NYU

NYC Taxi Fare Prediction

CS-GY 6513: Big Data

Team Sye!

05.09.2023

NYC Taxi Fare Prediction

Aim

The aim of this project is to predict the fare for taxis in New York City (NYC) using large-scale data processing technologies.

Description

The project involves collecting the NYC taxi dataset, pre-processing the data, storing it, performing exploratory data analysis (EDA), building a predictive model using, evaluating the model's performance, scaling the computations, and visualizing the findings.

python

Apache Spark

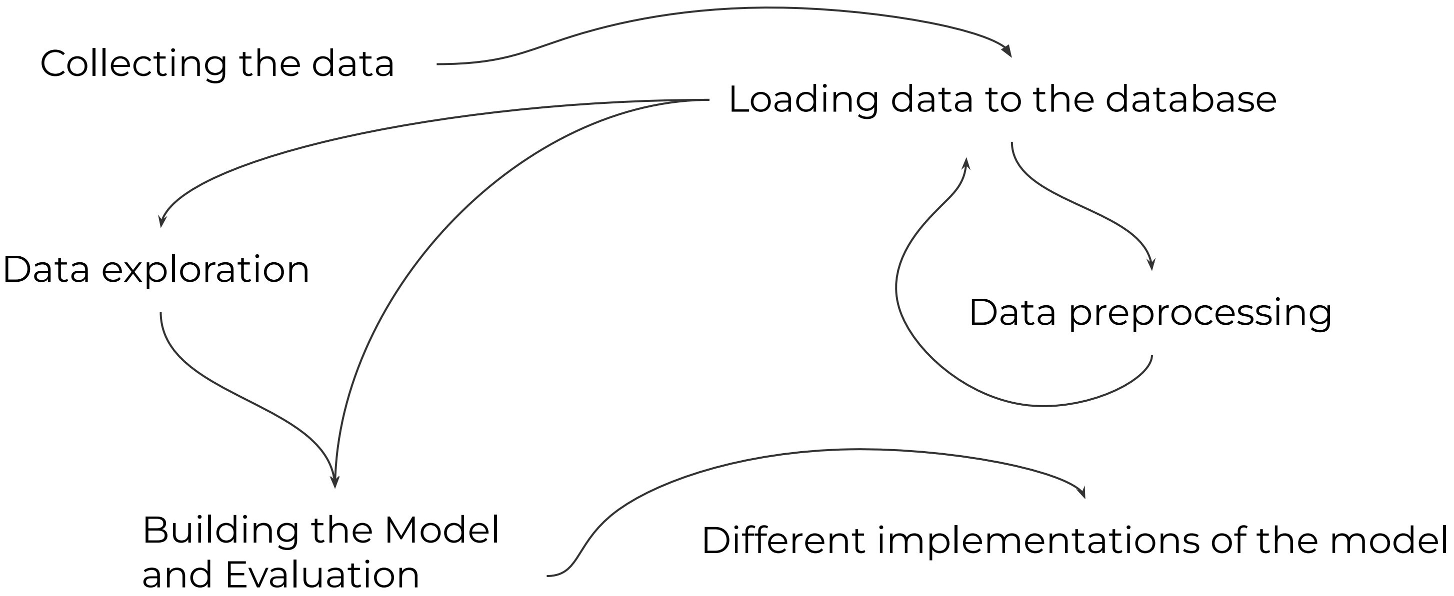
dask

Technologies Used

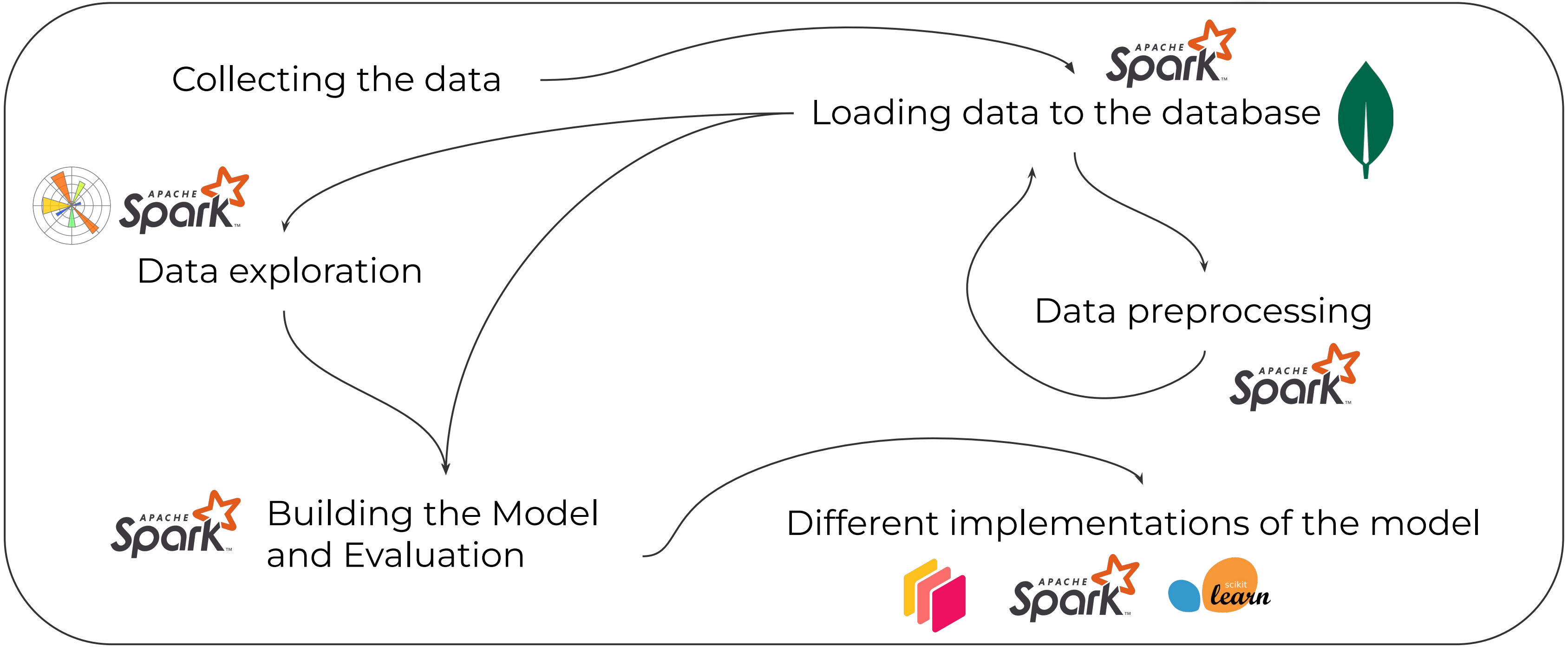
MongoDB

scikit learn

Workflow


Collecting the data
Loading data to the database
Data exploration
Data preprocessing
Building the Model and Evaluation
Different implementations of the model

Workflow w/ Technologies

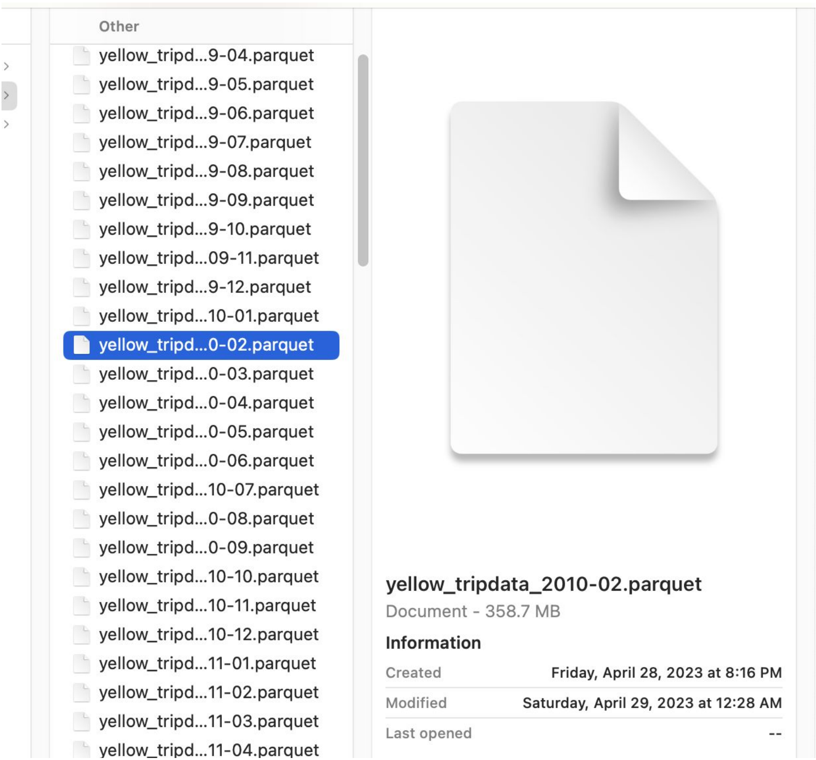


Dataset

Get the latest on the COVID-19 Vaccine
NYC Taxi & Limousine Commission
311 Search all NYC.gov websites
NYC
Taxi & Limousine Commission
Italiano Translate Text-Size
About
Passengers
Drivers
Vehicles
Businesses
TLC Online
Search
About TLC
Data and Reports
TLC Initiatives
Contact TLC
Data
Pilot Programs
Reports
TLC Trip Record Data
Request Data
TLC Trip Record Data
Yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The data used in the attached datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP). The trip data was not created by the TLC, and TLC makes no representations as to the accuracy of these data.
For-Hire Vehicle ("FHV") trip records include fields capturing the dispatching base license number and the pick-up date, time, and taxi zone location ID (shape file below). These records are generated from the FHV Trip Record submissions made by bases. Note: The TLC publishes base trip record data as submitted by the bases, and we cannot guarantee or confirm their accuracy or completeness. Therefore, this may not represent the total amount of trips dispatched by all TLC-licensed bases. The TLC performs routine reviews of the records and takes enforcement actions

Load the data

MongoDB®



Data Preprocessing

1. Removing unnecessary columns
2. Handling missing values
3. Creating new features

In [13]: `df.printSchema()`

```
root
 |-- VendorID: long (nullable = true)
 |-- tpep_pickup_datetime: timestamp (nullable = true)
 |-- tpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: double (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- PULocationID: long (nullable = true)
 |-- DOLocationID: long (nullable = true)
 |-- payment_type: long (nullable = true)
 |-- fare_amount: double (nullable = true)
 |-- extra: double (nullable = true)
 |-- mta_tax: double (nullable = true)
 |-- tip_amount: double (nullable = true)
 |-- tolls_amount: double (nullable = true)
 |-- improvement_surcharge: double (nullable = true)
 |-- total_amount: double (nullable = true)
 |-- congestion_surcharge: double (nullable = true)
 |-- airport_fee: double (nullable = true)
 |-- __index_level_0__: long (nullable = true)
```

NYU

Handling missing values

Check for missing values in numerical columns and non-numerical columns

Drop rows with missing values

Alternatively, you can fill missing values with specific values

- replace missing numeric values with the median
- replace missing categorical values with the mode

Creating new features

In [21]:
```
from pyspark.sql.functions import year, month, dayofmonth, hour, dayofweek

# Create new features
df = df.withColumn('pickup_year', year(df['tpep_pickup_datetime']))
df = df.withColumn('pickup_month', month(df['tpep_pickup_datetime']))
df = df.withColumn('pickup_day', dayofmonth(df['tpep_pickup_datetime']))
df = df.withColumn('pickup_hour', hour(df['tpep_pickup_datetime']))
df = df.withColumn('pickup_day_of_week', dayofweek(df['tpep_pickup_datetime']))

df = df.withColumn('dropoff_year', year(df['tpep_dropoff_datetime']))
df = df.withColumn('dropoff_month', month(df['tpep_dropoff_datetime']))
df = df.withColumn('dropoff_day', dayofmonth(df['tpep_dropoff_datetime']))
df = df.withColumn('dropoff_hour', hour(df['tpep_dropoff_datetime']))
df = df.withColumn('dropoff_day_of_week', dayofweek(df['tpep_dropoff_datetime']))
```

Load the data

MongoDB

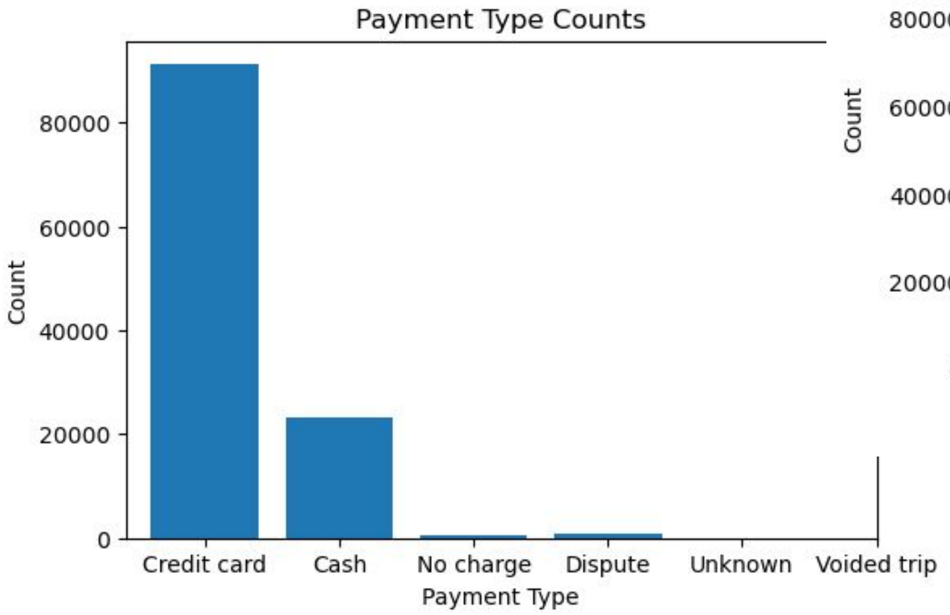
NYU

```
In [22]: df.printSchema()

root
 |-- VendorID: long (nullable = true)
 |-- tpep_pickup_datetime: timestamp (nullable = true)
 |-- tpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: double (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: double (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- PULocationID: long (nullable = true)
 |-- DOLocationID: long (nullable = true)
 |-- payment_type: long (nullable = true)
 |-- fare_amount: double (nullable = true)
 |-- extra: double (nullable = true)
 |-- mta_tax: double (nullable = true)
 |-- tip_amount: double (nullable = true)
 |-- tolls_amount: double (nullable = true)
 |-- improvement_surcharge: double (nullable = true)
 |-- total_amount: double (nullable = true)
 |-- congestion_surcharge: double (nullable = true)
 |-- airport_fee: double (nullable = true)
 |-- pickup_year: integer (nullable = true)
 |-- pickup_month: integer (nullable = true)
 |-- pickup_day: integer (nullable = true)
 |-- pickup_hour: integer (nullable = true)
 |-- pickup_day_of_week: integer (nullable = true)
 |-- dropoff_year: integer (nullable = true)
 |-- dropoff_month: integer (nullable = true)
 |-- dropoff_day: integer (nullable = true)
 |-- dropoff_hour: integer (nullable = true)
 |-- dropoff_day_of_week: integer (nullable = true)
```

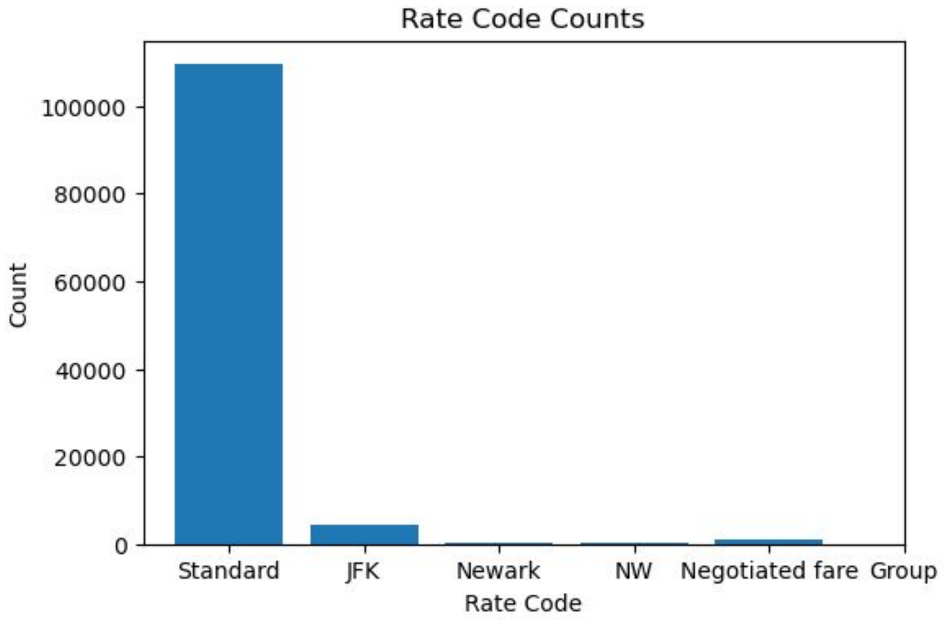
Data Exploration

We used PySpark to calculate basic statistics, identify trends and patterns, and visualize the data along with Matplotlib. This helped us understand the factors that influence taxi demand.

Trends

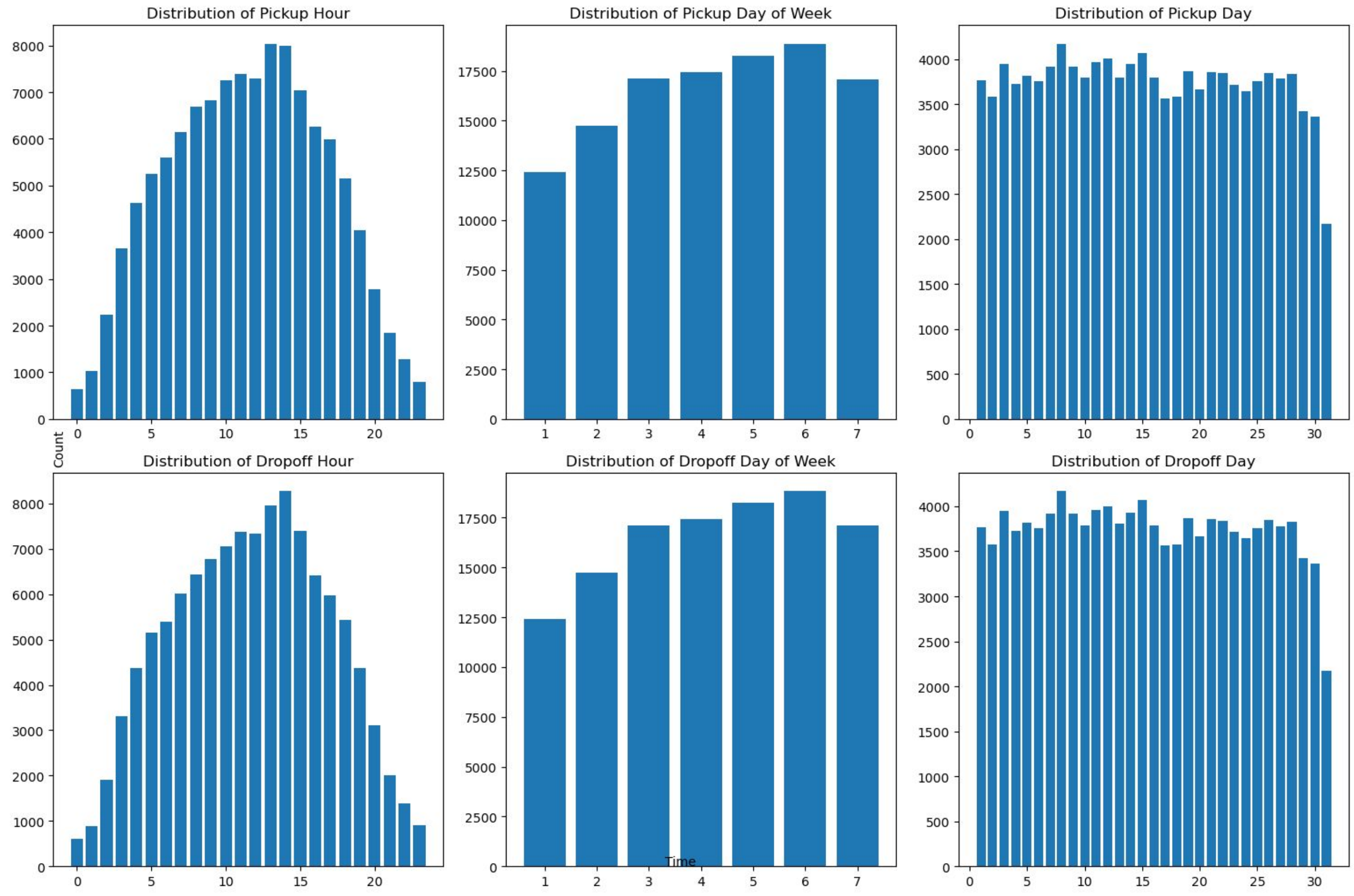


Payment Type Counts
Count
80000
60000
40000
20000
0
Credit card
Cash
No charge
Dispute
Unknown
Voided trip
Payment Type



Rate Code Counts
Count
100000
80000
60000
40000
20000
0
Standard
JFK
Newark
NW
Negotiated fare
Group
Rate Code

Patterns



Distribution of Pickup Hour
Distribution of Pickup Day of Week
Distribution of Pickup Day
Distribution of Dropoff Hour
Distribution of Dropoff Day of Week
Distribution of Dropoff Day
Count
Time

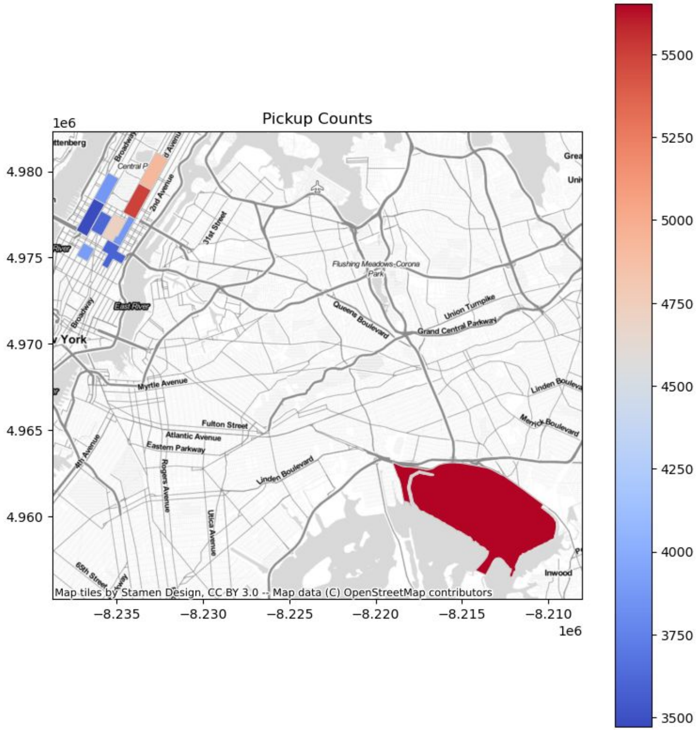
Correlations

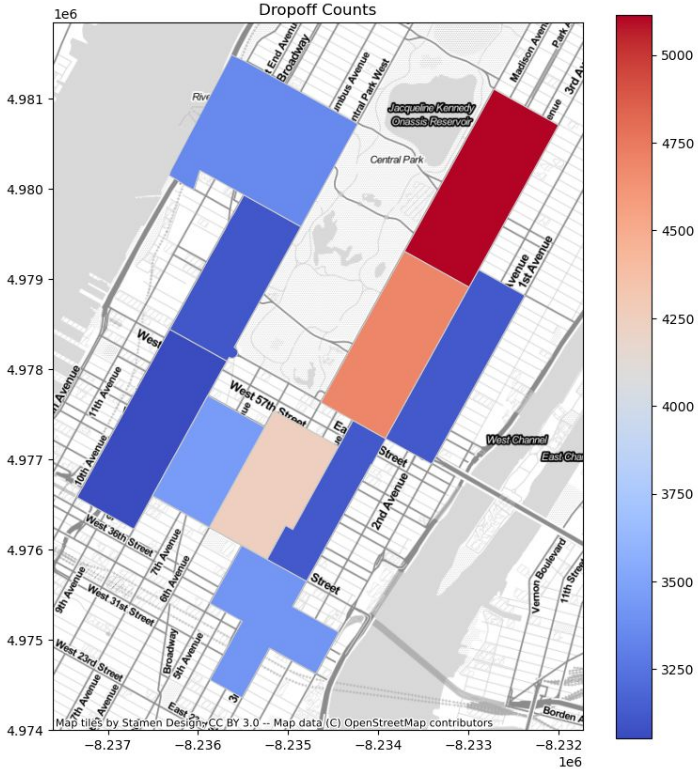
```
# Calculate the correlation between 'trip_distance' and 'fare_amount'
trip_distance_fare_amount_corr = df.stat.corr('trip_distance', 'fare_amount')
print("Correlation between trip distance and fare amount: ", trip_distance_fare_amount_corr)
```

```
23/05/09 11:57:33 WARN TaskSetManager: Stage 69 contains a task of very large size (9190 KiB). T
d task size is 1000 KiB.
[Stage 69:>                                                          (0 + 2) / 2]
Correlation between trip distance and fare amount:  0.8349329940751841
```

```
23/05/09 11:57:36 WARN TaskSetManager: Stage 70 contains a task of very large size (9190 KiB).
d task size is 1000 KiB.

Correlation between tip amount and total amount:  0.7010221277447248
23/05/09 11:57:38 WARN TaskSetManager: Stage 71 contains a task of very large size (9190 KiB).
d task size is 1000 KiB.
[Stage 71:>                                                          (0 + 2) / 2]
Correlation between trip distance and pickup hour:  0.012046207366665843
```

Top 10 pickup & drop off zones





Building the model

From the schema, the following features could be relevant for predicting the taxi fare:

- trip_distance, RatecodeID, PULocationID, DOLocationID
- pickup_hour, pickup_day, pickup_day_of_week, pickup_month
- passenger_count, extra, mta_tax, tolls_amount, improvement_surcharge, congestion_surcharge, airport_fee

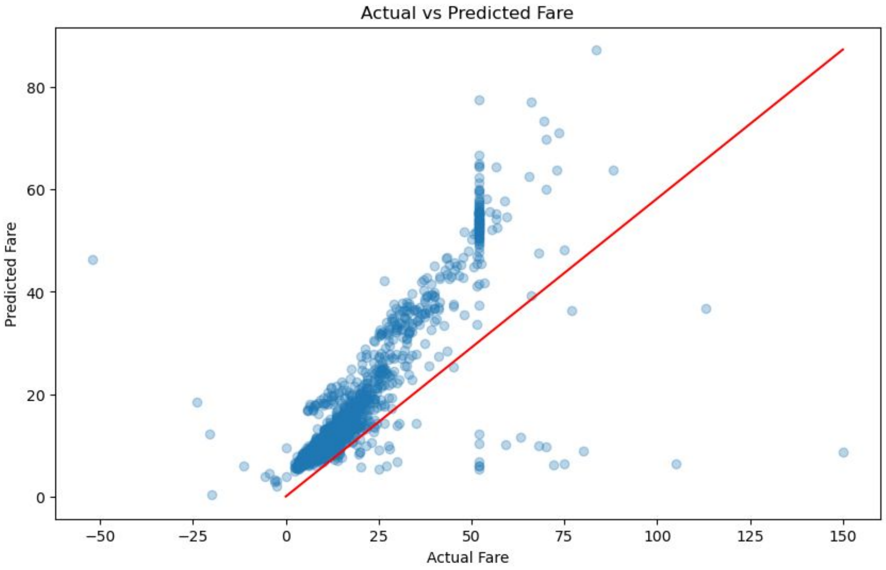
fare_amount is our target variable.

Building the model

In [18]:
```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression

feature_columns = ["trip_distance", "RatecodeID", "PULocationID", "DOLocationID",
                   "pickup_hour", "pickup_day", "pickup_day_of_week", "pickup_month",
                   "passenger_count", "extra", "mta_tax", "tolls_amount",
                   "improvement_surcharge", "congestion_surcharge", "airport_fee"]

assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
df = assembler.transform(df)

train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)

lr = LinearRegression(featuresCol="features", labelCol="fare_amount")

lr_model = lr.fit(train_data)

predictions = lr_model.transform(test_data)
```

```
23/05/09 13:34:34 WARN Instrumentation: [27522ae8] regParam is zero, which might cause
fitting.
```

Evaluation



We can further refine the model using

- Feature Engineering
- Handle Categorical Variables
- Use a Different Model
- Hyperparameter Tuning
- Cross-Validation

NYU

Experiments

```
# Train a linear regression model
model_python = sklearn.linear_model.LinearRegression()
model_python.fit(X_train, y_train)


# Evaluate the model
predictions_python = model_python.predict(X_test)

# Check the model performance
print(f'MSE: {mean_squared_error(y_test, predictions_python)}')
```

```
MSE: 64.60138133038457
```

```
In [7]: end_time = time.time()

        print('Time taken by Python implementation: ', end_time - start_time, 'seconds')
```

```
Time taken by Python implementation:  2.302728414535522 5 seconds
```

Experiments

```
lr = dask_ml.linear_modelLinearRegression()
lr.fit(X_train.values, y_train.values)
y_pred = lr.predict(X_test.values)
y_test, y_pred = dask.compute(y_test, y_pred)
print(f'MSE: {mean_squared_error(y_test, y_pred)}')
```

```
/home/jovyan/.local/lib/python3.10/site-packages/dask_ml/model_selection/_split.py:
lue for 'shuffle' must be specified when splitting DataFrames. In the future DataFr
d within blocks prior to splitting. Specify 'shuffle=True' to adopt the future beha
tain the previous behavior.
  warnings.warn(
```

```
MSE: 52.71844897649518
```

In [15]:

```
end_time = time.time()
print('Time taken by Dask implementation: ', end_time - start_time, 'seconds')
```

```
Time taken by Dask implementation:  1.675332307815518 seconds
```

Experiments

$>$

Experiments

>>>

```
In [4]: data = [("T", "H", "A", "N", "K"),
                ("Y", "O", "U", "!", " ")]
        df = spark.createDataFrame(data, ["1", "2", "3", "4", "5"])

        df.show()
```

```
+---+---+---+---+---+
|  1|  2|  3|  4|  5|
+---+---+---+---+---+
|  T|  H|  A|  N|  K|
|  Y|  O|  U|  !|   |
+---+---+---+---+---+
```

https://github.com/vchrombie/nyc-taxi-fare-prediction

Team: Sye!

1. Jaswanth Sai Nandipati - **jn2652**@nyu.edu
2. Sai Teja Reddy Parigi - **sp6923**@nyu.edu
3. Venu Vardhan Reddy Tekula - **vt2182**@nyu.edu