Predicting NYC Taxi Trip Times with Apache Spark

SENG 550 FINAL PROJECT

Marwane Zaoudi - 30113408
Nicola Primomo - 30124581
Hassaan Sohail - 30092431
Nada Mohamed - 30183972

Table of Contents

Contributions

Marwane Zaoudi :

- Cleaned , transformed data
- Took care of building the " gradient boosted trees model "
- Worked on the report

Nicola primomo :

- Cleaned , transformed data
- Took care of building the " "
- Made the video
- Helped run the scripts on their machine (due to resource limits of other members machines)
- Worked on the report

Hassan sohail :

- Cleaned , transformed data
- Helped work on the linear regression model
- Worked on the report

Nada Mohamed:

- Cleaned , transformed data
- Worked on the report

Declaration

All the contributions were somewhat similar , we agreed to accommodate whoever was in need of more time outside the group work , since this project came in a time where everyone had finals and other group projects in parallel. *We declare that the above contributions are accurate and agreed upon by all members of the group.*

Submission Details

For this project, the primary deliverable is Jupyter Notebook scripts containing all code, analysis, and results. Since the entire workflow and findings are encapsulated within the notebooks, a public repository was not created.

Abstract

This project tackles the challenge of predicting NYC taxi trip times using data from 2013, which includes millions of records, requiring distributed processing for efficient analysis. By leveraging distributed processing with Apache Spark and machine learning models, including Linear Regression, we provide a scalable solution, as the size and complexity of the data posed challenges for memory-intensive algorithms like Gradient Boosted Trees and Random Forest. Exploratory data analysis refined feature spaces, and evaluation metrics such as RMSE and

R-squared played a critical role in assessing model performance and highlighting the suitability of Linear Regression as a resource-efficient choice. Our insights can support city planning and improve transport services.

Introduction

The problem that we decided to work on for this project was predicting the NYC taxi trip times. To do this, we utilized a publicly available dataset of NYC taxi trips from the year 2013 [1]. This dataset contained millions of records, including information such as trip time, trip distance, trip coordinates, and other relevant fields. Our aim was to analyze this dataset to gain insights and build machine learning models capable of predicting trip times effectively

Accurate trip time predictions allow passengers to plan their travel more effectively, reducing uncertainties and improving satisfaction. Research shows that accurate time estimates are key to enhancing the perceived reliability of transportation services [2]. Insights from predicted travel patterns can help city planners optimize traffic management systems, reducing congestion during peak hours. Studies highlight how traffic prediction systems, leveraging trip data, have led to a measurable decrease in city congestion levels [3].Accurate predictions allow taxi drivers to manage their schedules more effectively, deciding on high-demand routes and improving overall efficiency. Trip predictions can also support ride-hailing services in surge pricing or vehicle dispatch decisions [4].

While prior studies, including [5], have focused heavily on model accuracy, they often required complex feature engineering and computationally expensive methods, making them less feasible for real-time applications. Furthermore, many existing approaches fail to address the variations caused by unusual events such as road closures or extreme weather conditions. Our project aimed to tackle these gaps by adopting a simpler, more interpretable solution that balances scalability with accuracy. We specifically decided to emphasize clear, visual insights into the data and model predictions to make our findings accessible to non-technical stakeholders.

Initially, the group aimed to explore two machine learning models: Linear Regression, and Gradient Boosted Trees. However, challenges with memory limitations and computational resources led us to focus on Linear Regression. This model was enhanced by incorporating additional features and optimizing scalability, addressing a limitation seen in previous solutions. Evaluation metrics, including root mean squared error (RMSE) and R-squared, guided our assessments, with Linear Regression emerging as the most practical and interpretable option for our dataset size and resource constraints. By recording the performance of these models, our goal was to demonstrate a reproducible methodology for efficient trip time prediction.

Methodology

Project Setup

The project began by establishing a distributed data processing environment using PySpark within Jupyter Notebook to manage the large NYC taxi trip dataset effectively. This dataset, sourced from (*NYC Taxi Trips by Andresmh*, n.d.) [1], comprised 13 files, each containing over 40 million records. The substantial size and complexity of the dataset necessitated the use of Apache Spark for its scalability and efficient handling of large-scale data processing tasks. Apache Spark was configured in a local environment with extensions to distributed systems to ensure seamless scalability when working with larger datasets. Key configurations included `groupId = org.apache.spark`, `artifactId = spark-core_2.12`, and `version = 3.5.4`, paired with Hadoop version 3.3.

The methodology employed key PySpark libraries and tools to facilitate efficient processing and analysis. The pyspark.sql.functions module provided versatile capabilities for column transformations, aggregations, and conditional operations during the data preparation phase. The VectorAssembler utility from PySpark's ML module was critical in combining multiple feature columns into a single feature vector, optimizing input handling for the machine learning model.

While the initial scope included experimenting with Gradient-Boosted Trees (GBT), memory constraints led the team to focus on Linear Regression as the primary model. Linear Regression provided an interpretable and computationally efficient alternative. Key tools, such as `RegressionEvaluator`, were used to assess model performance with evaluation metrics like Root Mean Squared Error (RMSE) and R-squared. These tools and configurations ensured that the project's workflow remained robust, despite computational constraints.

Exploration and Cleaning

The preprocessing phase focused on ensuring data quality and relevance for downstream tasks. The raw dataset initially contained numerous columns, including [1][6]:

```
 |-- medallion: string (nullable = true)
 |-- hack_license: string (nullable = true)
 |-- vendor_id: string (nullable = true)
 |-- rate_code: integer (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- trip_time_in_secs: integer (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
```
To streamline the dataset and reduce noise, irrelevant columns such as `medallion`, `hack_license`, and `store_and_fwd_flag` were dropped [7].

Addressing missing and invalid data was a key focus; rows with `NaN` or null values were removed to avoid complications during model training. A total of 3428 rows were found with null data and filtered [8]. Trips with `trip_distance = 0` or `passenger_count = 0` were eliminated as such entries lacked meaningful predictive value. 1 126 202 and 1597 rows respectively containing 0's in these fields were found and removed [9]. Additionally, trip times were also filtered to only include trips longer than one minute than two hours to limit outliers. Trips falling in this time range are likely due to extenuating circumstances such errors with data entry or unusually long traffic. Finally, the dataset was also checked for duplicates by cross-referencing all column values. No duplicate rows were found across the dataset, confirming its integrity [10].

To uniquely identify each trip across the dataset, a new column, `id`, was introduced [11]. This addition provided a robust identifier for every record in the processed data. By the end of this exploration and cleaning phase, a comprehensive and structured master DataFrame was ready for analysis, fully loaded into Apache Spark for efficient distributed processing.

Experimentation factors

Gradient Boosted Trees (GBT) Model in Experimentation

The **" Gradient boosted trees "** GBT model was chosen due to its robustness and capacity for effective optimization. GBTs work well with high-dimensional datasets and handling outliers making it suitable for our task of predicting NYC taxi trip times [13].

In our approach, the initial dataset preprocessing involved extracting time-related features, such as the hour of the day, day of the week, and whether the day was a weekend. Interaction terms, like the combination of hour of the day and weekend indicator, were added to enhance the model's predictive power. The model was trained with initial features, including trip distance, passenger count, and temporal indicators. However, the performance was suboptimal due to the limited feature set, reflected in higher error metrics and low generalization capability [20].

To address this, spatial data—pickup and dropoff longitude and latitude values—were incorporated into the feature set, capturing geographic information relevant to trip times. These spatial features were normalized using the `MinMaxScaler` to improve model stability and convergence during training.

Additionally, hyperparameter tuning was conducted using grid search combined with cross-validation to optimize key parameters such as maximum tree depth and the number of iterations. The tuning process aimed to identify the best-performing model, evaluated on a test set using metrics such as Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared.

A key advantage of using gradient boosted trees model is that it can account for imbalanced datasets— where certain classes are underrepresented. Gradient boosted trees help in reducing model bias through fine-tuning, improving performance and accuracy in such cases [13][14]. After running the model with the expanded feature set and tuning feature importances revealed the most influential factors for predicting trip times were the spatial features (pickup and dropoff latitudes and longitudes), followed by the time related features [15].

The first complete run of the GBT model with the expanded feature set took over **8.5 hours** to train on the distributed Spark environment. Despite the lengthy runtime, memory limitations caused by insufficient partitioning in Spark and excessive RAM consumption resulted in runtime exceptions. Specifically, an `OutOfMemoryError` was triggered during the training phase, which halted further progress [21]. These challenges stemmed from the extensive computational requirements of the GBT model and the system's inability to handle the large-scale data efficiently.

Due to limited computational resources given we only had access to our laptops, efforts to resolve these issues were deemed infeasible within the project's constraints. As a result, the team decided to pivot to a more computationally efficient model, Linear Regression (LR), to ensure the completion of the task within the available resources [12]. This decision reflected a pragmatic approach, prioritizing resource feasibility over pursuing further optimization of the GBT model. While this pivot constrained some of the modeling potential inherent to GBT, it enabled the successful completion of the task within realistic project boundaries.

Linear Regression Model in Experimentation

A Linear Regression model was ultimately chosen due to its simplicity, interpretability, and computational efficiency. It was well-suited to the project's constraints, allowing the team to predict taxi trip times as a function of multiple variables such as trip distance, day of the week, and hour of the day [12]. This model provided a feasible alternative after resource limitations prevented further experimentation with Gradient-Boosted Trees.

In our initial approach of the linear regression model, the dataset was processed to extract new time based features using the pickup_datetime which was already included in the dataset. The pickup_datetime was used to extract the features called pickup_dayofweek and pickup_hour to provide more time relevant features for a higher accuracy.

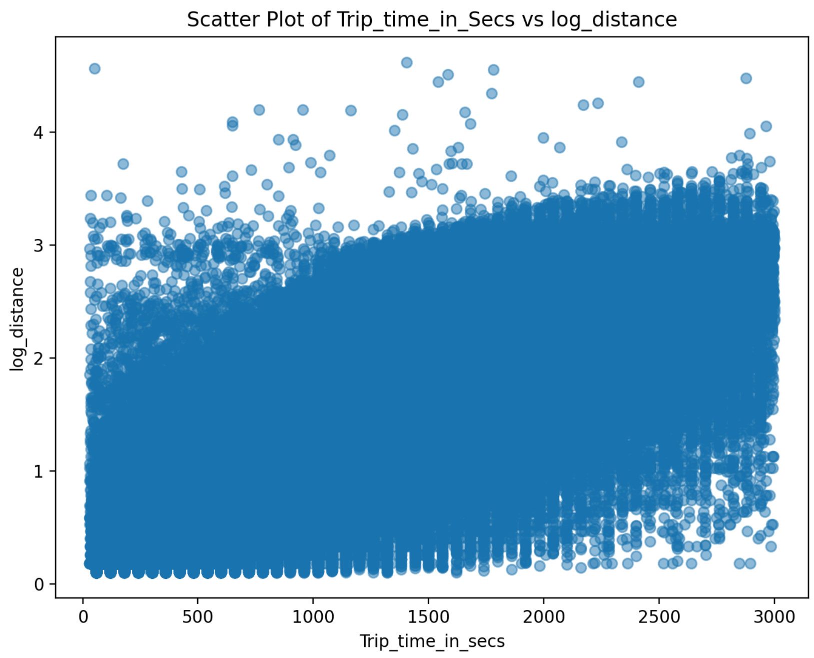
The new two time based features and two already included features in the dataset (trip_distance, passenger_count) were then assembled together by using the pyspark vectorassembler to create a feature column which would be compared with our independent variable (trip_time_in_secs).

After the features and independent variable were set, the dataset was split into training and testing dataset which was done 80% and 20% respectively. Following this, the regression model was built using the LinearRegression feature in pySpark and was trained on the training dataset.

Once the model had been trained, it was then evaluated on the test dataset and the performance metrics were printed to show the results of the model. The performance metrics that we decided to use were root mean squared error (RMSE), R-squared (r2), and mean absolute error (MAE). Here are the results of the initial linear regression model that was created:
RMSE: 882.23~ R-Squared: -1663154.3457~ MAE: 769.485~

What stood out to us from these results, was the R-squared value which ended up being extremely negative. This negative value means that the model is not being evaluated on the dataset it was fitted on and is producing extremely poor results [17]. This meant that we would have to go back and adjust our model to produce the results that we desire.

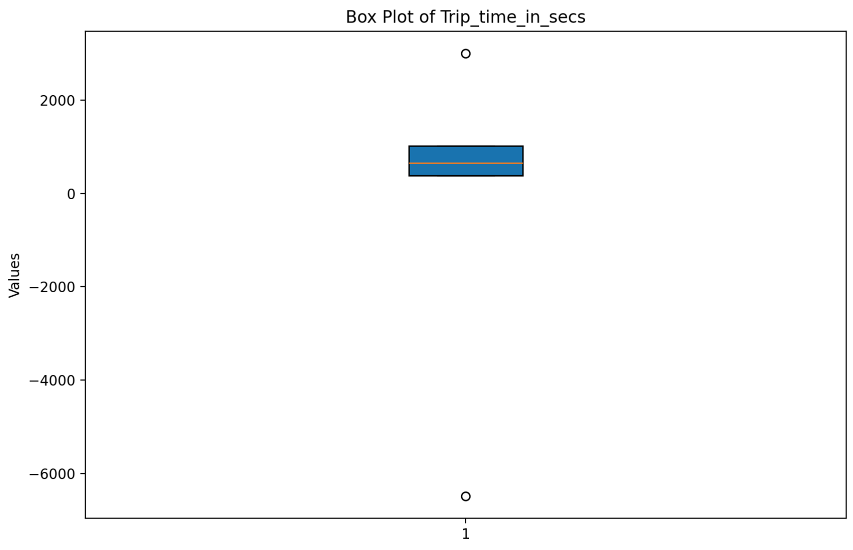
The first change we decided to implement in the updated model, was to add more features so that a higher variance would be produced with the independent variable resulting in a more accurate R-squared value. We extracted new time relevant features such as pickup_hour, pickup_minute, pickup_month, and pick_up features for all seven days of the week to have the most time based features as possible. We also decided to implement the geographical features such as pickup and dropoff longitude and latitude features as these spatial features would also have an impact on the trip time. Another change that we implemented to one of the features was trip_distance. Instead of using the raw distance values from the dataset, this was changed to use the log values of the trip_distance as an analysis of the dataset showed a more linear trend between the log_distance and trip_time_in_secs.



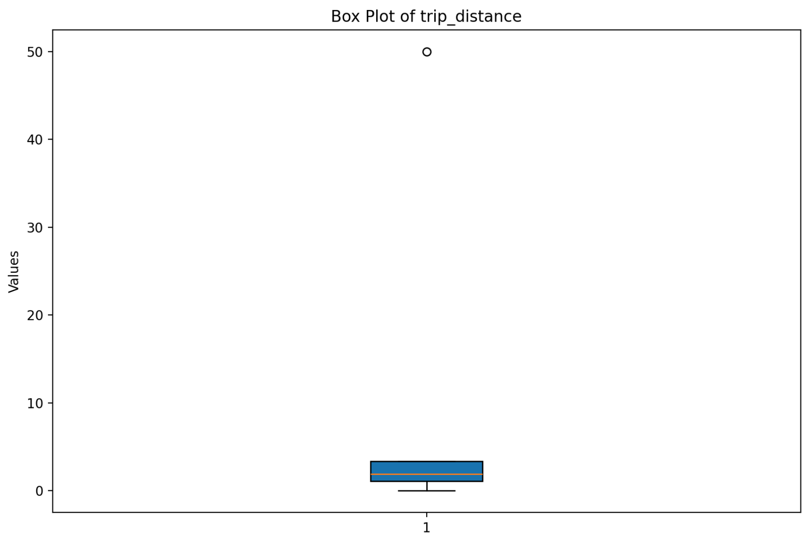
Scatter plot of Trip_time_in_Secs vs log_distance created using Dask and Matplotlib

It can be observed from this scatter plot that there is a linear relationship between the variables trip_time_in_secs and log_distance. A gradual linear increase is observed in the values of log_distance as the trip time increases. Furthermore, we can use the Pearson correlation coefficient to calculate whether there truly is a linear relationship between the two variables. The Pearson correlation coefficient is the measure of the linear relationship between two variables, where a variable of r = 1 means a perfect linear relationship [18]. We calculated the Pearson coefficient of log_distance vs trip_time_in_secs in Pyspark using a script [19]. The result of this calculation was the value r = 0.79594~, which is a high enough value to suggest a strong linear relationship between the two variables. This would mean that we can proceed with using the log_distance in a linear regression model.

Another issue that we fixed in our refined model was to handle outliers in the dataset, specifically outliers in the trip time values and the trip distance values. Having these outliers present was resulting in a skewness of the data which we realized was negatively impacting the predictions and accuracy of the model. Pyspark scripts were used to get rid of these outliers so that the data being used in the model was less skewed.



Box plot of the Trip times in seconds created using Dask and mathplotlib
From this box plot, it is evident that most of the trip times are very short, usually ranging between 30 seconds to 2000 seconds (Approximately 33 minutes). With this in mind we can see that a few very large outliers exist in the dataset which are most likely anomalies and are skewness in the data. We can also see an outlier that is a very large negative value, which is also causing the data to be extremely skewed and needs to be removed as having a negative trip time value is not feasible.



Box plot of the Trip distances created using Dask and mathplotlib

As we can see from this box plot of the trip distances, most of the trips in the dataset are of short distances, generally ranging between 0 and 10. It is evident that there is an extremely large outlier which is causing the data to get skewed and producing accurate results. From the analysis of this box plot, we were able to determine the outliers to remove for trip distances.

The final change made to the refined model was to use the StandardScaler feature in pySpark to standardize the features to reduce convergence and normalize the data. With all of these changes implemented, the refined model was complete and was then evaluated once more on the the test dataset, resulting in the following values for the performance metrics:
RMSE: 336.915~ R-Squared: 0.6344~ MAE: 224.534~

Results

This project highlights the challenges and opportunities in predicting NYC taxi trip times using large-scale datasets and distributed processing frameworks like Apache Spark. While our initial intent was to implement both Gradient Boosted Trees (GBT) and Linear Regression (LR) models, resource constraints limited the practical viability of the GBT model. Despite its theoretical advantages and initial promise, issues like prolonged runtime and memory limitations during training made it infeasible within our project scope.

Linear Regression, on the other hand, proved to be an efficient and interpretable alternative. By leveraging robust feature engineering, including spatial, temporal, and interaction features, and optimizing preprocessing, the LR model delivered competitive performance metrics, making it a practical choice for scalable trip time predictions.

Our findings underscore the importance of balancing model complexity, computational resources, and interpretability when working with real-world data. Future work could explore enhancements like optimizing Spark configurations, integrating external data sources, or deploying alternative scalable machine learning frameworks to revisit models like GBT. Overall, this project demonstrates the potential of data-driven insights to improve transportation planning and passenger experience.

References

[1] NYC Taxi Trips by andresmh. (n.d.). https://www.andresmh.com/nyctaxitrips/

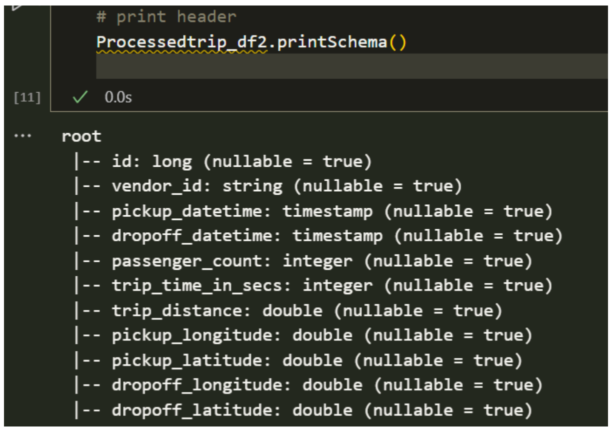
[2] Kim, H. et al., 2021, Transportation Reliability Study. Transportation Mode Detection Technology to Predict Wheelchair Users' Life Satisfaction in Seoul, South Korea | Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies

[3] Zheng, Y., Wu, W., Chen, Y., Qu, H., & Ni, L. M. (2016). Visual analytics in urban computing: An overview. IEEE Transactions on Big Data, 2(3), 276-296. Visual Analytics in Urban Computing: An Overview | IEEE Journals & Magazine | IEEE Xplore

[4] Jin, G., Cui, Y., Zeng, L., Tang, H., Feng, Y., & Huang, J. (2020). Urban ride-hailing demand prediction with multiple spatio-temporal information fusion network. *Transportation Research Part C: Emerging Technologies*, *117*, 102665. Urban ride-hailing demand prediction with multiple spatio-temporal information fusion network - ScienceDirect

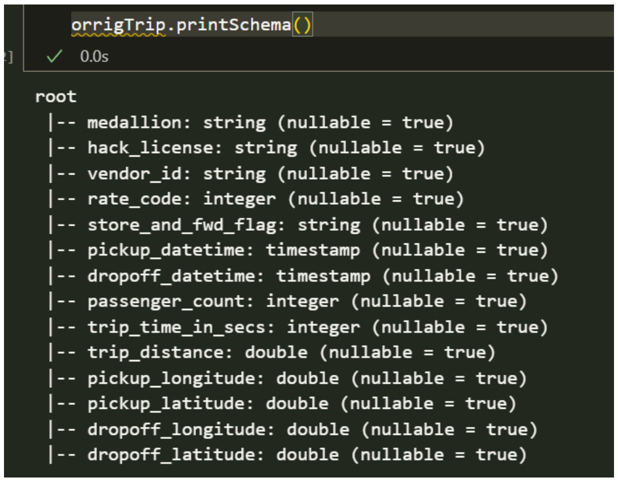
[5] Meg Risdal. New York City Taxi Trip Duration. https://kaggle.com/competitions/nyc-taxi-trip-duration, 2017. Kaggle.

[6]



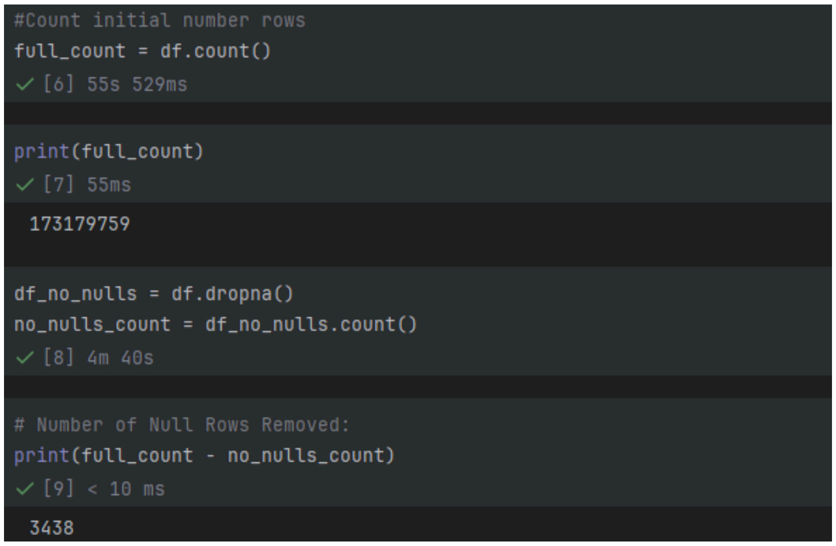
```
# print header
Processedtrip_df2.printSchema()

[11]  ✓ 0.0s

...  root
 |-- id: long (nullable = true)
 |-- vendor_id: string (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- trip_time_in_secs: integer (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
```

[7]



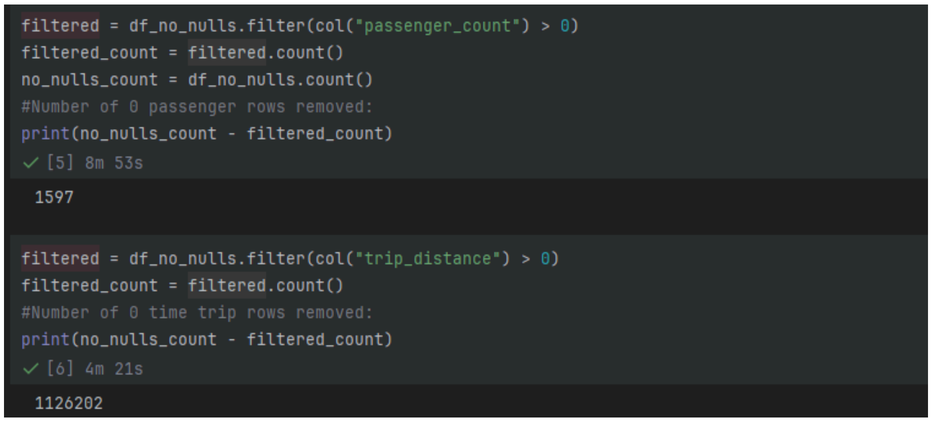
```
orrigTrip.printSchema()

✓ 0.0s

root
 |-- medallion: string (nullable = true)
 |-- hack_license: string (nullable = true)
 |-- vendor_id: string (nullable = true)
 |-- rate_code: integer (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- trip_time_in_secs: integer (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- pickup_longitude: double (nullable = true)
 |-- pickup_latitude: double (nullable = true)
 |-- dropoff_longitude: double (nullable = true)
 |-- dropoff_latitude: double (nullable = true)
```

[8]



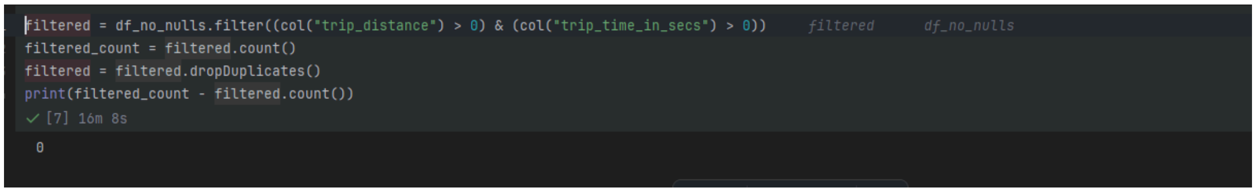
```
#Count initial number rows
full_count = df.count()
✓ [6] 55s 529ms

print(full_count)
✓ [7] 55ms

 173179759

df_no_nulls = df.dropna()
no_nulls_count = df_no_nulls.count()
✓ [8] 4m 40s

# Number of Null Rows Removed:
print(full_count - no_nulls_count)
✓ [9] < 10 ms

 3438
```

[9]



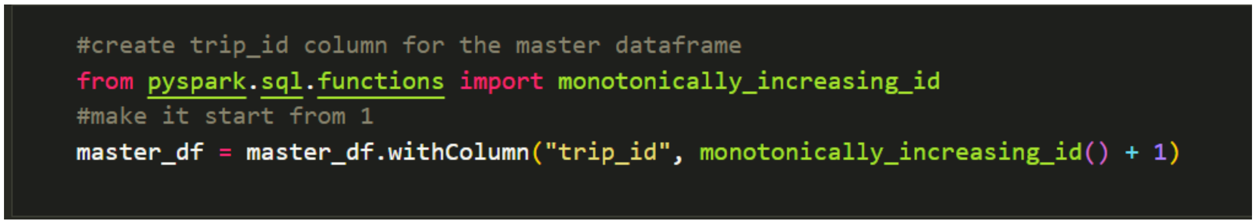
```
filtered = df_no_nulls.filter(col("passenger_count") > 0)
filtered_count = filtered.count()
no_nulls_count = df_no_nulls.count()
#Number of 0 passenger rows removed:
print(no_nulls_count - filtered_count)
✓ [5] 8m 53s

 1597

filtered = df_no_nulls.filter(col("trip_distance") > 0)
filtered_count = filtered.count()
#Number of 0 time trip rows removed:
print(no_nulls_count - filtered_count)
✓ [6] 4m 21s

 1126202
```

[10]



```
filtered = df_no_nulls.filter((col("trip_distance") > 0) & (col("trip_time_in_secs") > 0))     filtered     df_no_nulls
filtered_count = filtered.count()
filtered = filtered.dropDuplicates()
print(filtered_count - filtered.count())
✓ [7] 16m 8s

 0
```

[11]



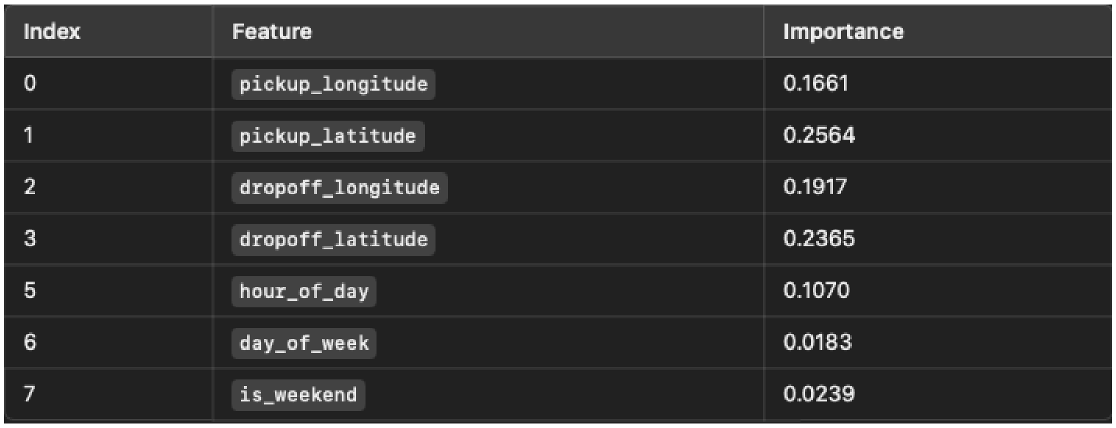
```
#create trip_id column for the master dataframe
from pyspark.sql.functions import monotonically_increasing_id
#make it start from 1
master_df = master_df.withColumn("trip_id", monotonically_increasing_id() + 1)
```

[12] Linear regression with PySpark

[13] PySpark Gradient Boosting model – Building and Evaluating Gradient Boosting model using PySpark MLlib: A Step-By-Step Guide - Machine Learning Plus

[14]
https://medium.com/@wilbossoftwarejourney/random-forest-vs-gradient-boosted-trees-pros-and-cons-8c1feec0ea0d#:~:text=This%20technique%20reduces%20bias%20and,algorithm%20for%20real%2Dworld%20applications.

[15]



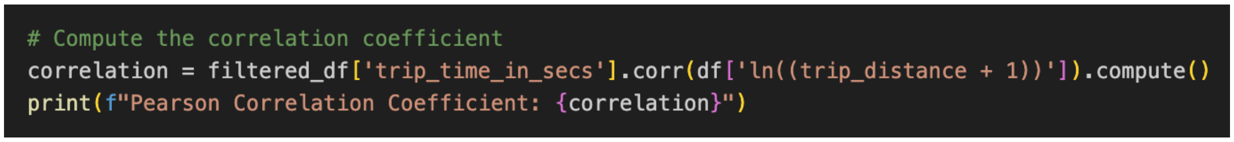
Index	Feature	Importance
0	`pickup_longitude`	0.1661
1	`pickup_latitude`	0.2564
2	`dropoff_longitude`	0.1917
3	`dropoff_latitude`	0.2365
5	`hour_of_day`	0.1070
6	`day_of_week`	0.0183
7	`is_weekend`	0.0239

[16] https://www.ibm.com/think/topics/linear-regression

[17] https://towardsdatascience.com/explaining-negative-r-squared-17894ca26321

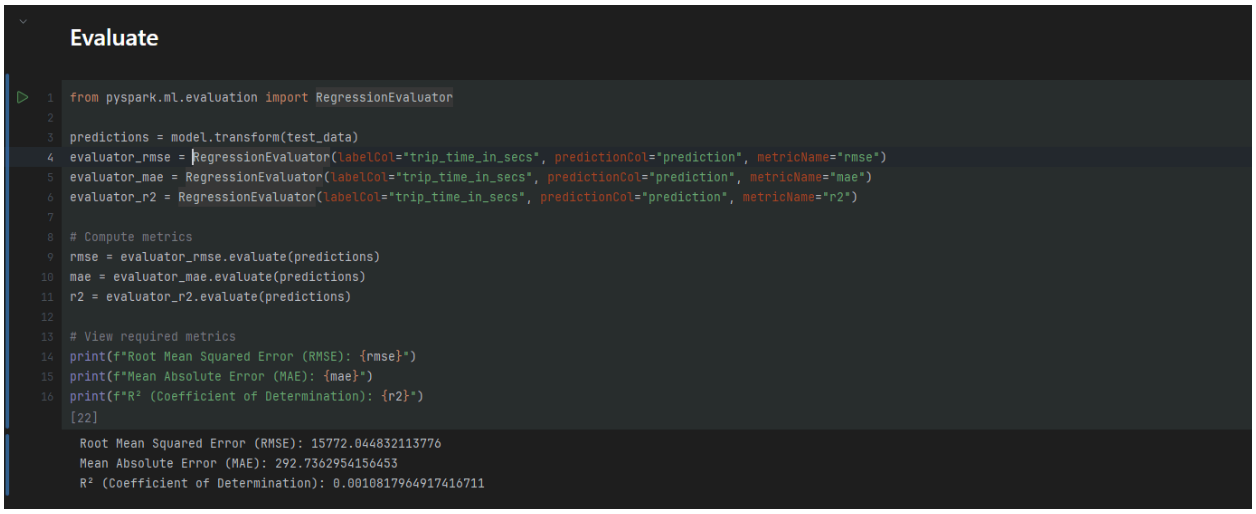
[18] https://www.socscistatistics.com/tests/pearson/#google_vignette

[19]



```
# Compute the correlation coefficient
correlation = filtered_df['trip_time_in_secs'].corr(df['ln((trip_distance + 1))']).compute()
print(f"Pearson Correlation Coefficient: {correlation}")
```

[20]

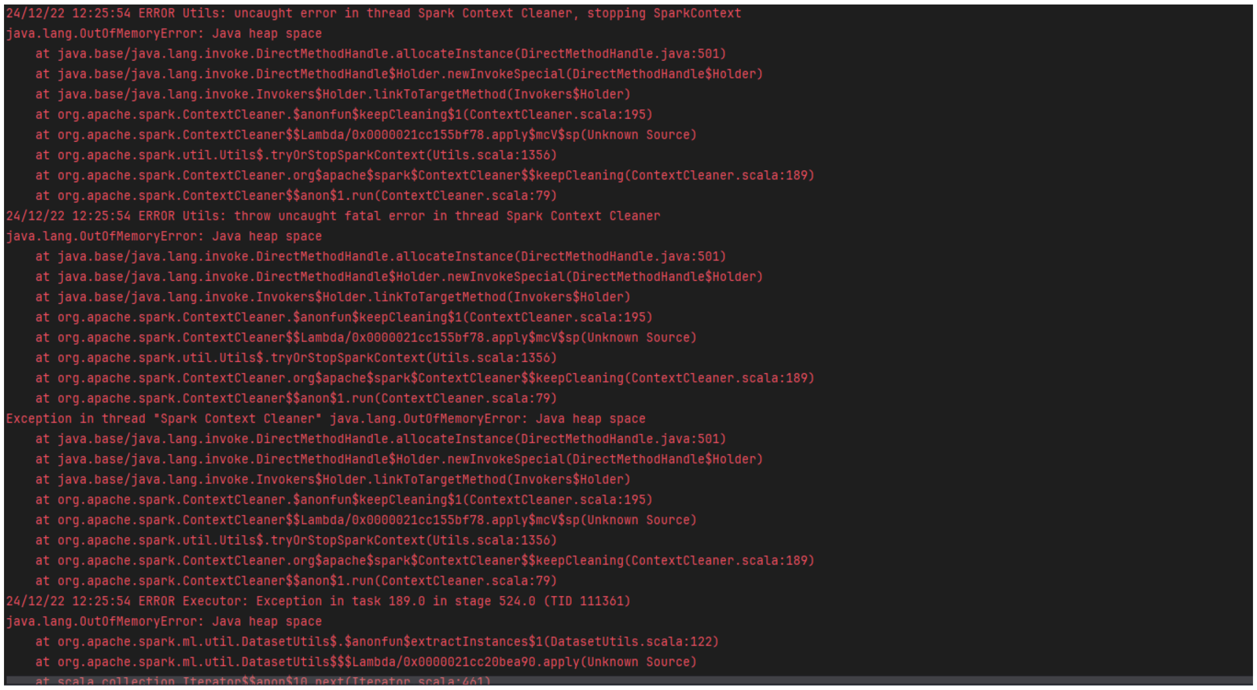


Evaluate

```
from pyspark.ml.evaluation import RegressionEvaluator

predictions = model.transform(test_data)
evaluator_rmse = RegressionEvaluator(labelCol="trip_time_in_secs", predictionCol="prediction", metricName="rmse")
evaluator_mae = RegressionEvaluator(labelCol="trip_time_in_secs", predictionCol="prediction", metricName="mae")
evaluator_r2 = RegressionEvaluator(labelCol="trip_time_in_secs", predictionCol="prediction", metricName="r2")

# Compute metrics
rmse = evaluator_rmse.evaluate(predictions)
mae = evaluator_mae.evaluate(predictions)
r2 = evaluator_r2.evaluate(predictions)

# View required metrics
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R² (Coefficient of Determination): {r2}")
[22]
```

```
Root Mean Squared Error (RMSE): 15772.044832113776
Mean Absolute Error (MAE): 292.7362954156453
R² (Coefficient of Determination): 0.0010817964917416711
```

[21]



```
24/12/22 12:25:54 ERROR Utils: uncaught error in thread Spark Context Cleaner, stopping SparkContext
java.lang.OutOfMemoryError: Java heap space
    at java.base/java.lang.invoke.DirectMethodHandle.allocateInstance(DirectMethodHandle.java:501)
    at java.base/java.lang.invoke.DirectMethodHandle$Holder.newInvokeSpecial(DirectMethodHandle$Holder)
    at java.base/java.lang.invoke.Invokers$Holder.linkToTargetMethod(Invokers$Holder)
    at org.apache.spark.ContextCleaner.$anonfun$keepCleaning$1(ContextCleaner.scala:195)
    at org.apache.spark.ContextCleaner$$Lambda/0x0000021cc155bf78.apply$mcV$sp(Unknown Source)
    at org.apache.spark.util.Utils$.tryOrStopSparkContext(Utils.scala:1356)
    at org.apache.spark.ContextCleaner.org$apache$spark$ContextCleaner$$keepCleaning(ContextCleaner.scala:189)
    at org.apache.spark.ContextCleaner$$anon$1.run(ContextCleaner.scala:79)
24/12/22 12:25:54 ERROR Utils: throw uncaught fatal error in thread Spark Context Cleaner
java.lang.OutOfMemoryError: Java heap space
    at java.base/java.lang.invoke.DirectMethodHandle.allocateInstance(DirectMethodHandle.java:501)
    at java.base/java.lang.invoke.DirectMethodHandle$Holder.newInvokeSpecial(DirectMethodHandle$Holder)
    at java.base/java.lang.invoke.Invokers$Holder.linkToTargetMethod(Invokers$Holder)
    at org.apache.spark.ContextCleaner.$anonfun$keepCleaning$1(ContextCleaner.scala:195)
    at org.apache.spark.ContextCleaner$$Lambda/0x0000021cc155bf78.apply$mcV$sp(Unknown Source)
    at org.apache.spark.util.Utils$.tryOrStopSparkContext(Utils.scala:1356)
    at org.apache.spark.ContextCleaner.org$apache$spark$ContextCleaner$$keepCleaning(ContextCleaner.scala:189)
    at org.apache.spark.ContextCleaner$$anon$1.run(ContextCleaner.scala:79)
Exception in thread "Spark Context Cleaner" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.lang.invoke.DirectMethodHandle.allocateInstance(DirectMethodHandle.java:501)
    at java.base/java.lang.invoke.DirectMethodHandle$Holder.newInvokeSpecial(DirectMethodHandle$Holder)
    at java.base/java.lang.invoke.Invokers$Holder.linkToTargetMethod(Invokers$Holder)
    at org.apache.spark.ContextCleaner.$anonfun$keepCleaning$1(ContextCleaner.scala:195)
    at org.apache.spark.ContextCleaner$$Lambda/0x0000021cc155bf78.apply$mcV$sp(Unknown Source)
    at org.apache.spark.util.Utils$.tryOrStopSparkContext(Utils.scala:1356)
    at org.apache.spark.ContextCleaner.org$apache$spark$ContextCleaner$$keepCleaning(ContextCleaner.scala:189)
    at org.apache.spark.ContextCleaner$$anon$1.run(ContextCleaner.scala:79)
24/12/22 12:25:54 ERROR Executor: Exception in task 189.0 in stage 524.0 (TID 111361)
java.lang.OutOfMemoryError: Java heap space
    at org.apache.spark.ml.util.DatasetUtils$.$anonfun$extractInstances$1(DatasetUtils.scala:122)
    at org.apache.spark.ml.util.DatasetUtils$$$Lambda/0x0000021cc20bea90.apply(Unknown Source)
    at scala.collection.Iterator$$anon$10.next(Iterator.scala:461)
```