

# NYC Yellow Taxi Trip Analysis Report

## 1. Dataset Description

Source: 2023 NYC Yellow Taxi Trip Dataset from NYC Open Data.

This dataset includes details of yellow taxi trips, such as pickup and dropoff times, trip distances, fare amounts, and various other fees. Key columns include VendorID, pickup and dropoff times, passenger count, trip distance, fare amount, tip amount, and additional surcharges.

## 2. Data Cleaning and Preprocessing

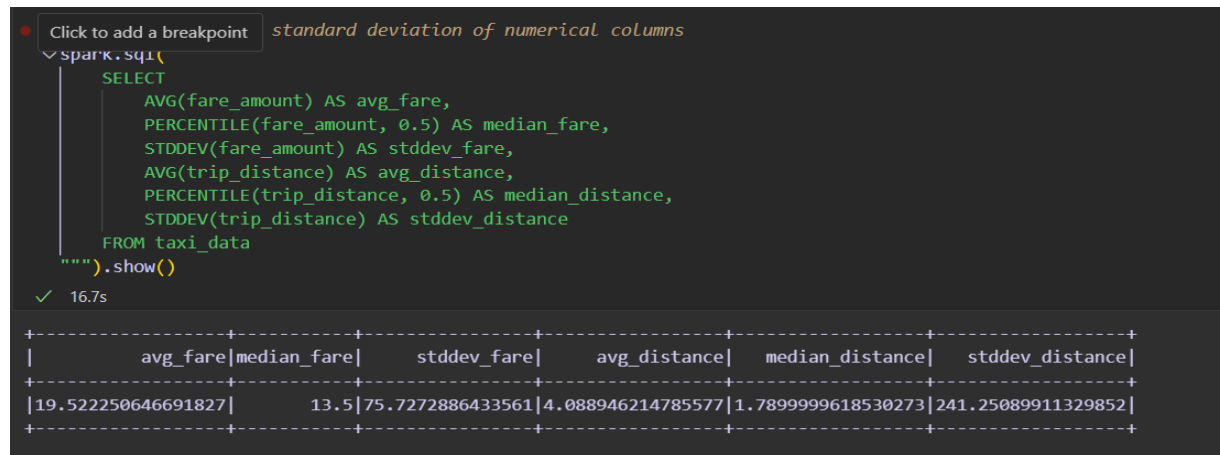
**Null Handling:** Columns with a significant number of nulls were either filled with default values or assumptions (e.g. default passenger count of 1, trip distance of 0.0 for missing entries). The dataset's datetime columns with missing values were removed for analysis consistency.

**Data Type Conversion:** Converted string columns to appropriate types, including integer and float conversions for fare amounts and trip distances.

**Outlier Treatment:** Identified extreme values in fare amounts and trip distances.

## 3. Data Analysis and Key Insights

**Statistical Summary:** Calculated average, median, and standard deviation for key numerical columns.



The screenshot shows a Jupyter Notebook interface with a Spark SQL query and its results. The query is as follows:

```
standard deviation of numerical columns
spark.sql(
    """
    SELECT
        AVG(fare_amount) AS avg_fare,
        PERCENTILE(fare_amount, 0.5) AS median_fare,
        STDDEV(fare_amount) AS stddev_fare,
        AVG(trip_distance) AS avg_distance,
        PERCENTILE(trip_distance, 0.5) AS median_distance,
        STDDEV(trip_distance) AS stddev_distance
    FROM taxi_data
    """).show()
```

The execution time is 16.7s. The results are displayed in a table with 6 columns: avg\_fare, median\_fare, stddev\_fare, avg\_distance, median\_distance, and stddev\_distance.

avg_fare	median_fare	stddev_fare	avg_distance	median_distance	stddev_distance
19.522250646691827	13.5	75.7272886433561	4.088946214785577	1.7899999618530273	241.25089911329852

### **Insights:**

Average Fare by Payment Type: Payment type 0 had the highest average fare.

payment_type	avg_fare	trip_count
0	22.373174886088183	1309356
1	19.81477939058017	29856932
2	19.388153647845137	6405059
3	8.127842588462	240862
4	1.7246593589624164	498015
5	0.0	2

**Trip Patterns by Time:** Analyzed average fare amount across pickup hours and months

```
spark.sql("""
SELECT
    HOUR(tpcp_pickup_datetime) AS pickup_hour,
    AVG(fare_amount) AS avg_fare,
    COUNT(*) AS trip_count
FROM taxi_data
GROUP BY pickup_hour
ORDER BY pickup_hour
""").show()
```

✓ 14.1s

pickup_hour	avg_fare	trip_count
NULL	19.522250646691827	38310226

```
spark.sql("""
SELECT
    MONTH(tpcp_pickup_datetime) AS pickup_month,
    AVG(fare_amount) AS avg_fare,
    COUNT(*) AS trip_count
FROM taxi_data
GROUP BY pickup_month
ORDER BY pickup_month
""").show()
```

✓ 14.0s

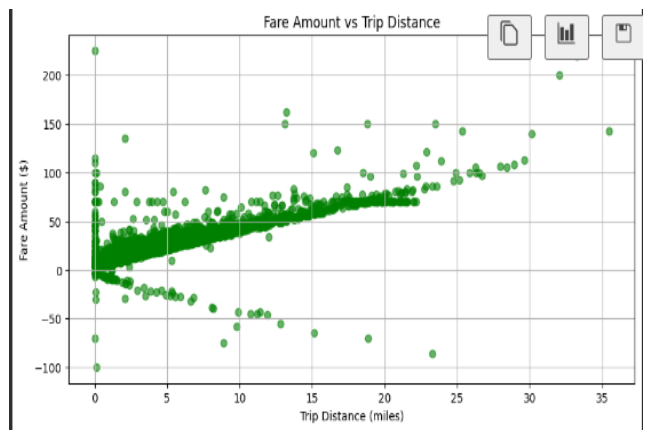
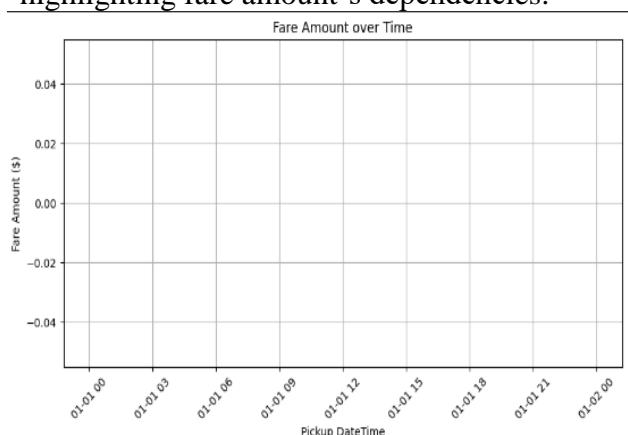
pickup_month	avg_fare	trip_count
NULL	19.522250646691827	38310226

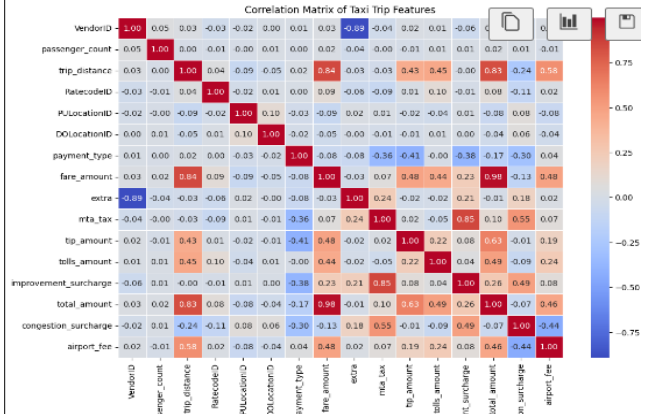
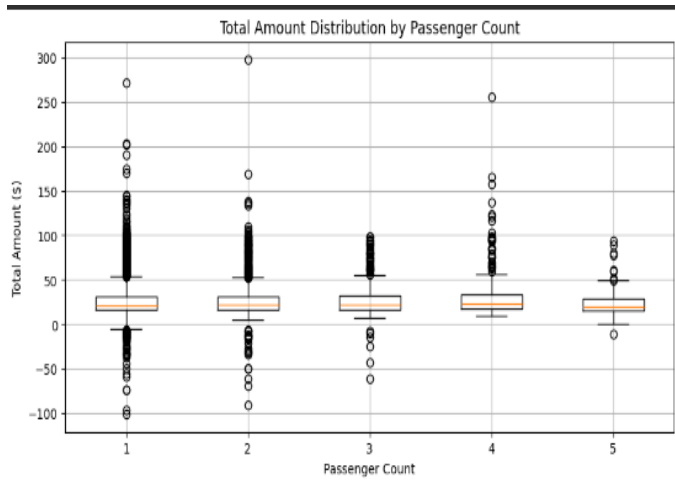
### **Visualizations:**

***Trend of Fare Amount over Time:*** Showed fare trends with timestamps.

***Fare vs. Trip Distance:*** Scatter plot illustrating the correlation between trip distance and fare amount.

***Correlation Matrix:*** Visualized correlations among numerical features, particularly highlighting fare amount's dependencies.





#### 4. Machine Learning Model Description

Build a model to predict fare amounts based on trip and passenger details.

Selected features included trip\_distance, passenger\_count, RatecodeID, PULocationID, DOLocationID, and various surcharge columns.

##### Pipeline:

Used VectorAssembler to consolidate all selected features.

Implemented a Linear Regression model with fare amount as the target variable.

```
from pyspark.ml.feature import VectorAssembler

# Define feature columns
feature_cols = ['trip_distance', 'passenger_count', 'RatecodeID', 'PULocationID', 'DOLocationID',
                'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge',
                'congestion_surcharge', 'airport_fee']

# Assemble features into a single vector
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
✓ 0.1s

from pyspark.ml.regression import LinearRegression

# Define the model
lr = LinearRegression(featuresCol="features", labelCol="fare_amount")
✓ 0.0s
```

#### 5. Model Evaluation and Performance

Data was split into training and testing sets (80-20 split)

##### Metrics:

Root Mean Squared Error (RMSE): The model's RMSE on the test set was approximately 86.56.

**Interpretation:** While the model captures some variance, improvements could be achieved by incorporating additional context such as weather data, peak hour identification, or alternative model types.

```
from pyspark.ml import Pipeline

# Define the pipeline
pipeline = Pipeline(stages=[assembler, lr])
✓ 0.0s

# Fit the pipeline on the training data
model = pipeline.fit(train_data)
✓ 2m 25.1s

from pyspark.ml.evaluation import RegressionEvaluator

# Make predictions on the test data
predictions = model.transform(test_data)

# Evaluate the model
evaluator = RegressionEvaluator(labelCol="fare_amount", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE):", rmse)
✓ 1m 6.2s

Root Mean Squared Error (RMSE): 86.55616114036957
```