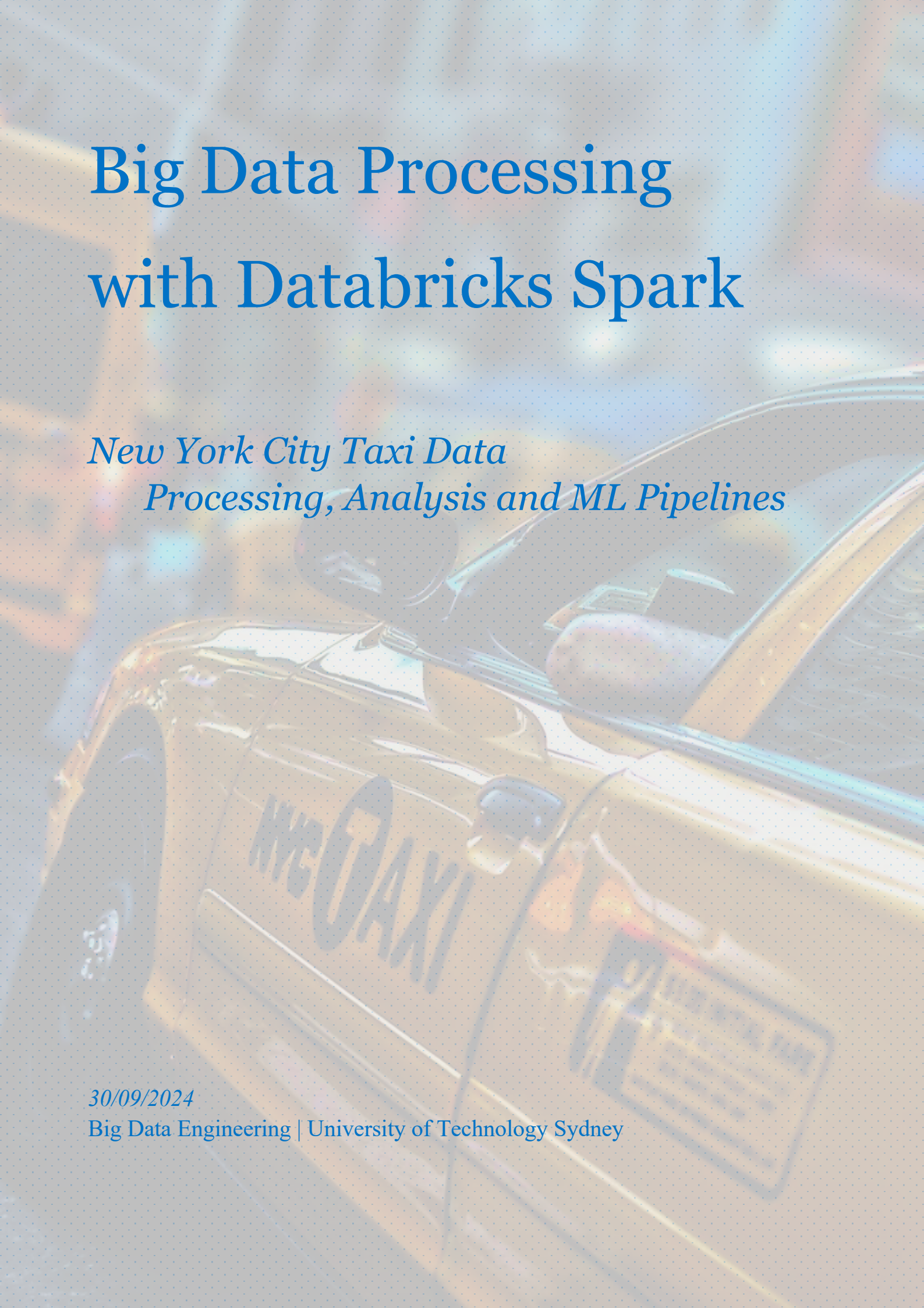
Big Data Processing with Databricks Spark

New York City Taxi Data Processing, Analysis and ML Pipelines



30/09/2024

Big Data Engineering | University of Technology Sydney

TABLE OF CONTENTS

1. Introduction

1.1 Brief Introduction

As an agency of New York City Government responsible for licenses and regulations of taxi industry, The New York City Taxi and Limousine Commission (TLC) has been publishing billions of taxi records over the years. This project aims to extract data from TLC parquet files stored in Azure cloud storage, followed by data processing, business analysis and construction of machine learning pipelines within the context of big data, using Spark on Databricks. The objective is to gain key business insights and provide model suggestions on predicting trip fare.

1.2 About the Dataset

Two datasets obtained from TLC are used in this project, yellow taxi with 663,055,251 records and green taxi with 66,200,401 records, from 2015 to 2022. The datasets contain fields such as trip start and end times, distances, driver-enter passenger count, and fare and payment information.

1.3 Executive Architecture

The executive architecture of this project is illustrated by Fig1. Source files are stored in Azure Container. Data extraction, Transformation, and Loading (ETL) processes are performed on Databricks. These processes are done to serve the applications, including business analysis through SparkSQL and machine learning pipelining with Pyspark.

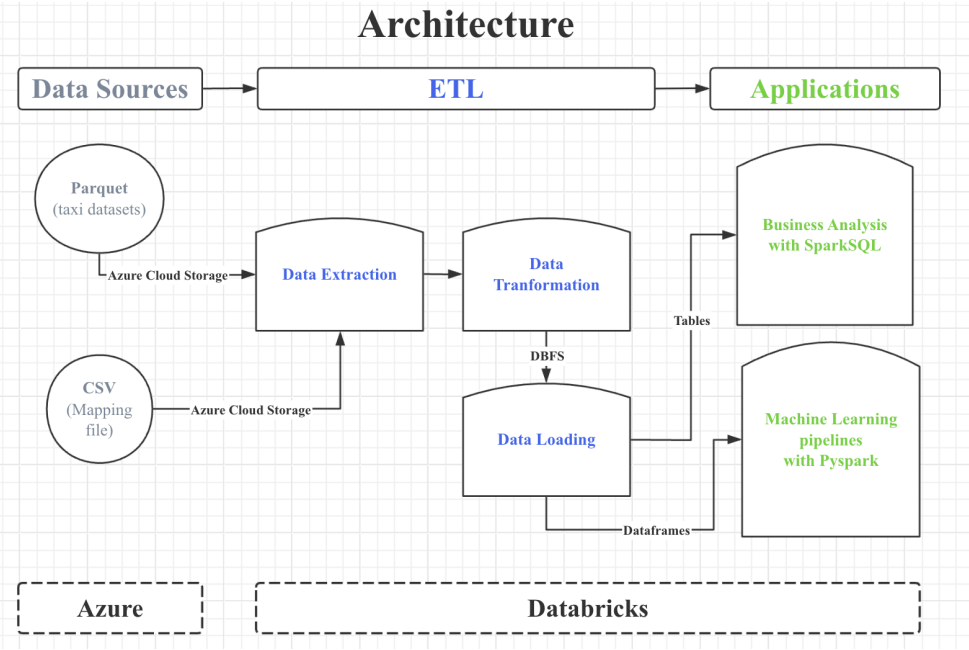


Fig 1. Project Architecture

2. Setup

The project started with setting up both Azure and Databricks environments. The original data parquet files from TCL were stored in Azure's container and it was connected to Databricks via storage account access key, along with storage account name and blog container name. Once connected, the blob container in Azure was mounted onto Databricks File System (DBFS).

2.1 Azure

1. Downloaded the yellow taxi and green taxi parquet files, and taxi zone reference CSV file.
2. Created a new container called "bde-tlc" on Azure; Uploaded the downloaded files into "bde-tlc" container.
3. Generated Access Key by navigating to Access keys from Security + networking on Azure storage account.

2.2 Databricks

Corresponding code source: NYC_taxi_data_Project.ipynb (Part1, 1.1.1)

1. Mounted the parquet files from blob container into Databricks File System by configuring Azure's storage account name, storage account access key and blob container name and specifying mount point.
2. Moved the mounted files to Databricks File System FileStore directory to avoid retrieving data from Azure repetitively, which may increase Azure credit usage.
3. Displayed all files in mount point to check if files were copied successfully to DBFS FileStore directory (Fig 2). Finally, unmount directory bde-tlc from Azure container.

path	name	size	modificationTime
dbfs:/FileStore/green/	green/	0	0
dbfs:/FileStore/taxi_zone_lookup.csv	taxi_zone_lookup.csv	12322	1726367759000
dbfs:/FileStore/yellow/	yellow/	0	0

path	name	size	modificationTime
dbfs:/FileStore/yellow/yellow_taxi_2015.parquet	yellow_taxi_2015.parquet	2896588873	1727089667000
dbfs:/FileStore/yellow/yellow_taxi_2016.parquet	yellow_taxi_2016.parquet	2616839324	1727089995000
dbfs:/FileStore/yellow/yellow_taxi_2017.parquet	yellow_taxi_2017.parquet	2072273391	1727090261000
dbfs:/FileStore/yellow/yellow_taxi_2018.parquet	yellow_taxi_2018.parquet	2098068908	1727090550000
dbfs:/FileStore/yellow/yellow_taxi_2019.parquet	yellow_taxi_2019.parquet	1788371641	1727090783000
dbfs:/FileStore/yellow/yellow_taxi_2020.parquet	yellow_taxi_2020.parquet	533759081	1727090848000
dbfs:/FileStore/yellow/yellow_taxi_2021.parquet	yellow_taxi_2021.parquet	678942509	1727090954000
dbfs:/FileStore/yellow/yellow_taxi_2022.parquet	yellow_taxi_2022.parquet	875304042	1727091118000

path	name	size	modificationTime
dbfs:/FileStore/green/green_taxi_2015.parquet	green_taxi_2015.parquet	404556105	1727088212000
dbfs:/FileStore/green/green_taxi_2016.parquet	green_taxi_2016.parquet	346731598	1727088338000
dbfs:/FileStore/green/green_taxi_2017.parquet	green_taxi_2017.parquet	251504885	1727088664000
dbfs:/FileStore/green/green_taxi_2018.parquet	green_taxi_2018.parquet	193874396	1727088872000
dbfs:/FileStore/green/green_taxi_2019.parquet	green_taxi_2019.parquet	142163406	1727088932000
dbfs:/FileStore/green/green_taxi_2020.parquet	green_taxi_2020.parquet	37076741	1727088992000
dbfs:/FileStore/green/green_taxi_2021.parquet	green_taxi_2021.parquet	23479865	1727089068000
dbfs:/FileStore/green/green_taxi_2022.parquet	green_taxi_2022.parquet	20085692	1727089117000

Fig 2. Flies List in DBFS

3. Data Processing

After the data in Azure was copied into DBFS, the datasets were ready to perform ETL processes. Details are as follows:

3.1 Data Extraction

Corresponding code source: NYC_taxi_data_Project .ipynb (Part1, 1.1)

1. Data ingestion:
    a) Read the yellow parquet files and green parquet files as dataframes from DBFS and counted the number of rows for each taxi color;
    b) Read the location reference csv file as a dataframe map_loc.
2. Comparing data sizes: Read the parquet file "green_taxi_2015.parquet", then converted it into a csv file called "green_taxi_2015.csv". Pushed the CSV file to the same Azure Blob Container, and compared the data sizes between parquet file and CSV file (Fig 3), where the CSV file size was five times larger than the parquet file, showcasing the parquet file's memory efficiency.

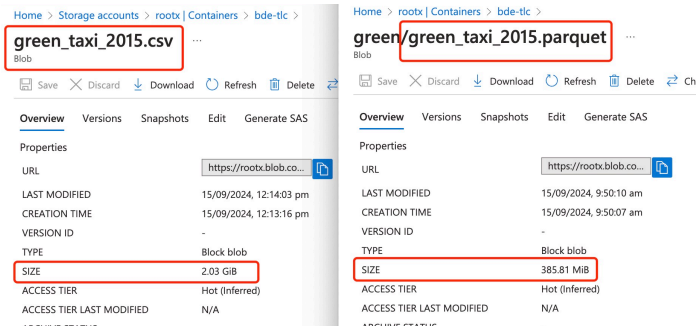


Fig 3. Size Comparison Between Parquet and CSV Files

3.2 Data Transformation

Corresponding code source: NYC_taxi_data_Project.ipynb (Part1, 1.2)

Once the data was extracted into DBFS and read as dataframes, it is essential to check invalid records, and combining datasets altogether before starting business analysis and machine learning process. The records might be invalid or unrealistic due to entry errors, system issues, GPS errors or immediately cancelation. Below are the steps implemented:

1. Created new columns “speed_mph” and “duration_min”, which are basic metrics under the taxi business context.
2. Filtered out trips where drop-off time was before the pick-up time.
3. Filtered out trips outside of valid time range, where the pick-up datetime was after 2022 or drop-off datetime was before 2015.
4. Filtered out trips with negative speed.
5. Filtered out trips above speed limits:
    a) For trips within NYC, the Speed limit is 25 mph (40.2 km per hour): Removed trips with speed greater than 25mph, when pick-up location and drop-off location were both in NYC.
    b) For trips involving airport freeway, the speed limit is 65 mph (106.4 km per hour): Removed trips with speed greater than 65 mph, when pick-up location or drop-off location involved airports within NYC (LocationID: 132, 138) or airport outside of NYC (LocationID: 1). Unknown location (LocationID: 264, 265), was assumed to be locations outside of NYC.
6. Filtered out trips with unrealistic durations, where the duration was less than 30s or more than 6h.
7. Filtered out trips with unrealistic distances, where the distance was less than 0.2 miles (0.32 km) or greater than 60 miles (96.5 km).
8. Fixed driver-entered value “passenger_count” by assigning null to values less than 1 or more than 6.
9. Flagged trips with zero or negative “total_amount”. The zero total amount could be the trips involving promotions and the negative values could be refunded trips, or they could be error data. They needed to be flagged and removed in business analysis and ML training process, as they could be noises.
10. Data combining:

a) Renamed varying column names in yellow and green taxi datasets regarding pick-up and drop-off datetime to a consistent name.

b) Set a flag column taxi_color to identify taxi types and data sources.

c) Combined the yellow and green taxi datasets, allowing missing columns.

d) Created new columns “PU_year” and “PU_month”, which could be used for future data partition or data split.

e) Transformed mile units to km in relevant columns and cast data to appropriate types to optimize memory usage.

11. Data mapping: Mapped both “PULocationID” and “DOLocationID” with map_loc, and removed redundant columns.

3.3 Data Loading

1. Data Loading: Broke the mapped dataframe into smaller chunks by year, and wrote them into parquet files by iterating each chunk.
2. Load data as table: Read the parquet file into table “nyc_taxi_data” with certain conditions, filtering out non-positive total amount to avoid noisy points in business analysis.

4. Business Analysis

Corresponding code source: NYC_taxi_data_Project.ipynb (Part2)

In this section, a series of business explorations were conducted through SparkSQL to get a comprehensive understanding of the taxi data and gain key business insights. It focused on various metrics of trip counts and total fare amount, trip distance, duration, examining their relationships with factors such as time, pick-up and drop-off location pairs, passenger counts. Finally, the goal of this section is to find the profitable duration bin that can be suggested to drivers to improve their incomes. The detailed analyses are presented below:

1. Explored number of trips, trip amount in terms of yeah, month, day of week, hour and passenger counts. Key insights from Fig 4:
    a) Average of amount per trip increased over time, which could be explained by inflation.
    b) The number of trips decreased over time, especially during the outbreak of the Covid-19 pandemic. The period from April, 2020 to August, 2020 saw the most loss of taxi trips.
    c) The peak month of taxi trips was December, and most trips occurred between 6 pm and 7 pm.

y_m	total_trips	dow	hr	avg_pcnt	avg_amt_trip	avg_amt_pas
2015-05	14441800	Sat	19	1.65	16.04	9.7
2016-12	11306924	Fri	19	1.62	15.76	9.71
2016-03	13341155	Thu	18	1.63	15.58	9.58
2015-02	13577029	Fri	19	1.64	15.01	9.15
2015-01	13758857	Sat	19	1.65	14.72	8.9
2016-11	10907093	Wed	19	1.61	16.16	10.02
2015-04	14252099	Thu	19	1.65	15.64	9.5
2015-12	12605758	Thu	19	1.65	15.84	9.62
2015-08	12218657	Sat	19	1.66	15.74	9.46
2015-11	12397698	Sun	19	1.64	15.93	9.72
2015-07	12644135	Wed	19	1.66	15.73	9.47
2015-09	12270625	Wed	19	1.65	16.04	9.74
2015-03	14592453	Sun	19	1.64	15.45	9.42
2017-09	9541067	Fri	19	1.61	16.3	10.14
2016-02	12471870	Sat	18	1.62	15.15	9.34

Fig 4. Screenshot of Business Exploration 1

2. Explored trip duration, distance and speed in terms of taxi color. Key insights from Fig 5:

    a) Two taxi types tented to have similar patterns on trip duration, distance, and speed.

    b) A general trip could be expected to occur in New York City, was a 14-minute trip, with a distance of 4.7 km and a speed of 19 km per hour.

taxi_color	avg_duration	med_duration	min_duration	max_duration
green	13.96	10.55	0.5	360
yellow	14.43	11.23	0.5	360

avg_distance	med_distance	min_distance	max_distance
4.63	3.04	0.32	96.56
4.77	2.74	0.32	96.56

avg_speed...	med_speed...	min_speed...	max_speed...
19.43	18.14	0.06	104.51
18.32	16.33	0.06	104.61

Fig 5. Screenshot of Business Exploration 2

3. Explored trip volume, duration, trip amount in terms of taxi color, pairs of pickup and drop-off locations, month, day of week and hour. Key insights from Fig 6 and Fig 7:

    a) The central of NYC, Manhattan had the most trips and the largest of amount of taxi fare.

    b) Long-distance taxi trips were most common in trips involving either EWR (Newark Airport, located outside of NYC) or Staten Island, a separate island within NYC (Fig). These trips tended to cover greater distances and had a higher fare amounts compared to other locations

taxi_color	PUBorough	DOBorough	mon	dow	hr	num_trips	avg_distance	avg_amt_trip	total_amt_paid
yellow	Brooklyn	Manhattan	1	Fri	8	3936	8.72	25.78	101450.68
yellow	Manhattan	Bronx	1	Fri	12	941	12.54	30.9	29080.23
yellow	Manhattan	Unknown	1	Sat	0	930	17.97	67.83	63083.23
yellow	Manhattan	EWR	1	Sat	8	492	27.97	89.63	44095.78
yellow	Brooklyn	Brooklyn	1	Sun	4	4901	3.76	12.41	60799.16
yellow	Bronx	Unknown	1	Sun	21	7	23.49	98.06	686.41
yellow	Manhattan	Queens	1	Mon	8	8612	19.07	47.44	408541.61
yellow	Brooklyn	Manhattan	1	Mon	10	800	8.88	25.52	20417.07
yellow	Bronx	Bronx	1	Mon	22	106	4.33	13.59	1440.45
yellow	Manhattan	Manhattan	1	Tue	3	25828	3.39	11.67	301365.59
yellow	Brooklyn	Brooklyn	1	Tue	17	2155	3.45	14.38	30978.68

Fig 6. Screenshot of Business Exploration 3

Fig 7. New York City Map

4. Exploring the distribution of tipped trips, and the average speed, average distance per dollar in terms of duration bins. Key insights from Fig8:

    a) More than 60% of the trips received non-cash tips, but only 11% of them were more than 5 dollars. Long duration trips tented to have tips more than 5 dollars.

    b) Long duration trips had higher average speed and average distance per dollar.

1.2 percent_tipped_trip
63.77

1.2 percent_tipped_al5_trip
11.86

duration_bin	1.2 avg_speed_kmh	1.2 avg_amt_kmd	1.2 percent_tipped_trip	1.2 percent_tipped_al5_trip
(0,5)	19.41	0.16	56.68	0.51
[5,10)	16.81	0.21	62.88	0.44
[10,20)	17.05	0.26	66.11	3.37
[20,30)	20.52	0.32	66.7	28.58
[30,60)	25.26	0.37	65.54	75.93
[60,+)	22.39	0.4	57.12	94.04

Fig 7. Screenshot of Business Exploration 4

5. Recommendation: Trip duration within 5-minute would be a profitable choice to maximize driver's income, as it had the highest hour rate (dollar per hour) base on average speed and average distance per dollar from previous analysis figures.

5. Machine Learning Pipelines

Corresponding code source: NYC_taxi_data_Project.ipynb (Part3)

After performing ETL, the data was prepared for further applications. In this section, two machine learning pipelines, a linear regression pipeline and a decision tree regression pipeline were built to predict the total amount of taxi trips based on latest two years data. These two algorithms were compared with baseline model in terms of performance. Additionally, processing time and model complexity were also considered to select a final model to evaluate its performance on unseen data.

5.1 Architecture

The architecture of this section is illustrated in Fig 8, and consists of three key components: Dataset Preparation, Machine Learning Algorithm Pipelining, and Models Evaluation.

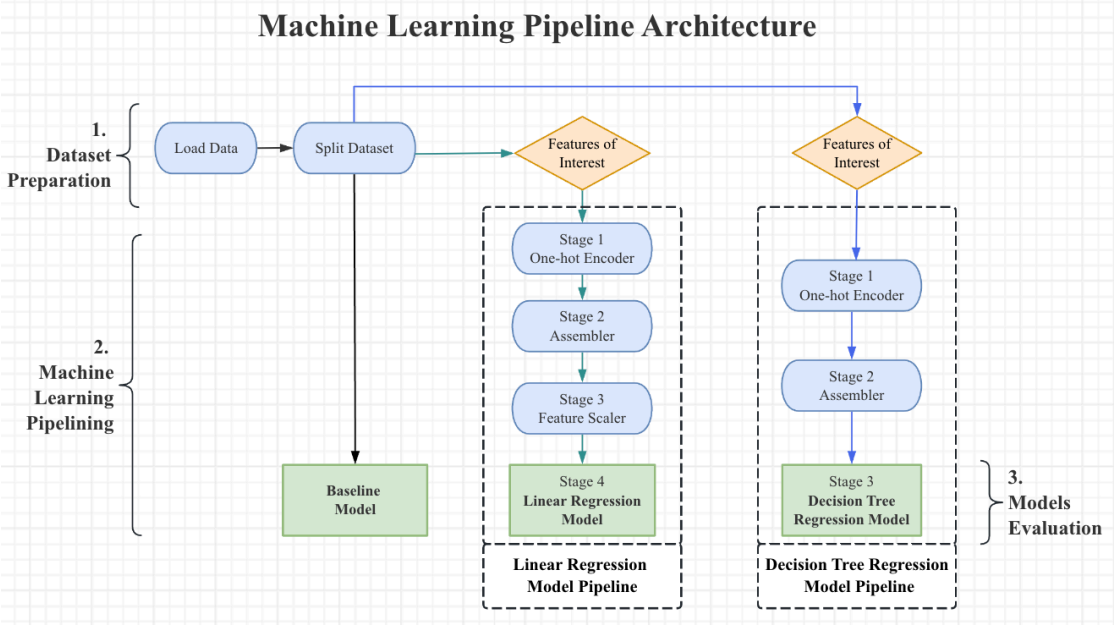


Fig 8. Machine Learning Pipeline Architecture

5.2 Dataset Preparation

1. **Read and Filtered Data**: The latest two years parquet file was loaded into a dataframe. To prevent noisiness in the dataset, records with non-positive total amounts were filtered out, ensuring that only valid fare amounts were considered..
2. **Split the Data**: The data was split into training, validation, and testing datasets based on specific time frames:
    a) **Testing Dataset**: The data from October, November, and December 2022 was selected as the testing dataset or unseen data for evaluating the final model.
    b) **Validation Dataset**: The data from July, August, and September 2020 was used as the validation dataset for parameters tunings and model selection.

c) **Training Dataset**: The remaining data, excluding the periods for testing and validation, was used to train the model.

5.3 Baseline Model

Simulated predictions by predicting all target variable as mean of the total amount from the training dataset. The performance, processing time and prediction time are shown by Fig 9. Noted that the training dataset captured older patterns and could be less representative, which lead to higher RMSE.

```
Baseline Model Processing time: 10.54s
train_rmse = 152.11
val_rmse = 63.83
```

Fig 9. Baseline Model Key Metrics

5.4 Linear Regression Model Pipeline

A serious of features of interest were selected and classified based on data types for linear model. New datafromes were created for the three datasets with selected variables. Then a linear regression model pipeline was built as follows:

1. **One-Hot Encoder:** Applied StringIndexer to convert categorical variables into numerical indices, followed by a OneHotEncoder to transform them into dummy variables.
2. **Assembler:** Combining all features into a single feature vector using the VectorAssembler.
3. **Scaler:** The feature vector was scaled using StandardScaler for the linear model.
4. **Linear Regressor**: Instantiated linear regression model by specifying scaled feature vector and label column, and reduced the maxIter to 10 to avoid over-computing.

After defining linear model pipeline stages, the pipeline was fitted using training dataset. Then the trained pipeline was applied to the validation dataset to generate predictions, along with its RMSE.

5.5 Decision Tree Regression Model Pipeline

A serious of features of interest were selected and classified based on data types for decision tree model. New datafromes were created for the three datasets with selected variables. Then a decision tree regression model pipeline was built as follows:

1. **One-Hot Encoder:** Applied StringIndexer to convert categorical variables into numerical indices, followed by a OneHotEncoder to transform them into dummy variables.
2. **Assembler:** Combining all features into a single feature vector using the VectorAssembler.
3. **Decision Tree Regressor**: Instantiated decision tree regression model by specifying scaled feature vector and label column and set a proper random seed.

After defining pipeline stages, the pipeline was fitted using training dataset. Then the trained pipeline was applied to the validation dataset to generate predictions, along with its RMSE.

5.6 Models Evaluation

1. **Model Selection:** Models were evaluated based on multiple metrics, such as model performance (RMSE), model complexity (the number of features), and processing time. As shown in Fig 10, linear model achieved the best RMSE performance on both training dataset and validation dataset. Though it considered more features in modelling, it beat the decision tree regression model significantly regarding processing time. Therefore, the linear model was selected for further evaluation on unseen testing dataset.

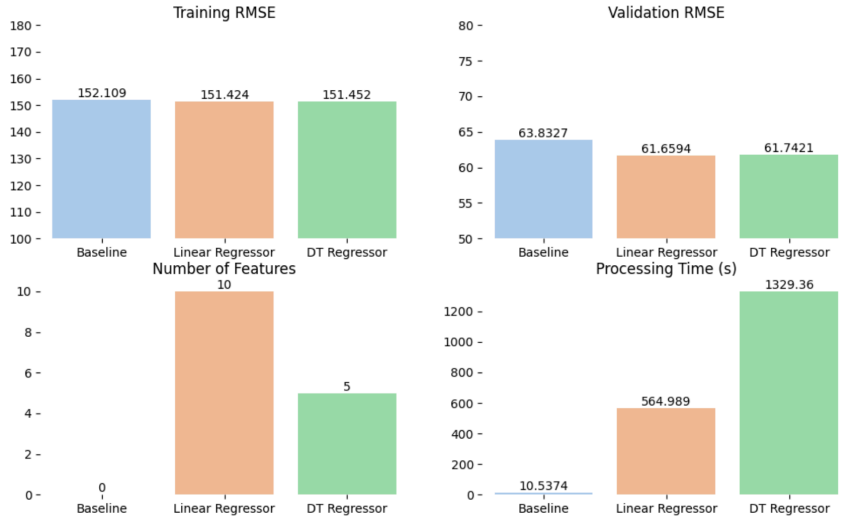


Fig 10. Models Evaluation for Model Selection

2. **Selected Model Performance on Unseen Testing Dataset:** The RMSEs for both baseline model and the selected linear regression model were computed (Fig 11). The results indicated that the linear regression model improved accuracy significantly, with a 73.39% reduction of RMSE compared to the baseline model, showing the strong ability of selected model to generalize and capture patterns on unseen data.

```
Baseline Model Testing RMSE = 17.92
Linear Model Testing RMSE = 4.77
Reduction of RMSE = 73.39%
```

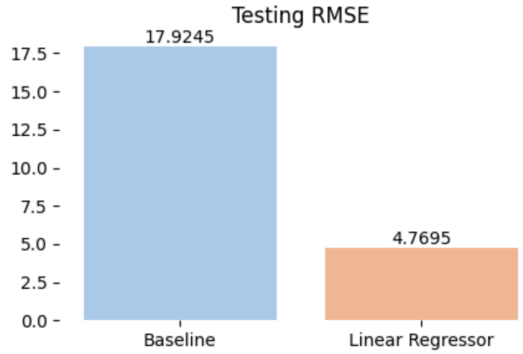


Fig 11. RMSE Comparison Between Baseline and Linear Regression Models

6. Conclusion

This project has illustrated how Parquet files optimize Big Data storage by making it memory-efficient, and how data can be stored in cloud storage platform like Azure, then mounted and transferred to a data processing platform Databricks. Within Databricks, data engineers can perform data Extraction, Transformation and Loading (ETL) processes to create well-structure datasets, ready for client-side business analysis. More importantly, building Machine Learning pipelines to support advanced predictive analytics. This approach offers an efficient, flexible and scalable for data processing, including but not limit to data extraction, loading, cleaning, and transformation. It also showcases Databricks strong capabilities in facilitating both data analysis and machine learning pipelining tasks.

7. Appendix

1. Assignment brief and source files used in this project are part of the learning materials from Big Data Engineering of University of Technology Sydney (UTS).
2. Relevant code script: NYC_taxi_data_Project.ipynb
3. Data Dictionary after ETL:

Filed name	Data type	Description
VendorID	byte	A code indicating the LPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc
pickup_datetime	timestamp	The date and time when the meter was engaged.
dropoff_datetime	timestamp	The date and time when the meter was disengaged.
passenger_count	double	The number of passengers in the vehicle. This is a driver-entered value.
distance_km	double	The elapsed trip distance in miles reported by the taximeter.
RatecodeID	byte	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	string	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. Y= store and forward trip N= not a store and forward trip
payment_type	byte	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	double	The time-and-distance fare calculated by the meter.
extra	double	Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
mta_tax	double	$0.50 MTA tax that is automatically triggered based on the metered rate in use.
tip_amount	double	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
tolls_amount	double	Total amount of all tolls paid in trip.
improvement_surcharge	double	$0.30 improvement surcharge assessed on hailed trips at the flag drop. The improvement surcharge began being levied in 2015.
total_amount	double	The total amount charged to passengers. Does not include cash tips.
congestion_surcharge	double	Total amount collected in trip for NYS congestion surcharge.
airport_fee	double	$1.25 for pick up only at LaGuardia and John F. Kennedy Airports.
speed_kmh	double	Trip speed calculated based on duration and distance.
duration_min	double	Trip duration when the meter was engaged and disengaged.
positive_amt	byte	This flag indicates whether the trip record has a positive total amount. 1 = Positive total amount 0 = Non-positive total amount
taxi_color	string	Yellow: Medallion Taxis, allowed to pick up passengers in all boroughs. Green: Street-Hail Liveries, allowed to pick up passengers in limited areas.
ehail_fee	double	Electronic hailing fee.
trip_type	byte	A code indicating whether the trip was a street-hail or a dispatch that is automatically assigned based on the metered rate in use but can be altered by the driver. 1= Street-hail 2= Dispatch
PU_year	short	The year when the taximeter was engaged.
PU_month	byte	The month when the taximeter was engaged.
PUBorough	string	Taxi Borough in which the taximeter was engaged.
PUZone	string	Taxi Zone in which the taximeter was engaged.
PU_service_zone	string	Taxi Service Zone in which the taximeter was engaged.
DOBorough	string	Taxi Borough in which the taximeter was disengaged.
DOZone	string	Taxi Zone in which the taximeter was disengaged.
DO_service_zone	string	Taxi Service Zone in which the taximeter was disengaged.