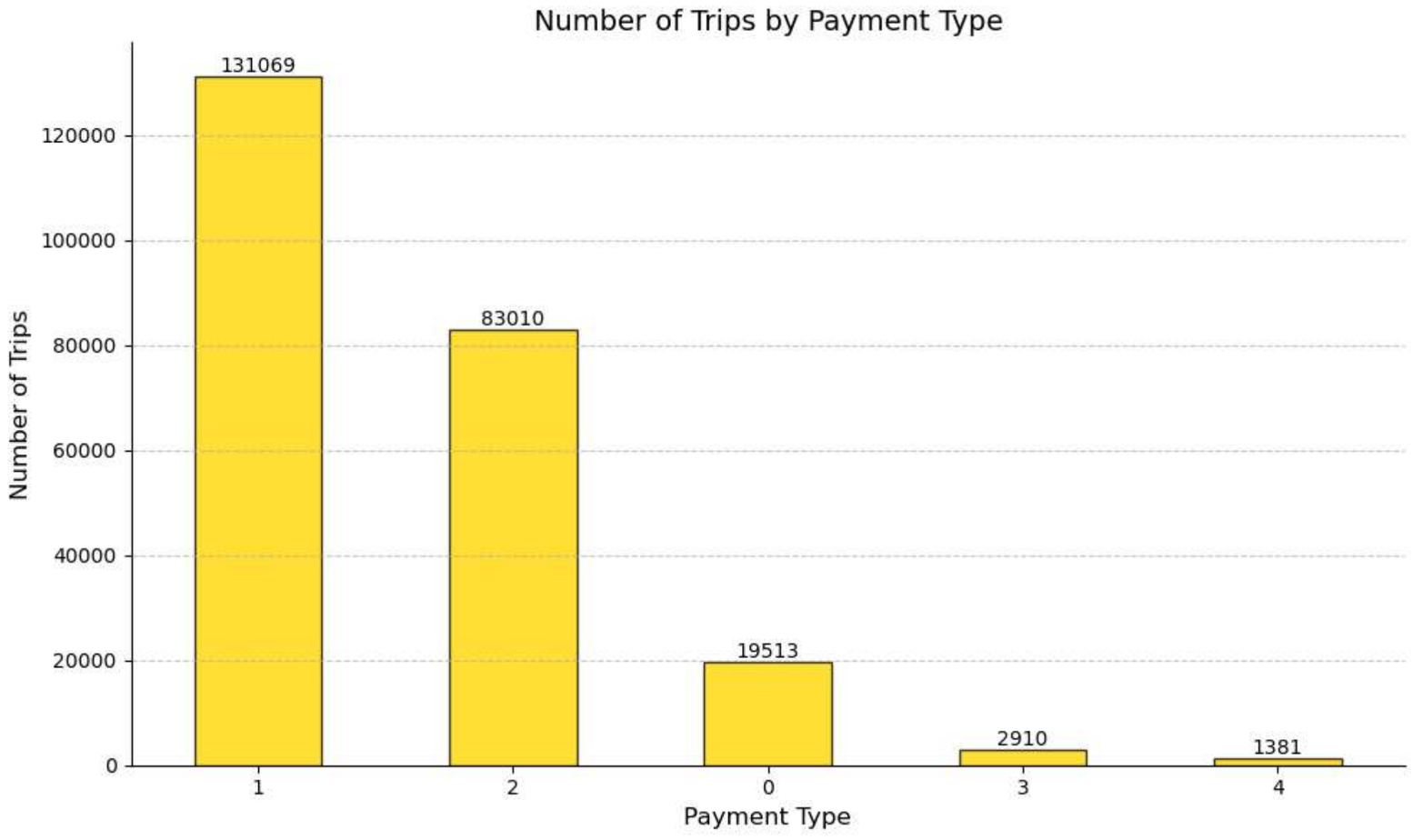
In [75]:
```
import matplotlib.pyplot as plt

payment_type_totals = taxi_df.groupby('payment_type')['total_amount'].sum()

colors = ['skyblue', 'gold', 'coral', 'lightgreen', 'lightseagreen']
plt.figure(figsize=(10, 6))

ax = payment_type_totals.plot(kind='bar', color=colors, edgecolor='black', alpha=0.9, width=0.8)

plt.xlabel('Payment Type', fontsize=12)
plt.ylabel('Total Amount Earned', fontsize=12)
plt.title('Total Amount Earned by Payment Type', fontsize=14)
plt.xticks(rotation=0, fontsize=10)
plt.yticks(fontsize=10)

for index, value in enumerate(payment_type_totals):
    plt.text(index, value, f"${round(value, 2)}", ha='center', va='bottom', fontsize=10, color='black', fontweight='bold')

plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.gca().set_facecolor('#f7f7f7')

for spine in plt.gca().spines.values():
    spine.set_visible(True)
    spine.set_linewidth(0.5)
    spine.set_color('grey')

plt.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=True)
plt.tick_params(axis='y', which='both', left=True, right=False, labelleft=True)

plt.tight_layout()

# Displays the generated plot
plt.show()
```



In [ ]:

In [60]:
```
import matplotlib.pyplot as plt

#Creates a bar chart depicting the distribution of payment types for taxi trips
plt.figure(figsize=(10, 6))

payment_counts = taxi_df['payment_type'].value_counts()
payment_counts.plot(kind='bar', color='gold', edgecolor='black', alpha=0.8)

plt.xlabel('Payment Type', fontsize=12)
plt.ylabel('Number of Trips', fontsize=12)
plt.title('Number of Trips by Payment Type', fontsize=14)
plt.xticks(rotation=0, fontsize=10)
plt.yticks(fontsize=10)

for index, value in enumerate(payment_counts):
    plt.text(index, value, str(value), ha='center', va='bottom', fontsize=10)

plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
```

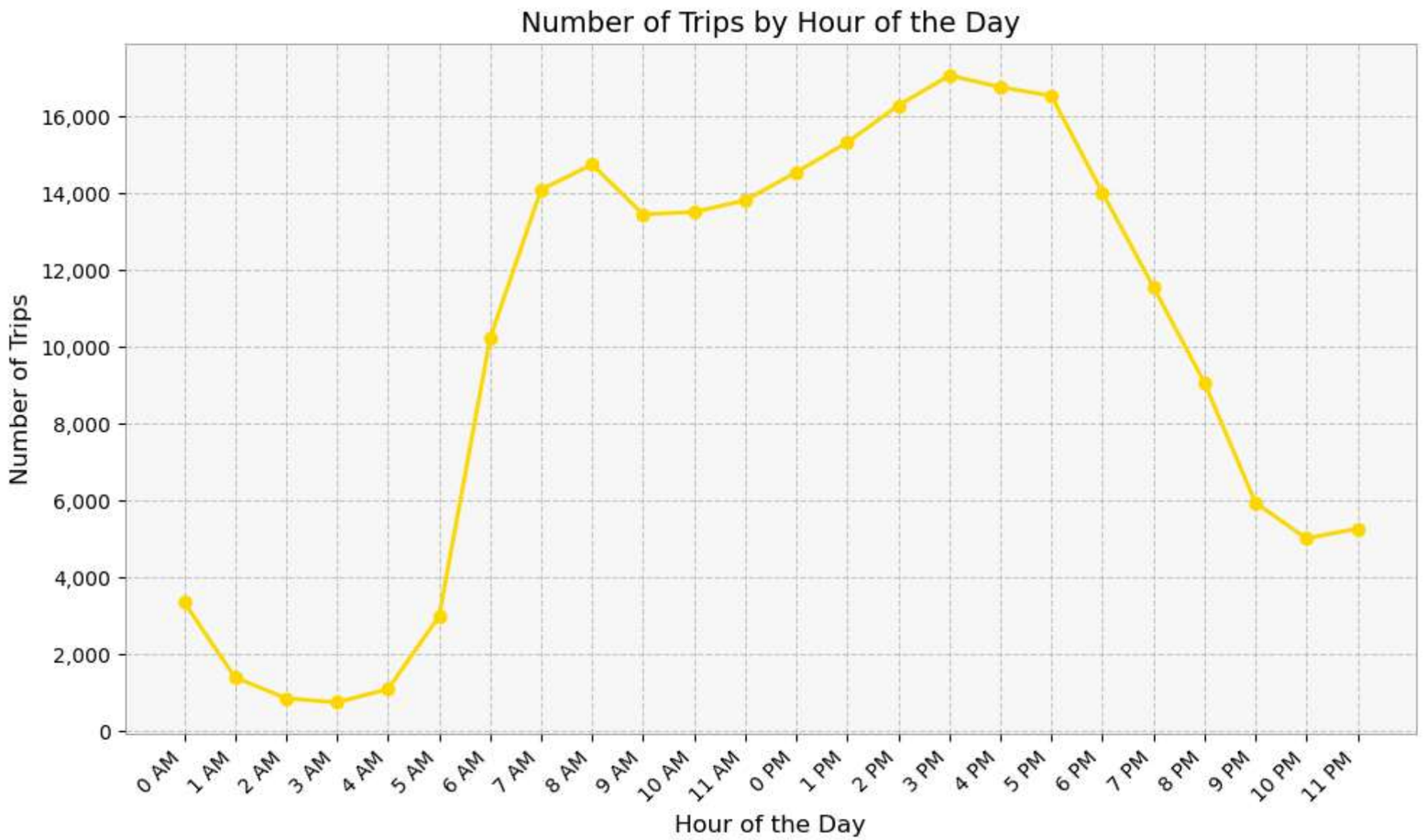
```
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()

# Shows the generated plot
plt.show()
```



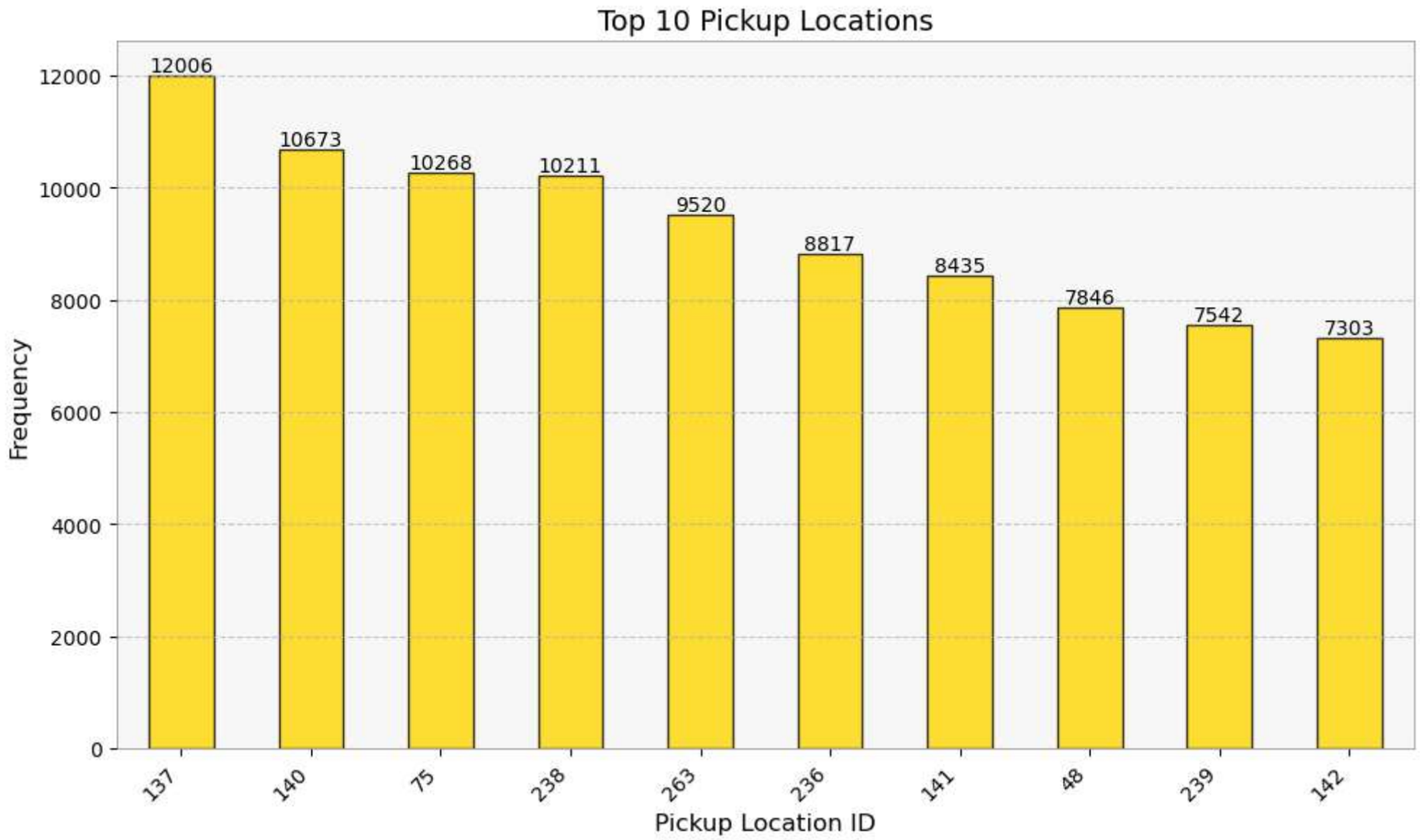
In [ ]:

In [76]:
```
import matplotlib.pyplot as plt

taxi_df['pickup_hour'] = taxi_df['tpep_pickup_datetime'].dt.hour

hourly_trip_counts = taxi_df['pickup_hour'].value_counts().sort_index()

plt.figure(figsize=(10, 6))

hourly_trip_counts.plot(kind='line', marker='o', color='gold', linestyle='-', linewidth=2)

plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Number of Trips', fontsize=12)
plt.title('Number of Trips by Hour of the Day', fontsize=14)
```

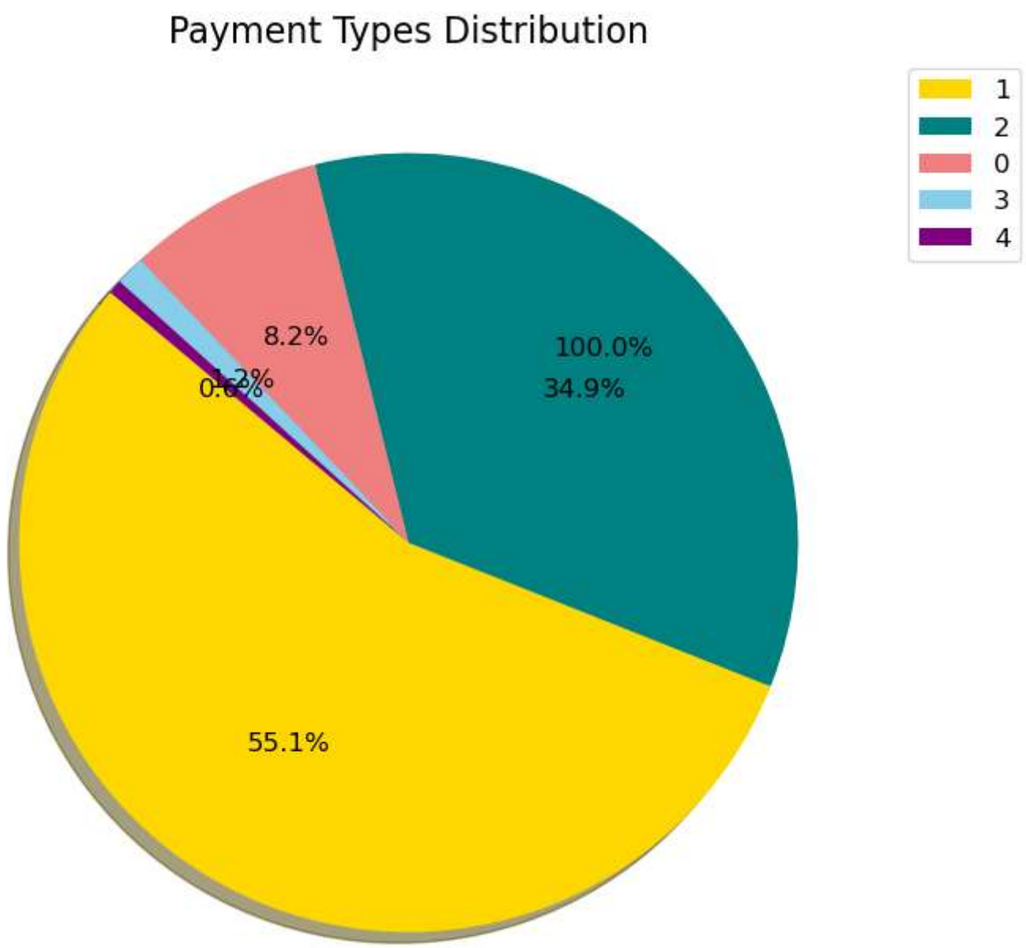
```
hour_labels = [f'{h % 12} AM' if h < 12 else f'{h % 12} PM' for h in range(24)]
plt.xticks(range(24), hour_labels, fontsize=10, rotation=45, ha='right')  # Adjust rotation and alignment here

plt.gca().get_yaxis().set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:,}".format(int(x))))
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.grid(True, linestyle='--', alpha=0.7)

plt.gca().set_facecolor('#f7f7f7')

for spine in plt.gca().spines.values():
    spine.set_visible(True)
    spine.set_linewidth(0.5)
    spine.set_color('grey')

plt.tight_layout()

#Displays the generated plot
plt.show()
```



In [77]:
```
import matplotlib.pyplot as plt

top_pickup_locations = taxi_df['pulocationid'].value_counts().nlargest(10)
plt.figure(figsize=(10, 6))
top_pickup_locations.plot(kind='bar', color='gold', edgecolor='black', alpha=0.8)

plt.xlabel('Pickup Location ID', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Top 10 Pickup Locations', fontsize=14)

plt.xticks(rotation=45, ha='right', fontsize=10)

for index, value in enumerate(top_pickup_locations):
    plt.text(index, value, str(value), ha='center', va='bottom', fontsize=10, color='black')

plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.gca().set_facecolor('#f7f7f7')

for spine in plt.gca().spines.values():
    spine.set_visible(True)
    spine.set_linewidth(0.5)
    spine.set_color('grey')

plt.tight_layout()

plt.show()
```



In [78]:
```
import matplotlib.pyplot as plt

payment_counts = taxi_df['payment_type'].value_counts()
colors = ['gold', 'teal', 'lightcoral', 'skyblue', 'purple']

plt.figure(figsize=(8, 8))
plt.pie(payment_counts, labels=None, autopct='%1.1f%%', colors=colors, shadow=True, startangle=140, textprops={'fontsize': 12})
plt.title('Payment Types Distribution', fontsize=16)

plt.gca().set_aspect('equal')
plt.gca().set_autoscale_on(False)
plt.gca().text(0.5, 0.5, '{:.1f}%'.format(sum(payment_counts) / payment_counts.sum() * 100), fontsize=12, ha='center', va='center')

plt.legend(payment_counts.index, loc='upper left', bbox_to_anchor=(1, 1), fontsize=12)

plt.show()
```



In [ ]:

Project Report

FINAL PROJECT REPORT

Abstract

An exploratory analysis on sample data collated from taxi trips taken within New York City

Group 3

Contents

Description of the Dataset

Source

The dataset we utilized in our analysis is sourced from the New York City Open Data platform. It comprises a subset of taxi trips taken within the city collated by the yellow taxi service providers in the month of April in 2020 and includes essential trip details such as pickup, drop-off locations, trip duration, fare amount, and more.

About

There are 17 variables and 237,883 observations in the dataset.

Table 1: Data Dictionary

Variable	Description	Code Definitions

VendorID	A code indicating the TPEP provider that provided the record.	
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	
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.	
Payment_type	A numeric code signifying how the passenger paid for the trip.	0= Unknown 1= Credit card 2= Cash 3= No charge 4= Dispute
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.	

Summary of Analysis

Data Acquisition, Cleaning and Validation

- We acquired data from the Json URL and imported the data into MongoDb hosted in AWS.
- Performed data cleaning and preparation on the taxi trip data using Pandas. The 'passenger_count' and 'payment_type' columns were converted to appropriate data types (int64) and missing values were handled. Selected columns were also converted to numeric types, but outlier detection was not implemented.
- We utilized Z-score to identify and handle outliers in the 'trip_distance' and 'total_amount' columns by removing data points with Z-scores greater than 3 or less than -3.
- We converted the 'trip_duration' from the initial format to minutes for easier analysis.

Research Questions and Analysis

What is the preferred customer payment method?

Using panda and matplotlib, we were able to analyse the data to determine customer preferred payment types.

LEGEND

- 0: Unknown
- 1: Credit Card
- 2: Cash
- 3: No charge
- 4: Dispute

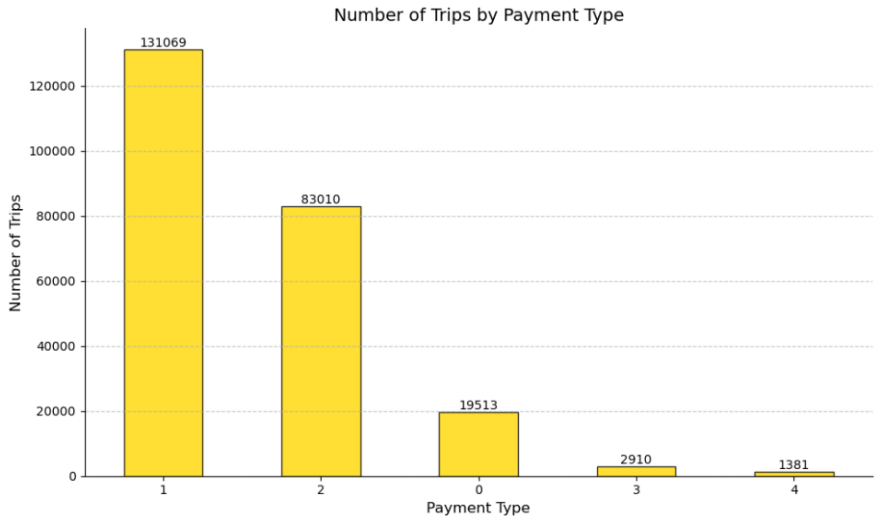


Figure 1

In figure 1 above, we plotted a bar chart of payment types against total number of trips taken for payment type. We can observe that Credit Card Payments have the highest number of trips of 131,069 followed by Cash Payments at 83,010 trips and 19,513 trips with unknown payment methods. It can be noted that credit cards are the favoured methods of payment over cash as credit cards were used for around 1.6 times more trips than cash payments. Credit cards being more compact and easier to carry could be a possible explanation behind this consumer trend.

LEGEND

- 0: Unknown
- 1: Credit Card
- 2: Cash
- 3: No charge
- 4: Dispute

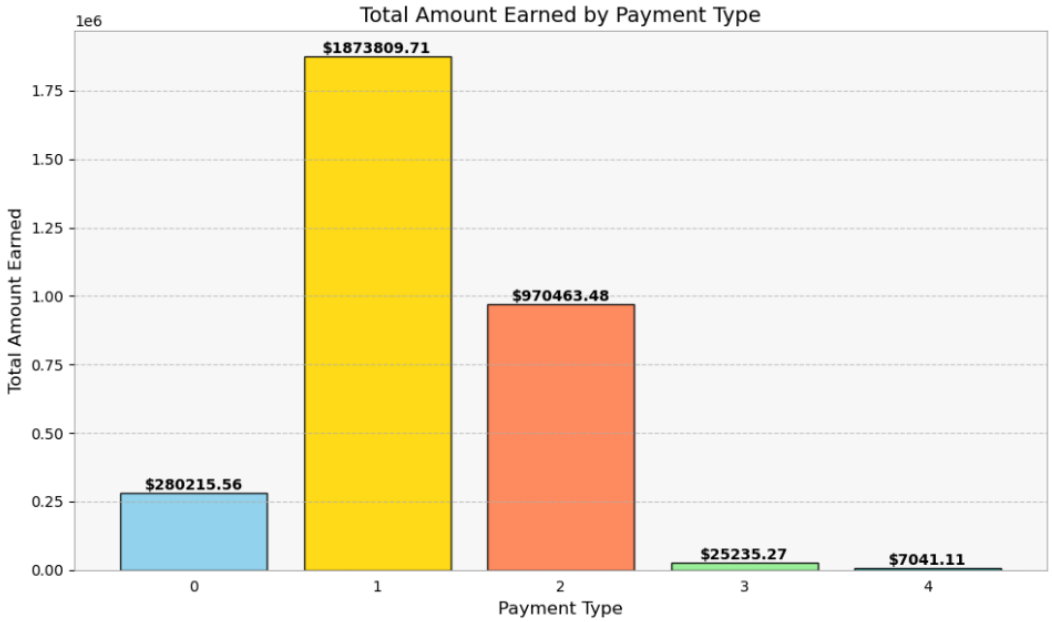


Figure 2

Figure 2 above, represents a bar chart of payment types against total amount earned by each payment type. We can observe that the Credit Card Payments has the highest total amount of $1,873,809.71 which is almost double the amount of Cash Payments of $970,463.48. It can be noted that although the amount earned in credit card payments is double that of the cash payments, Credit cards were only used for 1.6 times more trips than the cash trips. It can be implied based on the above findings that consumers favoured Credit cards over cash for trips that were either longer or generally more expensive.

LEGEND

- 0: Unknown
- 1: Credit Card
- 2: Cash
- 3: No charge
- 4: Dispute

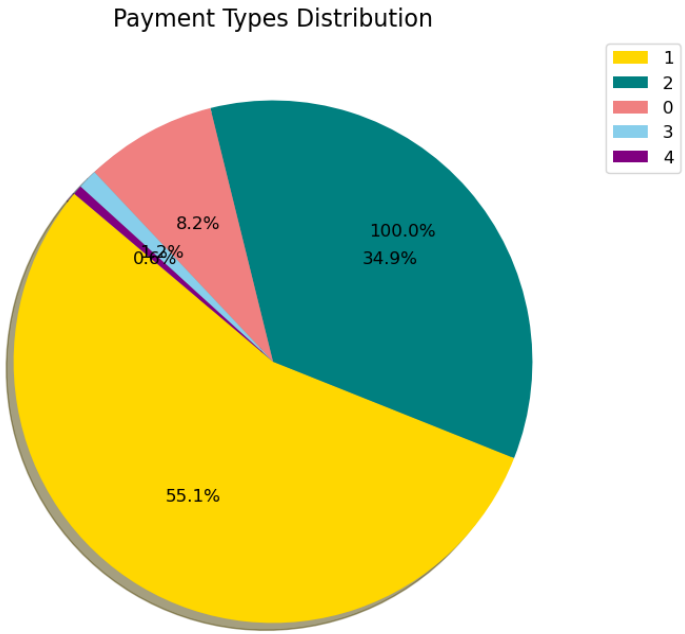


Figure 3

Figure 3 above, represents a pie chart showing the distribution of payment types across all taxi trips taken in the time frame being analysed. It backs up our findings from figure 1 and 2 which shows Credit Card Payment type as most widely used means of payment. Around 2% of the total trips resulted in no payment due to disputes & free rides and 8% of the trips had an unknown payment. Excluding these 2 factors from our analysis, we can see Credit cards were used by as much as 20% more consumers from the pool over cash.

We can therefore conclude that the most widely used and preferred payment method is the Credit card payment type.

When is the peak period for taxi trips?

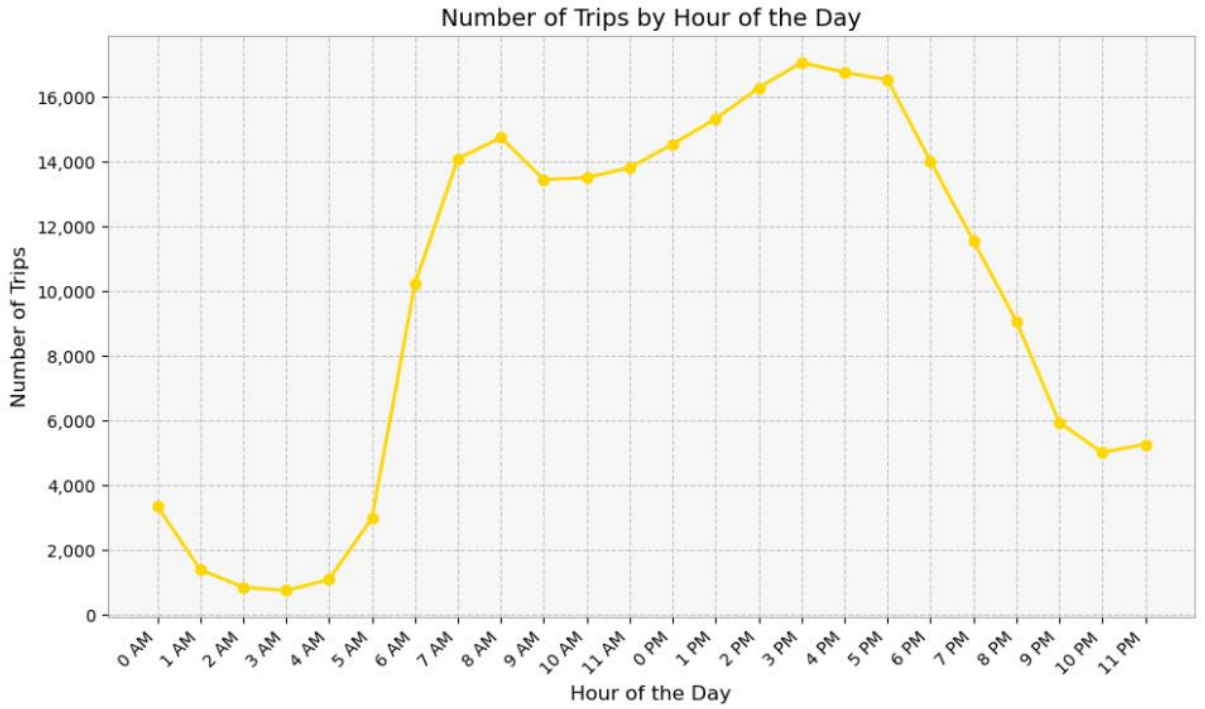


Figure 4

Figure 4 above represents a line chart of the cumulative number of trips taken at each hour of the day. From our observations, we can see that there is a steep rise in the number of trips from 5am to 8am which can be assumed to be peak office resumption periods. The number of trips keep on a gradual rise after then and peak at about 3pm, after which they drop till 9pm with the same intensity as they rose in the morning. With this we can conclude that Taxi's were used the most between 8am and 3pm based on the volume of trips taken in the whole month.

Where are the pickup hot zones?

LEGEND

- 137: Kips Bay
- 140: Lenox Hill East
- 75: East Harlem South
- 238: Upper West Side North
- 263: Yorkville West
- 236: Upper East Side North
- 141: Lenox Hill West
- 48: Lenox Hill West
- 239: Upper West Side South
- 142: Lincoln Square East

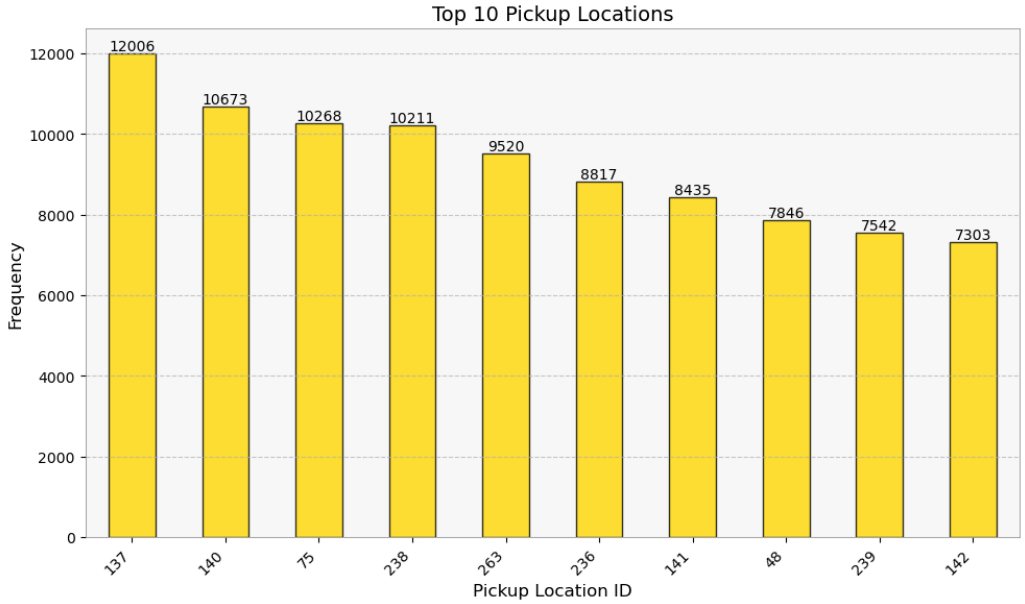


Figure 5

Figure 5 above shows the top 10 pickup locations with the highest number of trips taken. Around 40% of the total trips started from these 10 locations out of the total 263 locations recorded in the data set. We can note that 92,621 of the total 237,883 trips were initiated from 4% of the total locations.

Key Takeaways from Analysis

- Credit cards were the most used payment type with a share of 55% from the total trips.
- Trips associated with Credit card payments tend to be slightly longer or expensive.
- Peak times for taxi rides were between a 7-hour window from 8am to 3pm.
- 4% of the total locations contributed to 40% of the total trips taken.

Slide Deck

DATA PROGRAMMING

FINAL PROJECT- Group 3

2020 Yellow Taxi Trip Data

Intro

Welcome to our data analysis project focused on taxi trips within the vibrant city of New York!

In this project, we delve into a fascinating dataset containing records of taxi rides taken across different neighbourhoods, capturing the bustling life and transportation dynamics of one of the world's most iconic cities.

Our primary objective is to gain valuable insights into the patterns, trends, and behaviours observed in taxi trips across New York City. By employing Python's powerful data analysis tools and libraries, we aim to extract meaningful information from the dataset.

Dataset Description

- The dataset we utilized for this analysis is sourced from the New York City Open Data platform. It comprises a comprehensive collection of taxi trips taken within the city collated by the yellow taxi service providers and includes essential trip details such as pickup, drop-off locations, trip duration, fare amount, and more
- **Focus:** Trips in April 2020, Consisting of 237,883 records and 17 columns
- **Source:** https://data.cityofnewyork.us/Transportation/2020-Yellow-Taxi-Trip-Data/kxp8-n2sj

Meet the Team

Name: Aleksandar Milinovic

Role: Data Analyst

Student ID: 200578728

Name: Avhimanhu Sapkota

Role: Flask Developer

Student ID: 200547612

Name: Anmol Maan

Role: Cloud Support

Student ID: 200379017

Name: Mayank Khosla

Role: Data Prep and Visuals

Student ID: 200556791

Name: Chioma Roberts

Role: Project Manager

Student ID: 200537925

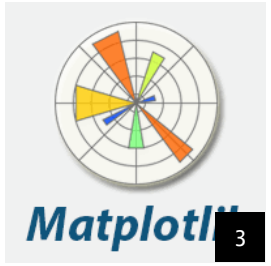
Tools

Tools used in the analysis process

1. **MongoDb**: NoSQL database that falls under the category of document-oriented databases used to store our dataset. It was a first option because of its simplicity and flexibility
2. **Pandas:** popular open-source data manipulation and analysis library for Python. It was used to understand the data and perform data cleaning and preprocessing
3. **Matplotlib**: widely used open-source Python library for creating static, interactive, and animated visualizations of data. It was used to create visualizations of our dataset
4. **Jupyter**: Interactive computing environment chose for source code executions
5. **AWS**: Cloud computing platform used to host our MongoDB

1

2


3

4

5

Research Questions

What is the preferred customer payment method?

When is the peak period for taxi trips?

Where are the pickup hot zones?

Key Analysis Areas

- **Data Exploration:** Uncover the structure and characteristics of the dataset, inspect for missing values using pandas
- **Data Cleaning and Prepossessing:** Performed data cleaning and preparation on the taxi trip data using Pandas
  - Converted relevant columns in the dataset to adequate data types for better processing
  - Excluded outliers using z-score computation on *trip_distance* and *total_amount* column
  - Converted the *trip_duration* from the initial format to minutes for easier analysis
- **Data Visualization:**
  - Examine the amount earned by payment type to determine customer payment type preferences
  - Examine the number of trips by payment types
  - Examine the number of trips that occur at different hours of the day to determine peak and off-peak periods
  - Examine hot zones for pickup locations

CODE REVIEW

```python
def getData(year, month):
    url = "https://data.cityofnewyork.us/resource/kxp8-n2sj.json"

    chunkSize = 1000
    offset = 0
    allData = []
    progressStep = 50000

    while True:
        params = {
            '$limit': chunkSize,
            '$offset': offset,
            '$where': f"tpep_pickup_datetime >= '{year}-{month:02d}-01T00:00:00.000' AND tpep_pickup_datetime < '{year}-{mont
        }

        response = requests.get(url, params=params)
        if response.status_code == 200:
            data = response.json()

            if not data:
                break

            allData.extend(data)
            offset += chunkSize


            if offset % progressStep == 0:
                print(f"Retrieved {offset} rows of data.")

        else:
            print("Failed to get data from the NYC TLC API.")
            return None

    return allData


year = 2020
month = 4
taxitripData = getData(year, month)
```

```python
if taxitripData:
    print("Taxi trip data acquired successfully!\n")
    print("Total records found:", len(taxitripData))
    print("\n")
    print("Sample data:", taxitripData[0])
```

```
Retrieved 50000 rows of data.
Retrieved 100000 rows of data.
Retrieved 150000 rows of data.
Retrieved 200000 rows of data.
Taxi trip data acquired successfully!

Total records found: 237883
```

```python
from pymongo import MongoClient

MongoDBConnectionLink = 'mongodb://BDAT:BDAT@ec2-3-144-229-9.us-east-2.compute.amazonaws.co

def insertDataDBS(data):
    try:
        client = MongoClient(MongoDBConnectionLink)
        db = client['ProjectDatabase']
        collection = db['TaxiData']
        result = collection.insert_many(data)

        if result.acknowledged:
            print("Data inserted into MongoDB successfully!")
        else:
            print("Failed to insert data into MongoDB.")
    except Exception as e:
        print("Error while connecting to MongoDB:", str(e))

if taxitripData:
    insertDataDBS(taxitripData)
```

```
Data inserted into MongoDB successfully!
```

Data Acquisition

- Data was acquired from an API with json data
- We acquired data for the month of April 2020
- Loaded into MongoDb
- MongoDb was hosted on AWS cloud environment

- Using panda, we aimed to get an understanding of our data

Data Statistics

```python
from pymongo import MongoClient
import pandas as pd

MongoDBConnectionLink = 'mongodb://BDAT:BDAT@ec2-3-144-229-9.us-east-2.compute.amazonaws.com

try:
    client = MongoClient(MongoDBConnectionLink)
    db = client['ProjectDatabase']
    collection = db['TaxiData']

    data_from_mongo = list(collection.find())
    client.close()

    taxi_df = pd.DataFrame(data_from_mongo)

    print("Data Overview:")
    print(taxi_df.head())

    print("\nData Types and Missing Values:")
    print(taxi_df.info())

    print("\nSummary Statistics:")
    print(taxi_df.describe())

    print("\nUnique Values in Categorical Column: Payment Type")
    print(taxi_df['payment_type'].unique())

except Exception as e:
    print("Error while connecting to MongoDB:", str(e))
```

```
Data Overview:
                        _id     tpep_pickup_datetime    tpep_dropoff_datetime  \
0  64d23dbb56f66987424f8858  2020-04-01T00:00:00.000  2020-04-01T00:27:00.000
1  64d23dbb56f66987424f8859  2020-04-01T00:00:15.000  2020-04-01T00:05:04.000
2  64d23dbb56f66987424f885a  2020-04-01T00:00:26.000  2020-04-01T00:09:25.000
3  64d23dbb56f66987424f885b  2020-04-01T00:00:28.000  2020-04-01T00:07:59.000
4  64d23dbb56f66987424f885c  2020-04-01T00:00:36.000  2020-04-01T00:03:32.000
```

- Using pandas, we performed some data cleaning on the data for better analysis
- Removed outliers using a z score computation
- Converted relevant columns

Data Cleaning and Preparation

```python
# Data Cleaning and Preparation
try:
    taxi_df['tpep_pickup_datetime'] = pd.to_datetime(taxi_df['tpep_pickup_datetime'])
    taxi_df['tpep_dropoff_datetime'] = pd.to_datetime(taxi_df['tpep_dropoff_datetime'])

    columns_to_convert = ['passenger_count', 'payment_type']
    taxi_df[columns_to_convert] = taxi_df[columns_to_convert].fillna(0).astype('int64')
    convert_cols = [
        'trip_distance', 'pulocationid', 'dolocationid', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
        'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'vendorid',
        'passenger_count', 'ratecodeid', 'store_and_fwd_flag', 'payment_type'
    ]
    taxi_df[convert_cols] = taxi_df[convert_cols].apply(pd.to_numeric, errors='coerce')


    def remove_outliers(column):
        z_scores = (column - column.mean()) / column.std()
        return column[abs(z_scores) <= 3]

    taxi_df['trip_distance'] = remove_outliers(taxi_df['trip_distance'])
    taxi_df['total_amount'] = remove_outliers(taxi_df['total_amount'])
    taxi_df['trip_duration_minutes'] = (taxi_df['tpep_dropoff_datetime'] - taxi_df['tpep_pickup_datetime']).dt.total_seconds

    print("Data Overview:")
    print(taxi_df.head())

    print("\nData Types and Missing Values:")
    print(taxi_df.info())

    print("\nSummary Statistics:")
    print(taxi_df.describe())

    print("\nUnique Values in Categorical Columns:")
    print(taxi_df['payment_type'].unique())
    if 'ratecodeid' in taxi_df.columns:
        print(taxi_df['ratecodeid'].unique())
    else:
        print("Column 'ratecodeid' does not exist in the dataset.")

except Exception as e:
    print("An error occured:", str(e))
```

- Total amount earned for each payment type

Data Visualization

```
import matplotlib.pyplot as plt

payment_type_totals = taxi_df.groupby('payment_type')['total_amount'].sum()

colors = ['skyblue', 'gold', 'coral', 'lightgreen', 'lightseagreen']
plt.figure(figsize=(10, 6))

ax = payment_type_totals.plot(kind='bar', color=colors, edgecolor='black', alpha=0.9, width=0.8)

plt.xlabel('Payment Type', fontsize=12)
plt.ylabel('Total Amount Earned', fontsize=12)
plt.title('Total Amount Earned by Payment Type', fontsize=14)
plt.xticks(rotation=0, fontsize=10)
plt.yticks(fontsize=10)

for index, value in enumerate(payment_type_totals):
    plt.text(index, value, f"${round(value, 2)}", ha='center', va='bottom', fontsize=10, color='black', fontweight='bold')

plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.gca().set_facecolor('#f7f7f7')

for spine in plt.gca().spines.values():
    spine.set_visible(True)
    spine.set_linewidth(0.5)
    spine.set_color('grey')

plt.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=True)
plt.tick_params(axis='y', which='both', left=True, right=False, labelleft=True)

plt.tight_layout()

# Displays the generated plot
plt.show()
```

- Number of trips per payment type

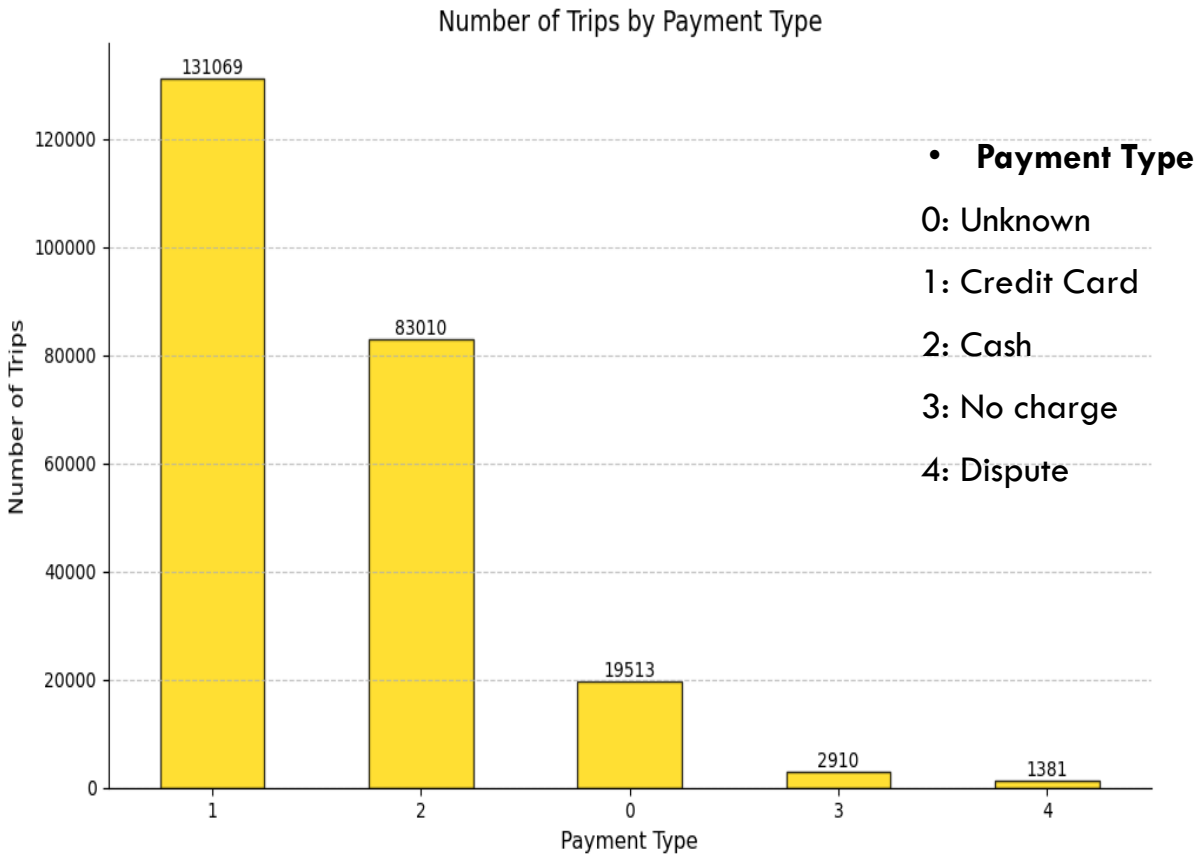
Data Visualization

```
import matplotlib.pyplot as plt

# Creates a bar chart depicting the distribution of payment types for taxi trips
plt.figure(figsize=(10, 6))

payment_counts = taxi_df['payment_type'].value_counts()
payment_counts.plot(kind='bar', color='gold', edgecolor='black', alpha=0.8)

plt.xlabel('Payment Type', fontsize=12)
plt.ylabel('Number of Trips', fontsize=12)
plt.title('Number of Trips by Payment Type', fontsize=14)
plt.xticks(rotation=0, fontsize=10)
plt.yticks(fontsize=10)

for index, value in enumerate(payment_counts):
    plt.text(index, value, str(value), ha='center', va='bottom', fontsize=10)

plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()

# Shows the generated plot
plt.show()
```

Data Visualization

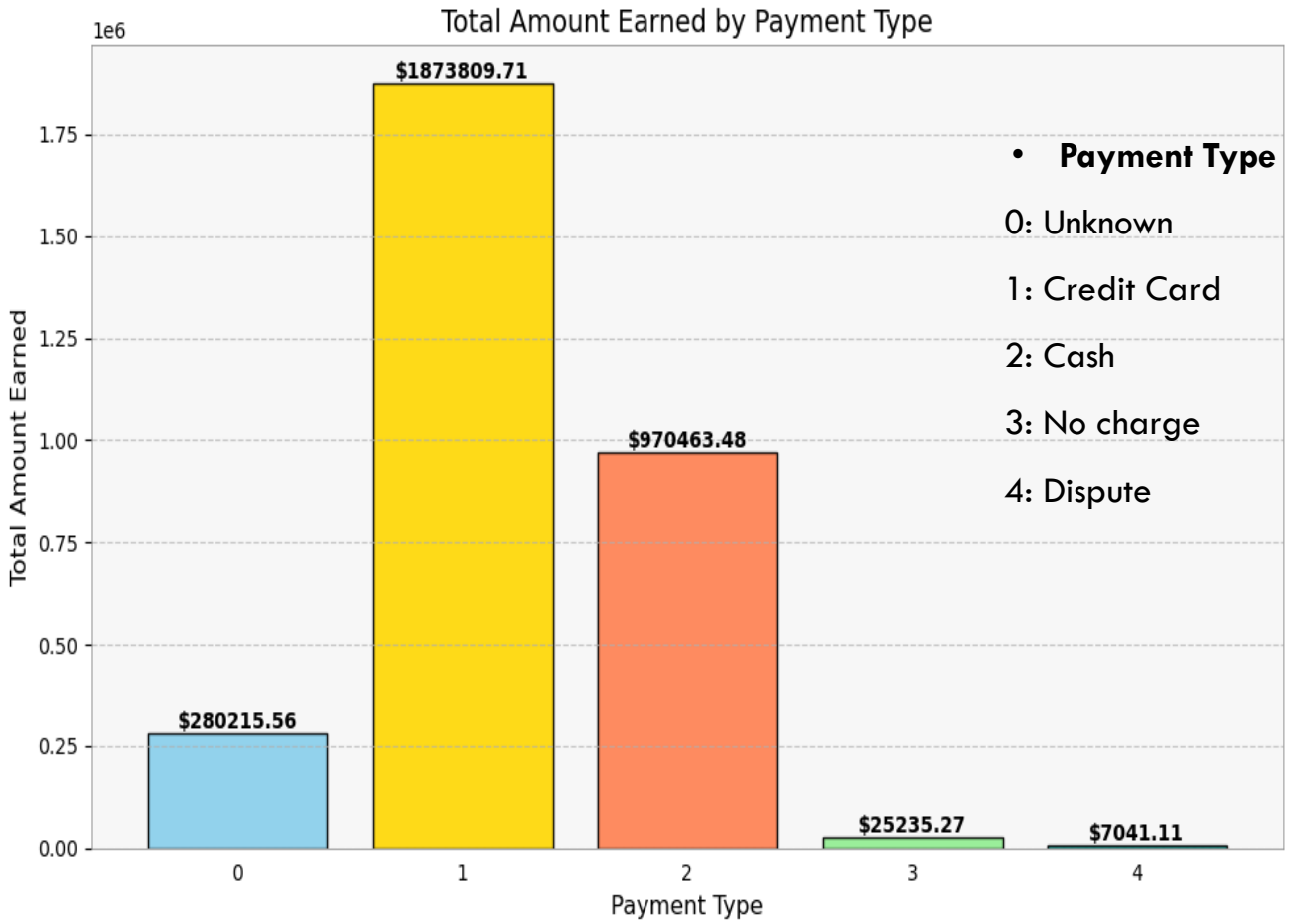
- Number of trips by hour of the day

```python
import matplotlib.pyplot as plt

taxi_df['pickup_hour'] = taxi_df['tpep_pickup_datetime'].dt.hour

hourly_trip_counts = taxi_df['pickup_hour'].value_counts().sort_index()

plt.figure(figsize=(10, 6))

hourly_trip_counts.plot(kind='line', marker='o', color='gold', linestyle='-', linewidth=2)

plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Number of Trips', fontsize=12)
plt.title('Number of Trips by Hour of the Day', fontsize=14)

hour_labels = [f'{h % 12} AM' if h < 12 else f'{h % 12} PM' for h in range(24)]
plt.xticks(range(24), hour_labels, fontsize=10, rotation=45, ha='right')  # Adjust rotation and alignment here

plt.gca().get_yaxis().set_major_formatter(plt.FuncFormatter(lambda x, loc: "{:,}".format(int(x))))
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)

plt.grid(True, linestyle='--', alpha=0.7)

plt.gca().set_facecolor('#f7f7f7')

for spine in plt.gca().spines.values():
    spine.set_visible(True)
    spine.set_linewidth(0.5)
    spine.set_color('grey')

plt.tight_layout()

# Displays the generated plot
plt.show()
```



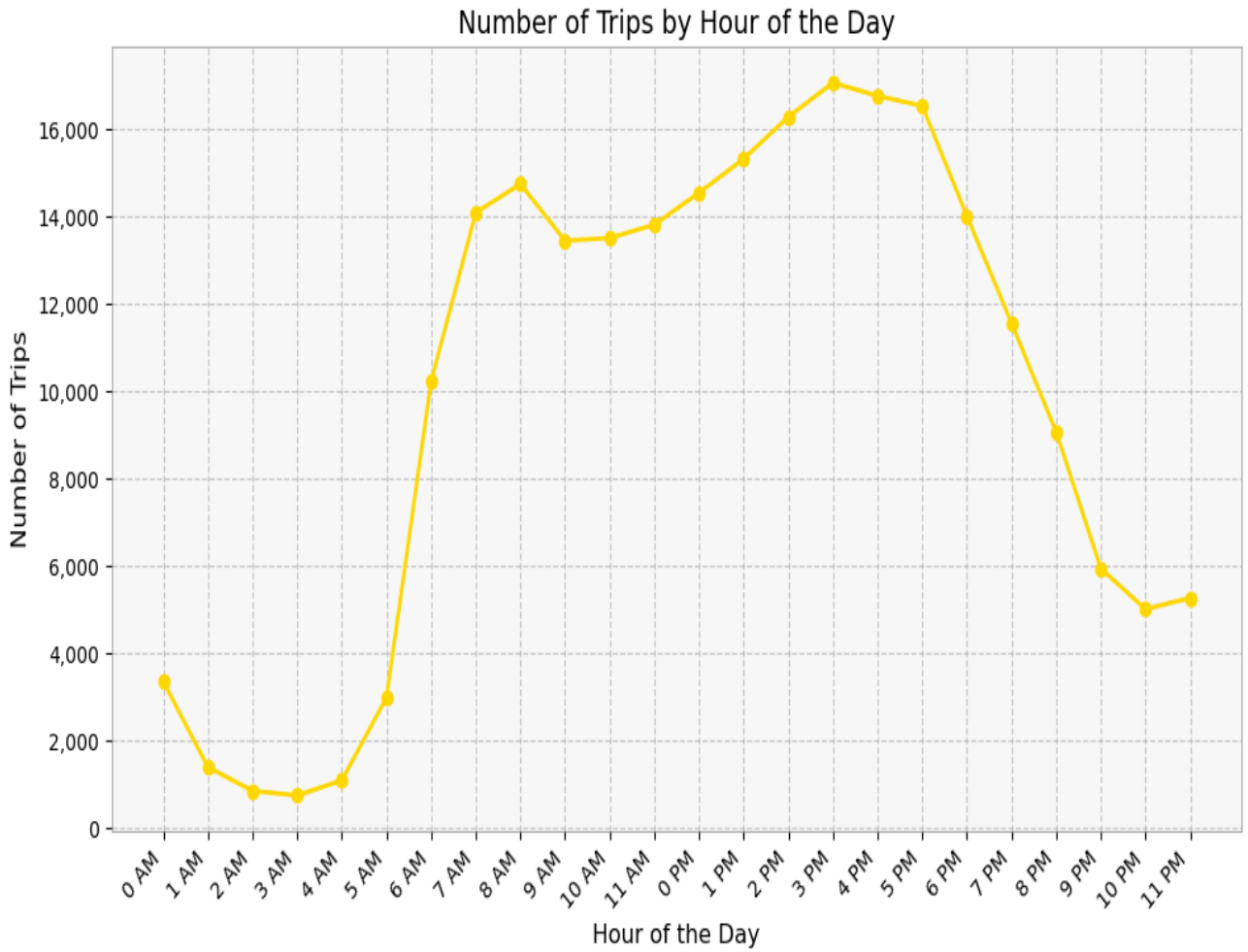
What is the preferred customer payment method?

- Credit Cards were the choice of payment for 131,000 consumers making them the most used payment types with a share of 55% of the total trips
- Cash was used in 83,010 trips making 35% of the total transactions
- 19,513 trips had unknown payment types
- Around 3,000 of the total trips were discounted or not charged
- And 1381 trips resulted in a dispute



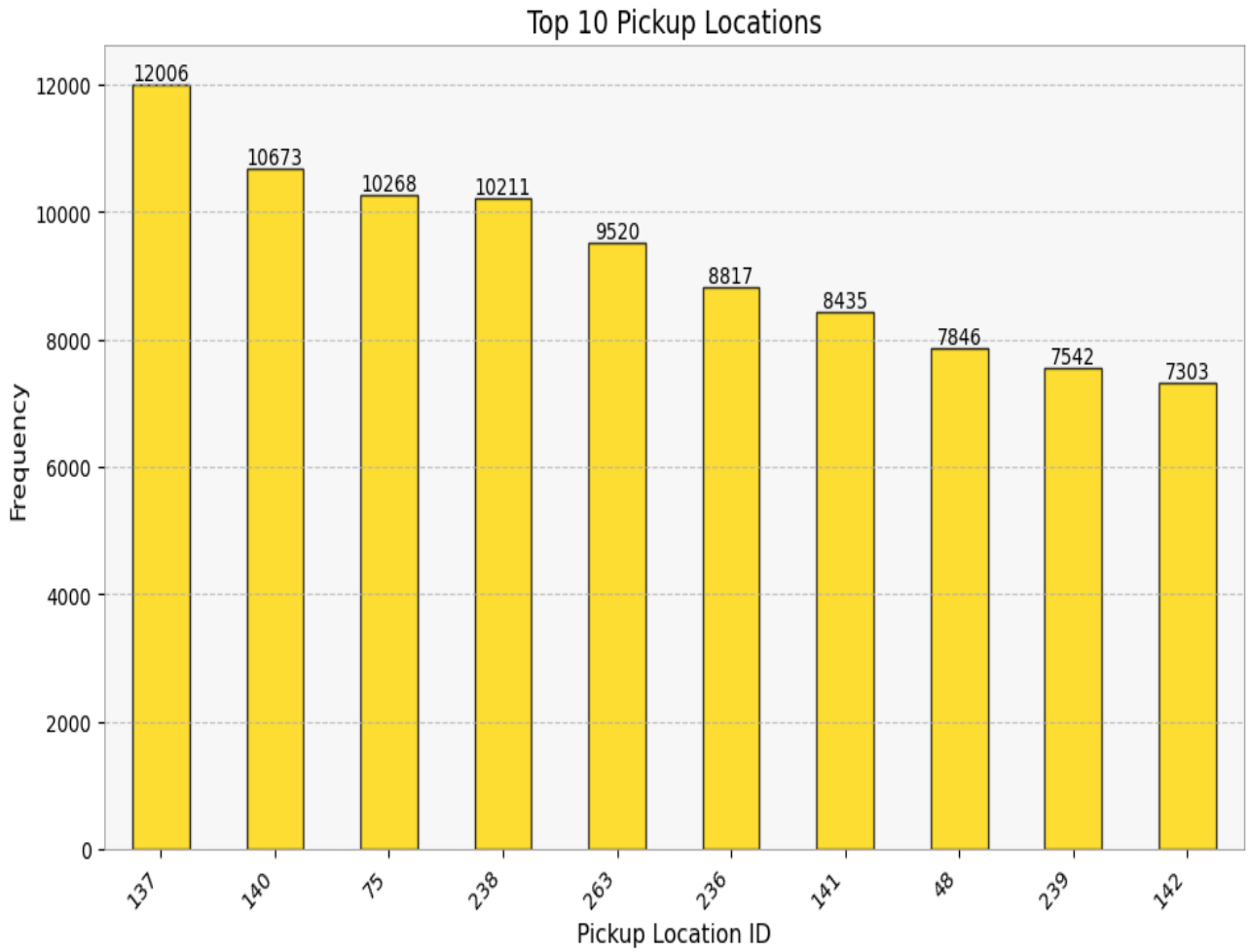
What is the preferred customer payment method?

- Credit Cards were the most common payment types and contributed to $1.9 million in earnings
- Cash was used for $1 million in transactions
- Around $300k in earnings was received through unknown payment types.
- Around 2% of the total trips were affected by discounts and disputes which comprised of $30k in earnings



When is the peak period for taxi trips?

- Peak hours remained between 8am and 3pm
- Number of trips start picking up at 4am followed by a steep incline in trips from 5am to 8am before it reaches the peak period
- The number follows a similar trajectory downwards after the peak period and drops to around 30% of daily trip count



What are the pickup hot zones?

- **LOCATION**

137: Kips Bay

140: Lenox Hill East

75: East Harlem South

238: Upper West Side North

263: Yorkville West

236: Upper East Side North

141: Lenox Hill West

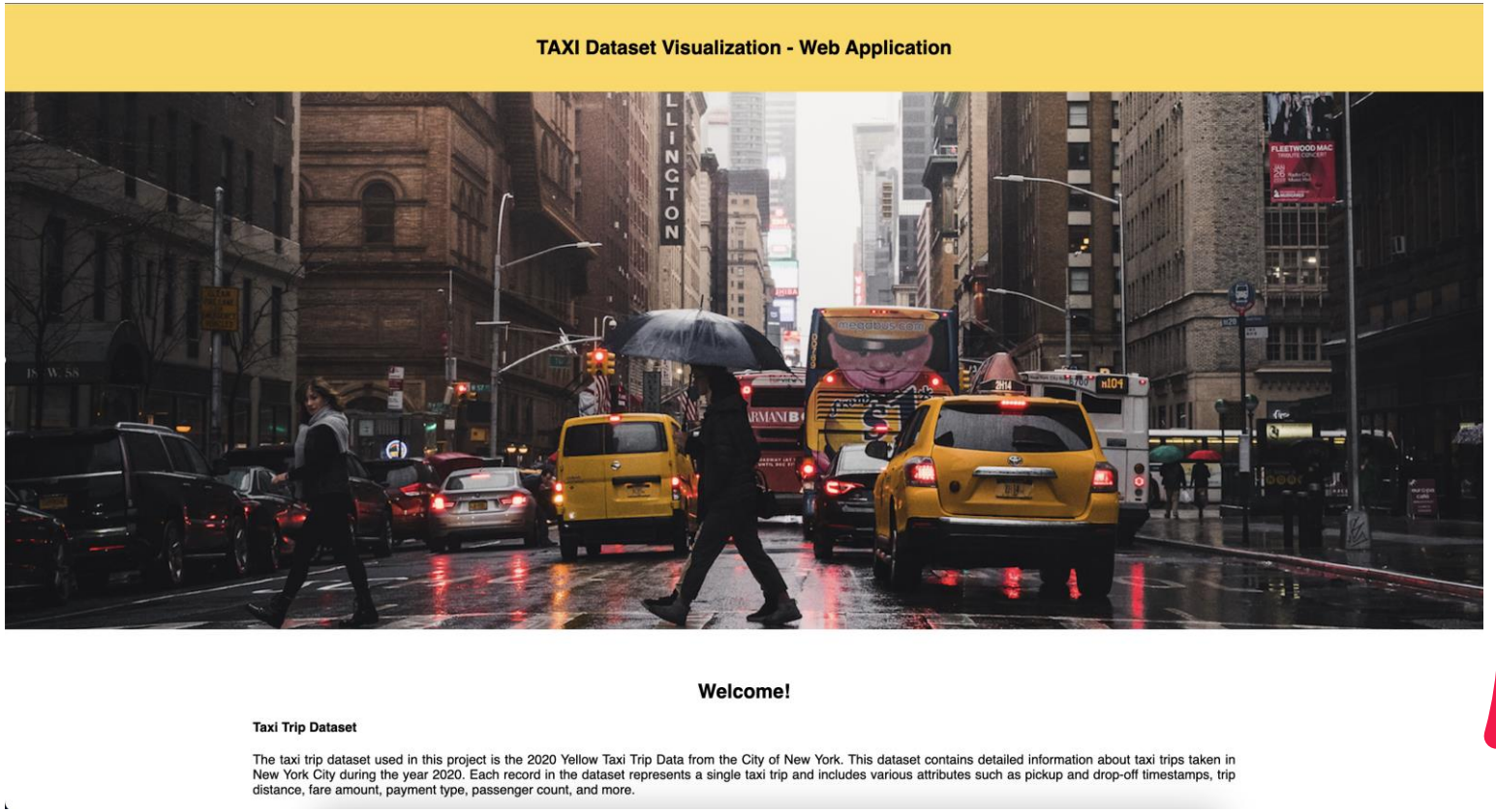
48: Lenox Hill West

239: Upper West Side South

142: Lincoln Square East

- On the left side are the Top 10 pickup points out of the 263 total locations studied in this data
- This makes them 4% of the total locations and 40% of the total trips started at these 10 locations

Bonus: Flask Web App for Visualizations



Thank You