NYC Taxi Data Analysis

The New York City Taxi and Limousine Commission (TLC), created in 1971, is the agency responsible for licensing and regulating New York City's Medallion (Yellow) taxi cabs, for-hire vehicles (community-based liveries, black cars and luxury limousines), commuter vans, and paratransit vehicles. **Over 200,000 TLC licensees complete approximately 1,000,000 trips each day.** In New York City, the yellow taxis are an important iconic symbol that represents the city. However, NYC taxi market has deteriorated for the traditional yellow cabs since the Uber and other ridesharing applications introduced in the city. It is therefore important to understand their pickup patterns.

1.DATA CONNECTION

AWS S3 Storage

We are using AWS S3 storage to store our data files for this project

```
from pyspark import SparkConf
from pyspark.sql import SparkSession
aws_access_key_id='AKIAZUJ2HPPBQK7QL4ZF',
aws_secret_access_key='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'
builder = SparkSession.builder.appName("csp554_PROJECT")
builder = builder.config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")\
.config("spark.hadoop.com.amazonaws.services.s3.enableV4", "true")\
.config("spark.jars.packages", "org.apache.hadoop:hadoop-aws:3.2.2,com.amazonaws:aws-java-sdk-bundle:1.11.888")\
.config("spark.hadoop.fs.s3a.aws.credentials.provider", "org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider")\
.config("spark.hadoop.fs.s3a.access.key", aws_access_key_id)\
.config("spark.hadoop.fs.s3a.secret.key", aws_secret_access_key)
spark = builder.getOrCreate()
spark
```

SparkSession - hive

SparkContext

[Spark UI]

Version
    `v3.3.0`
Master
    `local[*, 4]`

Import Libraries

```
from pyspark.sql import *
from pyspark.sql.functions import *
from pyspark.sql.types import *

import pandas
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import FuncFormatter
```

Schema for trip data

```
ddl_schema = (StructType([
   StructField("VendorID", IntegerType(), False),
   StructField("tpep_pickup_datetime", TimestampType(), False),
   StructField("tpep_dropoff_datetime", TimestampType(), False),
   StructField("passenger_count", IntegerType(), False),
   StructField("trip_distance", FloatType(), False),
   StructField("RatecodeID", IntegerType(), False),
   StructField("store_and_fwd_flag", StringType(), False),
   StructField("PULocationID", IntegerType(), False),
   StructField("DOLocationID", IntegerType(), False),
   StructField("payment_type", IntegerType(), False),
   StructField("fare_amount", FloatType(), False),
   StructField("extra", FloatType(), False),
   StructField("mta_tax", FloatType(), False),
   StructField("tip_amount", FloatType(), False),
   StructField("tolls_amount", FloatType(), False),
   StructField("improvement_surcharge", FloatType(), False),
   StructField("total_amount", FloatType(), False),
   StructField("congestion_surcharge", FloatType(), False)]))
```

2.DATA GATHERING (INPUT DATA)

Read taxi trip input files

```
inputPath = "s3://choladevicssp554/yellow_tripdata_2022-*.parquet"
NYInputDF = (
     spark
    .read
    .parquet(inputPath, header = True)
)

#NYInputDF.show(5)
display(NYInputDF.limit(5))
```

Table

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	R
1	1	2022-10-01T00:03:41.000+0000	2022-10-01T00:18:39.000+0000	1	1.7	1
2	2	2022-10-01T00:14:30.000+0000	2022-10-01T00:19:48.000+0000	2	0.72	1
3	2	2022-10-01T00:27:13.000+0000	2022-10-01T00:37:41.000+0000	1	1.74	1
4	1	2022-10-01T00:32:53.000+0000	2022-10-01T00:38:55.000+0000	0	1.3	1
5	1	2022-10-01T00:44:55.000+0000	2022-10-01T00:50:21.000+0000	0	1	1

5 rows

Check taxi trip data columns

```
NYInputDF.columns
```

```
Out[20]: ['VendorID',
 'tpep_pickup_datetime',
 'tpep_dropoff_datetime',
 'passenger_count',
```

```
 'trip_distance',
 'RatecodeID',
 'store_and_fwd_flag',
 'PULocationID',
 'DOLocationID',
 'payment_type',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'congestion_surcharge',
 'airport_fee']
```

'VendorID' is a code indicating the TPEP provider that provided the record.

```
   1= Creative Mobile Technologies, LLC;


   2= VeriFone Inc.
```

'tpep_pickup_datetime' is the date and time when the meter was engaged.

'tpep_dropoff_datetime' is the date and time when the meter was disengaged.

'Passenger_count' is the number of passengers in the vehicle. This is a driver-entered value.

'Trip_distance' is he elapsed trip distance in miles reported by the taximeter.

'PULocationID' is the TLC Taxi Zone in which the taximeter was engaged

'DOLocationID' is the TLC Taxi Zone in which the taximeter was disengaged

'RateCodeID' is 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' is the flag that 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' is 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' is 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 that is automatically triggered based on the metered rate in use.

'Improvement_surcharge' $0.30 improvement surcharge assessed trips at the flag drop.

'Tip_amount' is automatically populated for credit card tips. Cash tips are not included.

Total record count for Taxi trip data

```
NYInputDF.count()
```

```
Out[21]: 39656098
```

We have a total of ~39.6M trip records from the year 2022

Schema for Taxi Zone Lookup data

```
taxi_schema = (StructType([
   StructField("LocationID", IntegerType(), False),
   StructField("Borough", StringType(), False),
   StructField("Zone", StringType(), False),
   StructField("service_zone", StringType(), False)]))
```

Read taxi zone lookup file

```
taxi_zones = spark.read.csv('s3://choladevicssp554/taxi_zone_lookup.csv', header=True, schema = taxi_schema)
display(taxi_zones.limit(5))
```

Table

	LocationID	Borough	Zone	service_zone
1	1	EWR	Newark Airport	EWR
2	2	Queens	Jamaica Bay	Boro Zone
3	3	Bronx	Allerton/Pelham Gardens	Boro Zone
4	4	Manhattan	Alphabet City	Yellow Zone

5	5	Staten Island	Arden Heights	Boro Zone

5 rows

```
taxi_zones.columns
```

```
Out[24]: ['LocationID', 'Borough', 'Zone', 'service_zone']
```

'LocationID' is the ID respresenting the location

'Borough' is the borough associated with the trip

'Zone' is the zone name ssociated with the trip

'service_zone' is the overall service zone of the trip

Check record count for taxi zone lookup table

```
taxi_zones.count()
```

```
Out[25]: 265
```

Taxi zone lookup contains 265 entries

3.DATA QUALITY CHECK

Filter out unknown boroughs

```
taxi_zones = taxi_zones.filter(taxi_zones.Borough != 'Unknown')
taxi_zones.count()
```

```
Out[26]: 263
```

This could be junk data for this analyis. After filtering out unknown boroughs we have 263 zones.

```
taxi_zones.cache()
```

```
Out[27]: DataFrame[LocationID: int, Borough: string, Zone: string, service_zone: string]
```

Potentially erroneous total_amount

```
display(NYInputDF.select("total_amount").describe())
```

Table

	summary	total_amount
1	count	39656098
2	mean	21.671268443177723

3	stddev	96.37360220544986
4	min	-2567.8
5	max	401095.62

5 rows

```
print("Count of rows with total amount below zero: ",NYInputDF.filter(NYInputDF.total_amount < 0).count())
display(NYInputDF.filter(NYInputDF.total_amount < 0).limit(5))
```

```
Count of rows with total amount below zero:  255706
```

Table

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	R
1	2	2022-10-01T00:52:07.000+0000	2022-10-01T01:02:44.000+0000	1	2.83	1
2	2	2022-10-01T00:29:57.000+0000	2022-10-01T00:34:37.000+0000	1	0.31	1
3	2	2022-10-01T00:31:37.000+0000	2022-10-01T00:32:53.000+0000	1	0.31	1
4	2	2022-10-01T00:46:38.000+0000	2022-10-01T00:46:44.000+0000	1	0	2
5	2	2022-10-01T00:06:08.000+0000	2022-10-01T00:48:23.000+0000	1	17.05	1

5 rows

There are some instances where the total_amount is less than 0. This does not make a lot of sense given the data set, and seems to likely be caused by erroneous data entry.

Unreasonable Trip Distance

```
display(NYInputDF.select("trip_distance").describe())
```

Table

	summary	trip_distance
1	count	39656098
2	mean	5.959398968098615
3	stddev	599.1907143779275
4	min	0.0
5	max	389678.46

5 rows

```
print("Trip records count with less than 0 distanc: ",NYInputDF.filter(NYInputDF.trip_distance <= 0).count())
display(NYInputDF.filter(NYInputDF.trip_distance <= 0).limit(5))
```

```
Trip records count with less than 0 distanc:  574059
```

Table

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	R
1	2	2022-10-01T00:46:38.000+0000	2022-10-01T00:46:44.000+0000	1	0	2
2	2	2022-10-01T00:46:38.000+0000	2022-10-01T00:46:44.000+0000	1	0	2
3	1	2022-10-01T00:32:03.000+0000	2022-10-01T00:49:48.000+0000	1	0	1
4	1	2022-10-01T00:41:34.000+0000	2022-10-01T00:44:51.000+0000	1	0	1
5	2	2022-10-01T00:59:21.000+0000	2022-10-01T00:59:25.000+0000	1	0	5

5 rows

There are some instances where the trip_distance is less than or equal to 0. Since thie distance cannot be 0 miles, this does not make a lot of sense given the data set, and seems to likely be caused by erroneous data entry.

Erroneous passenger_count

```
display(NYInputDF.select("passenger_count").describe())
```

Table

	summary	passenger_count
1	count	38287795
2	mean	1.4011492173942115
3	stddev	0.9628938026962127
4	min	0.0
5	max	9.0

5 rows

There are records where the passenger_count values equal to 0 or greater. Since this value is entered by taxi drivers, the 0s may be instances where the driver did not enter any value.

Incomplete dataset/erroneous dates

```
min_date, max_date = NYInputDF.select(min("tpep_pickup_datetime"), max("tpep_pickup_datetime")).first()
min_date, max_date
```

```
Out[33]: (datetime.datetime(2001, 1, 1, 0, 3, 14),
 datetime.datetime(2023, 4, 18, 14, 30, 5))
```

The data seems to contain erroneous data. There seems to be quite a bit of trips from the past (e.g. 2001), and trips from the future (e.g. 2023).

Trips with unknown Vendors

```
NYInputDF.where(col('VendorID').isNull()).count()
```

```
Out[34]: 0
```

```
NYInputDF.registerTempTable("raw_taxi_data")
```

```
/databricks/spark/python/pyspark/sql/dataframe.py:234: FutureWarning: Deprecated in 2.0, use createOrReplaceTempView inst
ead.
  warnings.warn("Deprecated in 2.0, use createOrReplaceTempView instead.", FutureWarning)
```

4.DATA CLEANING

Filtering Out Problematic Data

By doing the data quality check we came across some errorneous trip records. Hence we will now filter out the problematic data

```
query = """
SELECT
    *
FROM
    raw_taxi_data st
WHERE
    st.total_amount < 10000
    AND st.total_amount > 0
    AND st.trip_distance < 1000
    AND st.trip_distance > 0
    AND st.fare_amount > 0
    AND st.passenger_count < 100
    AND st.passenger_count >= 0
    AND st.tpep_pickup_datetime >= '2022-01-01'
    AND st.tpep_pickup_datetime <= '2022-12-31'
    AND st.VendorID is not Null
"""
# create a new DataFrame based on the query
cleaned_data = spark.sql(query)
print("Count of records after cleaning: ", cleaned_data.count())
```

```
Count of records after cleaning:  37478729
```

Deriving hour, day of week, month, trip duration & trip speed

We have the trip date time stamp by which we can derive some relevent information. We are also using the trip distance and trip diration to derive trip speed

```
cleaned_data = cleaned_data.withColumn('month', month('tpep_pickup_datetime')) \
              .withColumn('dayofweek', dayofweek('tpep_pickup_datetime')) \
              .withColumn('hour', hour('tpep_pickup_datetime')) \
    .withColumn('trip_duration_min', round((col('tpep_dropoff_datetime').cast("long") -
col('tpep_pickup_datetime').cast("long"))/60, 2)) \
    .withColumn('trip_speed_mph', round((col('trip_distance')/ col('trip_duration_min'))*60, 2)) \
    .withColumn("week_day", date_format(col("tpep_pickup_datetime"), "EEEE"))
display(cleaned_data.limit(5))
```

Table

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	R
1	1	2022-10-01T00:03:41.000+0000	2022-10-01T00:18:39.000+0000	1	1.7	1
2	2	2022-10-01T00:14:30.000+0000	2022-10-01T00:19:48.000+0000	2	0.72	1
3	2	2022-10-01T00:27:13.000+0000	2022-10-01T00:37:41.000+0000	1	1.74	1
4	1	2022-10-01T00:32:53.000+0000	2022-10-01T00:38:55.000+0000	0	1.3	1
5	1	2022-10-01T00:44:55.000+0000	2022-10-01T00:50:21.000+0000	0	1	1

5 rows

5.DATA ANALYSIS

```
print("Records with NUll Pickup location: ",cleaned_data.where(col('PULocationID').isNull()).count())
print("Records with NUll Drop Location: ",cleaned_data.where(col('DOLocationID').isNull()).count())
```

```
Records with NUll Pickup location:  0
Records with NUll Drop Location:  0
```

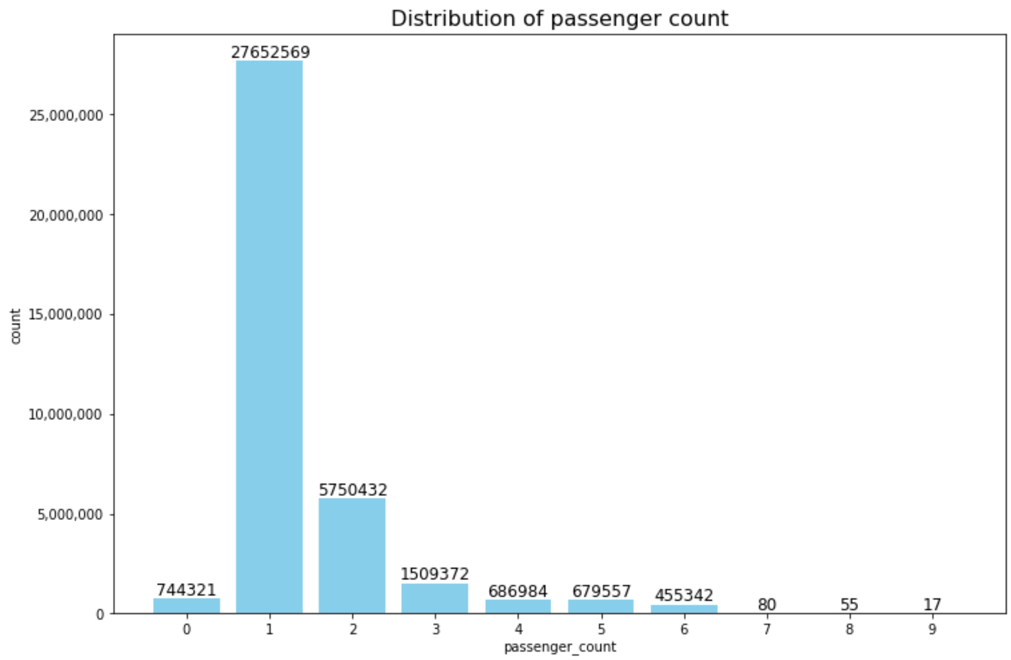
```
#util functions for visualization
def format_y_ticks(value, _):
    return '{:,.0f}'.format(value)

def distributions(df, col1, col2):
    plt.figure(figsize=(12, 8))
    bars = plt.bar(df[col1], df[col2],color='skyblue')
    for bar, value in zip(bars, df[col2]):
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), f'{value}', ha='center', va='bottom',
fontsize=12, color='black')
    plt.xticks(df[col1])
    plt.gca().yaxis.set_major_formatter(FuncFormatter(format_y_ticks))
    return plt
```

```
cleaned_data.cache()
```

```
Out[40]: DataFrame[VendorID: bigint, tpep_pickup_datetime: timestamp, tpep_dropoff_datetime: timestamp, passenger_count:
double, trip_distance: double, RatecodeID: double, store_and_fwd_flag: string, PULocationID: bigint, DOLocationID: bigin
t, payment_type: bigint, fare_amount: double, extra: double, mta_tax: double, tip_amount: double, tolls_amount: double, i
mprovement_surcharge: double, total_amount: double, congestion_surcharge: double, airport_fee: double, month: int, dayofw
eek: int, hour: int, trip_duration_min: double, trip_speed_mph: double, week_day: string]
```

Distribution of Passenger count

```
temp_pd_df= cleaned_data.groupby("passenger_count").count().orderBy("passenger_count").toPandas()
plt = distributions(temp_pd_df, 'passenger_count','count')
plt.xlabel("passenger_count")
plt.ylabel("count")
plt.title("Distribution of passenger count",fontsize=16)
plt.show()
```



The passenger count has values ranging from 0 to 9. The passenger count values are entered manually by the driver. So the 0s are probably the values where the driver did not enter any information. If we look at the chart, about 85% of the trips have only 1 passenger.

Distribution of Trip distance

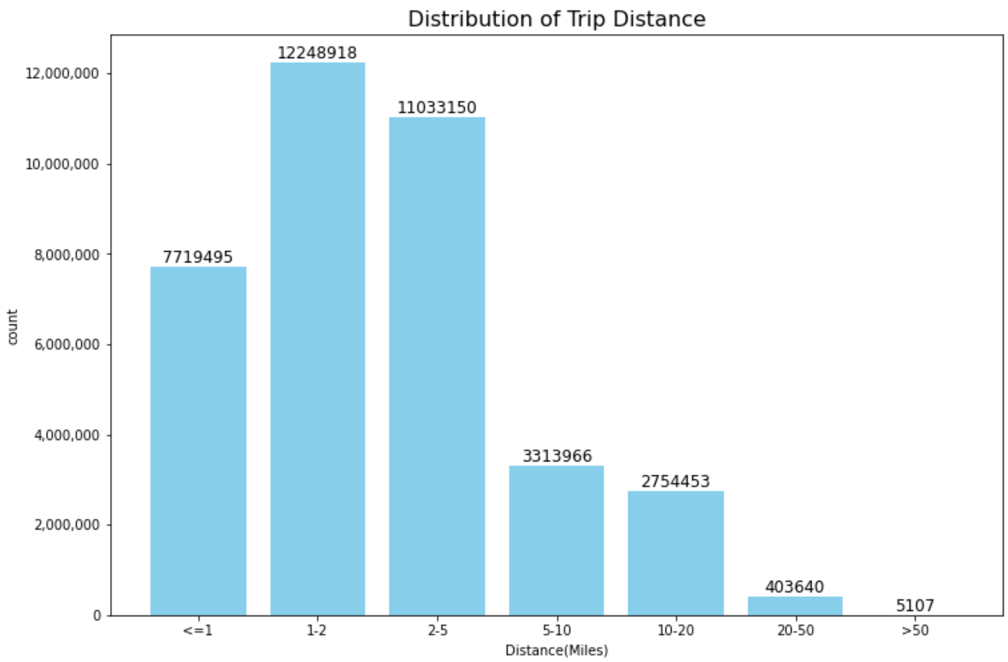
```
display(cleaned_data.select("trip_distance").describe())
```

Table

	summary	trip_distance
1	count	37478729
2	mean	3.5217762835554316
3	stddev	4.517171721999825
4	min	0.01
5	max	905.02

5 rows

```
trip_dt = cleaned_data.withColumn("distance_buckets", when((col('trip_distance') <=1.0), "<=1")
                                        .when((col('trip_distance') > 1.0) & (col('trip_distance') <=2.0), "1-2")
                                        .when((col('trip_distance') > 2.0) & (col('trip_distance') <=5.0), "2-5")
                                        .when((col('trip_distance') > 5.0) & (col('trip_distance') <=10.0), "5-10")
                                        .when((col('trip_distance') > 10.0) & (col('trip_distance') <=20.0), "10-20")
                                        .when((col('trip_distance') > 20.0) & (col('trip_distance') <=50.0), "20-50")
                                        .otherwise(">50"))
trip_dt= trip_dt.groupby("distance_buckets").count().orderBy("distance_buckets").toPandas().reindex([5,0,2,4,1,3,6])

plt = distributions(trip_dt, 'distance_buckets','count')
plt.xlabel("Distance(Miles)")
plt.ylabel("count")
plt.title("Distribution of Trip Distance", fontsize=16)
plt.show()
```



The value for trip distance have been made into several buckets based on the data. The highest being 50+ bucket.About 62% of the overall trips are shorterr distances. Most common trip distance usually lies in the range between 1-5 miles.

Distribution of Total amount

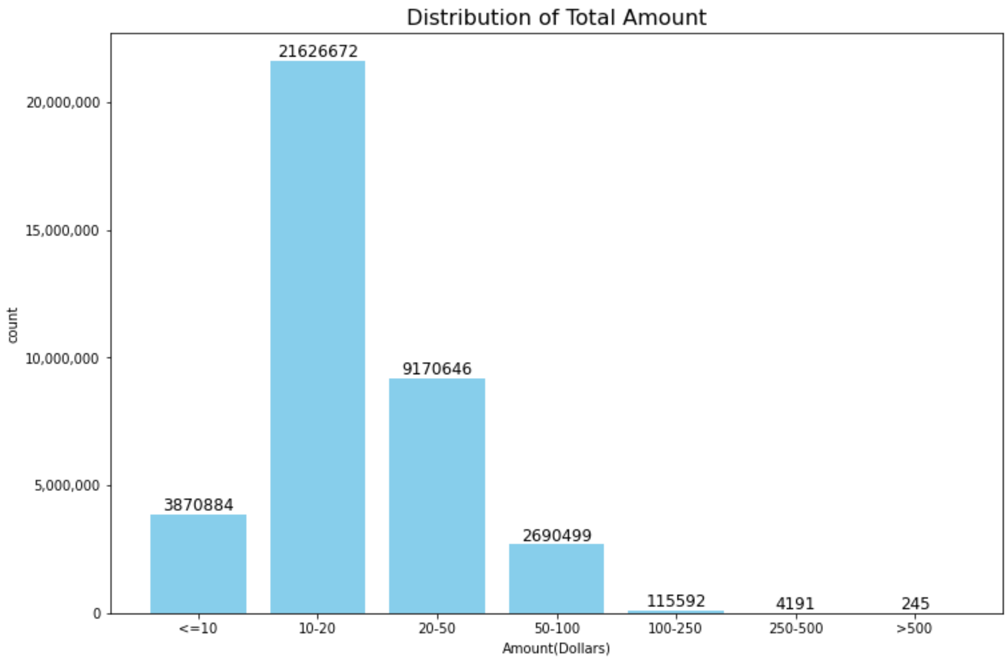
```
display(cleaned_data.select("total_amount").describe())
```

Table

	summary	total_amount
1	count	37478729
2	mean	21.471989157963144
3	stddev	19.020115196622505
4	min	0.31
5	max	7060.85

5 rows

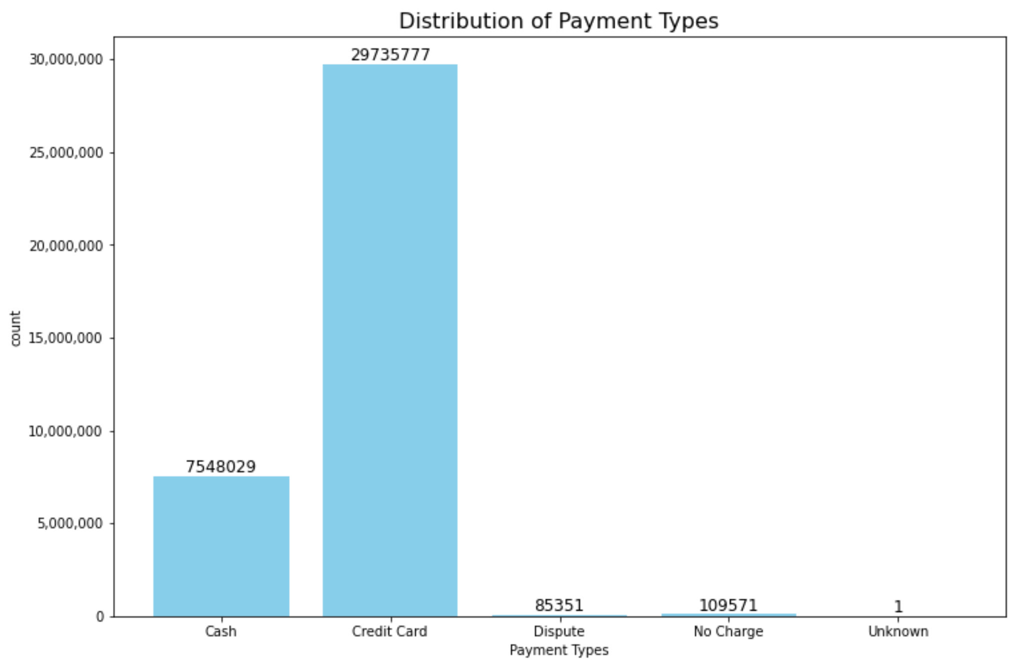
```
tot_amnt = cleaned_data.withColumn("tot_amnt_buckets", when((col('total_amount') <=10.0), "<=10")
                                    .when((col('total_amount') > 10.0) & (col('total_amount') <=20.0), "10-20")
                                    .when((col('total_amount') > 20.0) & (col('total_amount') <=50.0), "20-50")
                                    .when((col('total_amount') > 50.0) & (col('total_amount') <=100.0), "50-100")
                                    .when((col('total_amount') > 100.0) & (col('total_amount') <=250.0), "100-
250")
                                    .when((col('total_amount') > 250.0) & (col('total_amount') <=500.0), "250-
500")
                                    .otherwise(">500"))
tot_amnt=
tot_amnt.groupby("tot_amnt_buckets").count().orderBy("tot_amnt_buckets").toPandas().reindex([5,0,2,4,1,3,6])

plt = distributions(tot_amnt, 'tot_amnt_buckets','count')
plt.xlabel("Amount(Dollars)")
plt.ylabel("count")
plt.title("Distribution of Total Amount", fontsize=16)
plt.show()
```



The fare amount goes from the range 0.1 to almost 7k dollars, so the amount has been divided into several buckets for the visualization purposes. Since the most common trips were shorter trips, the total amount also coorelates with the trip distance. Majority of the amounts were within 20$

Distribution of different payment types

```
payment_df = cleaned_data.withColumn("payment_types", when((col('payment_type') ==1), "Credit Card")
                                        .when((col('payment_type') ==2), "Cash")
                                        .when((col('payment_type') ==3), "No Charge")
                                        .when((col('payment_type') ==4), "Dispute")
                                        .when((col('payment_type') ==5), "Unknown")
                                        .when((col('payment_type') ==6), "Voided Trip")
                                        .otherwise("Unknown"))
payment_df= payment_df.groupby("payment_types").count().orderBy("payment_types").toPandas()
#.reindex([5,0,2,4,1,3,6])
plt = distributions(payment_df, 'payment_types','count')
plt.xlabel("Payment Types")
plt.ylabel("count")
plt.title("Distribution of Payment Types", fontsize=16)
plt.show()
```



From the distribution of payment type, we can see that most of the people prefer paying through credit card. This seems to be an intersiting insights because since these are call up taxis unlike uber/lyft, the assumption of cash being highest mode of transport is ruled out. Probably, the drivers started carrying some sort of devices for credit card payments.

Visualizing trip patterns

Trips on a monthly basis

```
display(cleaned_data
 .groupby("month")
 .count().orderBy("month"))
```

Table   Bar chart

	month	count
1	1	2352843
2	2	2833026
3	3	3453413
4	4	[illegible]

4	4	3424742
5	5	3396064
6	6	3357168

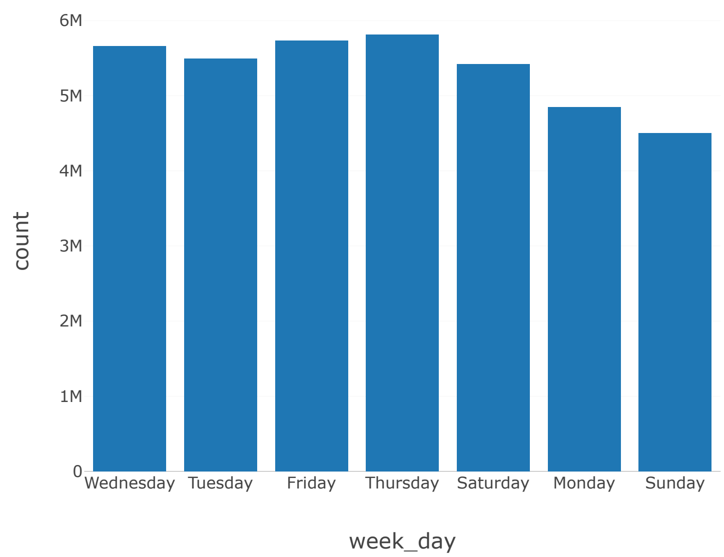
12 rows

There is no major differences between the months. Most of them fall into similar range expect January/February. This could be due to the peak winter season in NYC and it makes harder for dirvers to reach the locations in snowy areas.

Trips per day of the week

```
display(cleaned_data
 .groupby("week_day")
 .count())
```

Visualization



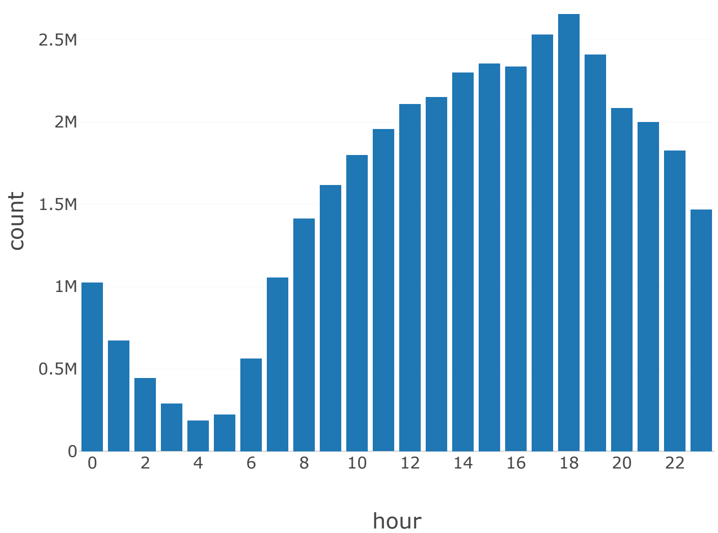
7 rows

This is an interesting insight. People tend to use the taxis more on weekdays rather weekends. This could be to avoid traffic delays, office days.

Trips on an hourly basis

```
display(cleaned_data
 .groupby("hour")
 .count()
 .orderBy("hour"))
```

Visualization



24 rows

Hourly analysis of trips shows that the highest number of trips occur between 5 to 7 pm. As expected, this is the time where people head back to their homes from their offices.

Distinct pickup and drop off locations

```
print("Pickup Locations:
",cleaned_data.select("PULocationID").where(col("PULocationID").isNotNull()).distinct().count())
print("Drop Off Locations :",
cleaned_data.select("DOLocationID").where(col("DOLocationID").isNotNull()).distinct().count())
```

```
Pickup Locations:  262
Drop Off Locations : 262
```

We have 262 distinct pickup and dropoff location ids

Combining trip data with taxi lookup data

```
trip_data = cleaned_data.join(taxi_zones, cleaned_data.PULocationID == taxi_zones.LocationID, how='inner')
trip_data = trip_data.withColumnRenamed('Borough', 'pick_up_borough')
trip_data = trip_data.withColumnRenamed('Zone', 'pick_up_zone')
trip_data = trip_data.withColumnRenamed('service_zone', 'pick_up_service_zone')
trip_data = trip_data.drop('LocationID')

trip_data = trip_data.join(taxi_zones, trip_data.DOLocationID == taxi_zones.LocationID, how='inner')
trip_data = trip_data.withColumnRenamed('Borough', 'drop_off_borough')
trip_data = trip_data.withColumnRenamed('Zone', 'drop_off_zone')
trip_data = trip_data.withColumnRenamed('service_zone', 'drop_off_service_zone')
trip_data = trip_data.drop('LocationID')
```

```
display(trip_data.limit(8))
```

Table

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	R
1	1	2022-10-01T00:03:41.000+0000	2022-10-01T00:18:39.000+0000	1	1.7	1
2	2	2022-10-01T00:14:30.000+0000	2022-10-01T00:19:48.000+0000	2	0.72	1
3	2	2022-10-01T00:27:13.000+0000	2022-10-01T00:37:41.000+0000	1	1.74	1
4	1	2022-10-01T00:32:53.000+0000	2022-10-01T00:38:55.000+0000	0	1.3	1
5	1	2022-10-01T00:44:55.000+0000	2022-10-01T00:50:21.000+0000	0	1	1
6	1	2022-10-01T00:22:52.000+0000	2022-10-01T00:52:14.000+0000	1	6.8	1

8 rows

```
trip_data.cache()
```

```
Out[53]: DataFrame[VendorID: bigint, tpep_pickup_datetime: timestamp, tpep_dropoff_datetime: timestamp, passenger_count:
double, trip_distance: double, RatecodeID: double, store_and_fwd_flag: string, PULocationID: bigint, DOLocationID: bigin
t, payment_type: bigint, fare_amount: double, extra: double, mta_tax: double, tip_amount: double, tolls_amount: double, i
mprovement_surcharge: double, total_amount: double, congestion_surcharge: double, airport_fee: double, month: int, dayofw
eek: int, hour: int, trip_duration_min: double, trip_speed_mph: double, week_day: string, pick_up_borough: string, pick_u
p_zone: string, pick_up_service_zone: string, drop_off_borough: string, drop_off_zone: string, drop_off_service_zone: str
ing]
```

Most frequent pick up location

Most Frequent Borough

```
display(trip_data
 .groupBy("pick_up_borough")
 .count()
 .orderBy("count", ascending=False))
```

Table

	pick_up_borough	count
1	Manhattan	33487681
2	Queens	3157371
3	Brooklyn	186106
4	Bronx	34404
5	Staten Island	2087
6	EWR	2022

6 rows

Manhattan being the heart of the NYC, it is the most frequent pickup borough

```
neighborhoodCounts = (trip_data.groupBy("pick_up_borough").count()).toPandas()
neighborhoodCounts.rename(columns={"pick_up_borough":"boro_name"}, inplace=True)
neighborhoodCounts
```

	boro_name	count
0	Queens	3157371
1	EWR	2022
2	Brooklyn	186106
3	Staten Island	2087
4	Manhattan	33487681
5	Bronx	34404

The geoJson file for borough boundaries is attached in the shared uploads and is downloaded from the link: https://data.cityofnewyork.us/City-Government/Borough-Boundaries/tqmj-j8zm (https://data.cityofnewyork.us/City-Government/Borough-Boundaries/tqmj-j8zm)

```
display(trip_data
 .groupBy("drop_off_borough")
 .count()
 .orderBy("count", ascending=False))
```

Table

	drop_off_borough	count
1	Manhattan	33249650
2	Queens	1905346
3	Brooklyn	1397280
4	Bronx	214658
5	EWR	93736
6	Staten Island	9001

6 rows

Similar to pickup, Manhattan is also the most frequent drop off borough

Most Frequent Zone

```
display(trip_data
 .groupBy("pick_up_zone")
 .count()
 .orderBy("count", ascending=False).limit(5))
```

Table

	pick_up_zone	count
1	Upper East Side South	1800180
2	JFK Airport	1772129
3	Upper East Side North	1592154
4	Midtown Center	1518991
5	Penn Station/Madison Sq West	1255604

5 rows

```
display(trip_data
 .groupBy("drop_off_zone")
 .count()
 .orderBy("count", ascending=False).limit(5))
```

Table

	drop_off_zone	count
1	Upper East Side North	1626346
2	Upper East Side South	1551012
3	Midtown Center	1378285
4	Murray Hill	1113590
5	Times Sq/Theatre District	1105135

5 rows

```
display(trip_data
 .groupBy("pick_up_zone")
 .count()
 .orderBy("count", ascending=True).limit(5))
```

Table

	pick_up_zone	count
1	Crotona Park	6
2	Eltingville/Annadale/Prince's Bay	7
3	Rossville/Woodrow	11
4	Breezy Point/Fort Tilden/Riis Beach	11
5	Rikers Island	12

5 rows

```
display(trip_data
 .groupBy("drop_off_zone")
 .count()
 .orderBy("count", ascending=True).limit(5))
```

Table

	drop_off_zone	count
1	Great Kills Park	3
2	Governor's Island/Ellis Island/Liberty Island	18
3	Jamaica Bay	31
4	Freshkills Park	34
5	Rossville/Woodrow	144

5 rows

Looking at all the above tables demonstrating top and bottom 5 pickup and drop off zones, It is clear that Upper East Side South is the most frequent pickup zone whereas Upper East Side North is the most frequent dropoff zone. Alternatively, Crotona park and Great Kills park are lease preferred pickup and drop off zones.

Most Frequent trips

```
display(trip_data
 .groupBy("pick_up_borough","drop_off_borough")
 .count()
 .orderBy("count", ascending=False))
```

Table   frequent_trips

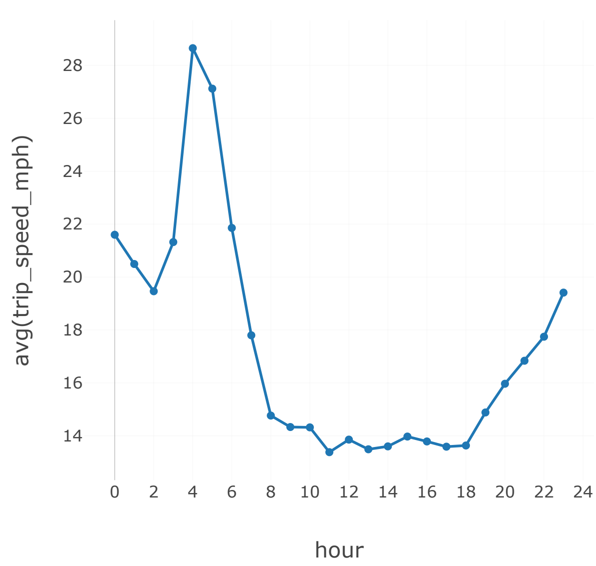
	pick_up_borough	drop_off_borough	count
1	Manhattan	Manhattan	31258819
2	Queens	Manhattan	1905735
3	Manhattan	Queens	1215738
4	Manhattan	Brooklyn	799309
5	Queens	Queens	671821
6	Queens	Brooklyn	500244

36 rows

Traffic flow for each day of the week

```
display(cleaned_data.groupby("hour").mean("trip_speed_mph"))
```

Visualization

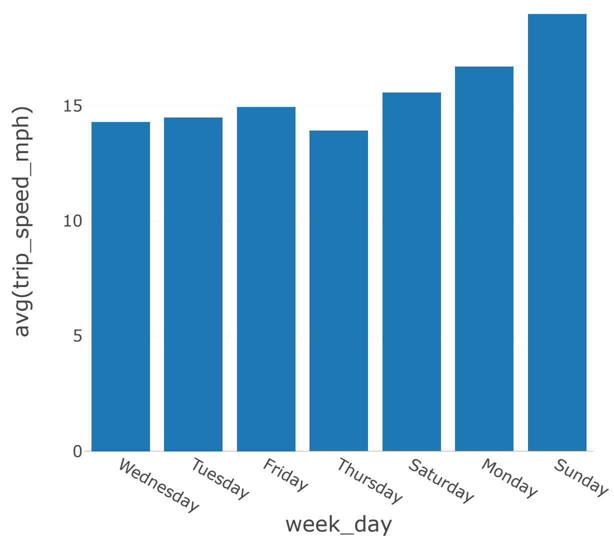


24 rows

Analyzing average trip speed on a hourly basis shows that 8 am to 6 pm are the busiest hours of the day with heavy traffic flow which leads to lower trip speed comparatively

```
display(cleaned_data.groupby("week_day").mean("trip_speed_mph"))
```

Visualization

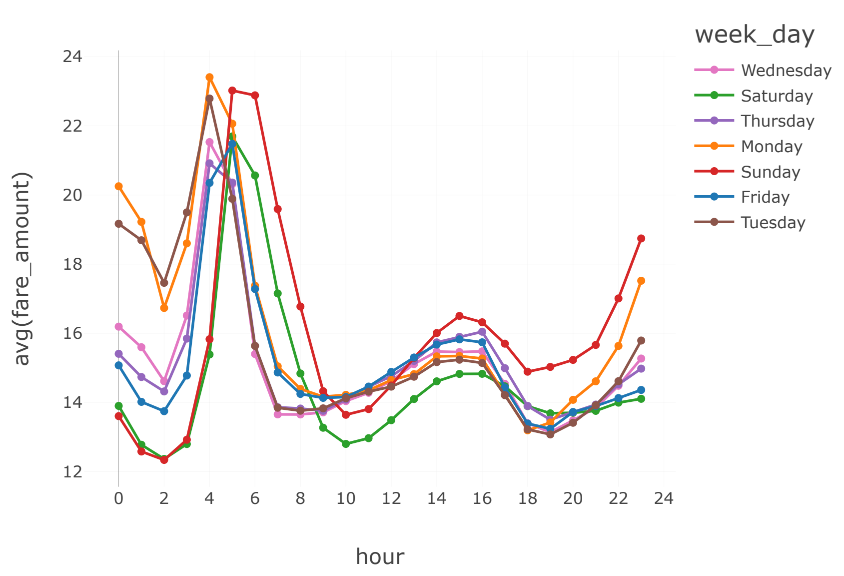


7 rows

The trip speed on Thursdays is the lowest which might be because Thursday is the day with the highest number of trips. On the other hand, Sunday has comparatively higher trip speed which is again due to less trips on Sundays

```
display(cleaned_data.groupby("hour","week_day").mean("fare_amount"))
```

Visualization

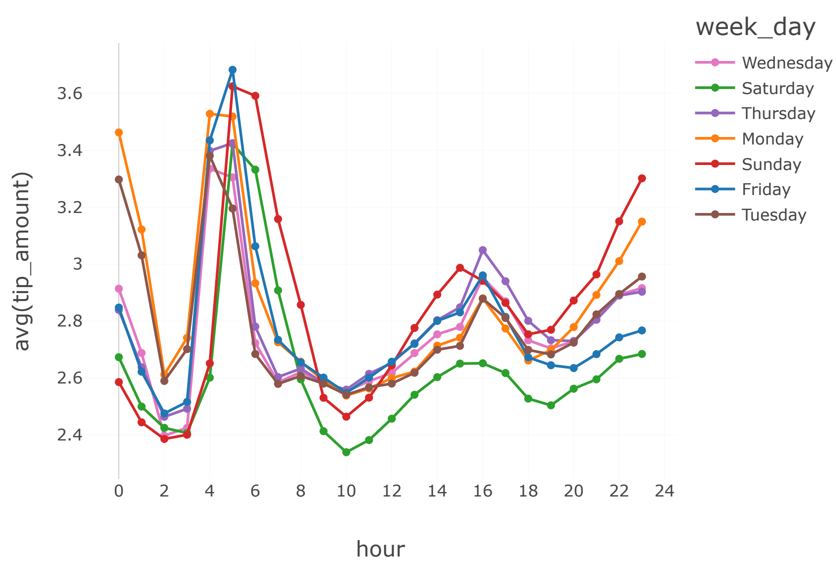


168 rows

The above plot shows average fare amount on an hourly basis for each day of the week. It is observed the trips taken between midnight to early morning have a slightly higher average fare amount. This might be because of it being an odd time to travel.

```
display(cleaned_data.groupby("hour","week_day").mean("tip_amount"))
```

Visualization



168 rows

The plot shows average tip amount on an hourly basis for each day of the week. It is observed that, people tend to give more tip odd hours whereas they give comparatively less during the bussiest hours of the day.

```
count_uess_df = (trip_data
 .where(trip_data.pick_up_zone == "Upper East Side South")
 .groupBy("drop_off_zone")
 .count()
 .orderBy(("count"), ascending=False)).toPandas()
count_uess_df = count_uess_df[:10]
avg_fare_uess_df = (trip_data
 .where(trip_data.pick_up_zone == "Upper East Side South")
 .groupBy("drop_off_zone")
 .avg("fare_amount")
 .orderBy(avg("fare_amount"), ascending=False)).toPandas()
count_uess_df = count_uess_df.merge(avg_fare_uess_df, how="left", on="drop_off_zone")
count_uess_df["src"] = "Upper East Side South"
count_uess_df.rename(columns={"drop_off_zone": "dst"}, inplace=True)
count_uess_df["count"] = count_uess_df["count"].astype("object")
count_uess_df["count"] = count_uess_df["count"].astype("int")
count_uess_df.head()
```

	dst	count	avg(fare_amount)	src
0	Upper East Side North	252835	6.689553	Upper East Side South
1	Upper East Side South	164721	5.797472	Upper East Side South
2	Midtown Center	110981	7.721712	Upper East Side South
3	Midtown East	87003	7.074599	Upper East Side South
4	Lenox Hill West	73550	6.008313	Upper East Side South

Analyzing the Rate Codes

```
display(trip_data.groupby('RatecodeID').avg('trip_distance', 'fare_amount'))
```

Table

	RatecodeID	avg(trip_distance)	avg(fare_amount)
1	1	2.796603291932519	12.42490513671462
2	4	15.340280723643167	61.6061447286338
3	3	17.28990122943647	68.97399369968497
4	2	17.937567553755393	52.600335612865216
5	99	7.820314759520436	32.74317478984135
6	6	2.6711320754716987	4.0150943396226415

7 rows

Rate codes 2 and 3 are to JFK and Newark Airports so it is not too surprising that there is extra charge.

Rate code 4 is trips to Nassau County or Westchester County which have a median trip distance slightly lower than for rate codes 2 and 3. This is because trips to these counties are charged at the standard city rate within New York City and at twice the metered rate while in Westchester or Nassau County.

Rate code 5 is for negotiated fares and the median trip distance for fares with this rate code is much less which suggests that cab driver's are much better off when fares are negotiated.