Master Thesis Report

Master ADEO 2

Title:

Dynamic Price Optimization Using Cloud Environment:

A Novel Approach for Cab Fare Pricing

Work By

Azza Kamoun

Supervisor

Prof.Madan Radhakrishnan

Table of Contents

Abstract:

This study aims to estimate the cost of taxi rides using machine learning techniques. The methodology entails data gathering, preprocessing, and modeling with the surge multiplier effect and weather conditions taken into account in the regression. A new static price dataset was constructed by combining two existing datasets. By estimating the number of trips per hour for each day and adding surge multipliers based on quartiles of the demand for taxis, the surge multiplier impact was captured. The data was then preprocessed using feature engineering, data cleansing, and data transformation. The results showed that the random forest model performed better than the other models (Decision Trees, Monte Carlo, LSTM) across all evaluation metrics when four machine learning models were applied.
The models were deployed in both Google Cloud Platform and Amazon Web Services, and a real-time simulation application was developed to showcase the successful implementation of the models.

Keywords: machine learning, taxi ride prices, surge multiplier, weather conditions, data preprocessing, random forest, Decision Trees, Monte Cartlo , LSTM (Long-Short-Term-Memory), model evaluation, cloud deployment, GCP (Google Cloud Platform), AWS (Amazon Web Serices), Streamlit, real time simulation.

I. Introduction

1. Context

Ride-hailing businesses like Uber, Lyft.. have seen substantial development and have had a tremendous influence on urban mobility. As a result, the technology and business model behind these platforms have become attractive topics of study in different disciplines from economics to computer science and operational research. For instance, one of the most critical components of the service that has been an important focus point for recent research is dynamic pricing. Dynamic price optimization is a strategy that involves adjusting prices in real-time based on various factors that are determined by the business. This approach has been widely adopted in various industries to optimize revenue and improve the overall customer experience. As for its current application for ride-hailing services, the main goal behind it is to optimize revenue for the company by ensuring that prices are high enough to incentivize drivers to work during periods of high demand, while also being low enough to attract riders during periods of low demand. The companies use advanced techniques such as machine learning in order to dynamically optimize their prices to match the supply and demand. They use what is known as a "surge multiplier" where fares are higher when the demand is higher than the available cars. The optimal surge multiplier is usually set between 1 for non-peak hours and 1.5 for peak hours, as explained by Dr.Dawn Woodard, a Senior data science Manager of Maps Uber in a conference held by Microsoft Research in 2018 [1] + **Figures 1&2**. Additionally, as the demand tends to substantially increase during New Year's Eve, Halloween public events, or on Weekends, the multiplier tends to increase (i.e.1.8x or 2.5x) per ride according to a study published at AltexSoft tech blog [2].

It is true that this technological advancement has reshaped the dynamics of the market, however Brian M. Rosenthal's article "New York Is Urged to Consider Surge Pricing for Taxis" identified a problem with the taxi business in New York City [12]. According to the report, the taxi sector has been declining owing to competition from ride-hailing services due to the flexible services they have been providing to the customers.

As a result, many licensed taxi drivers have been struggling to make ends meet, and taxi medallions, which are necessary to operate a cab in the city, became less

desirable. To solve this issue, several politicians and experts advocated for the implementation of surge pricing for taxis in New York City, allowing drivers to charge higher prices during peak demand periods, in order to incentivize the formal license-based sector and not make the actual taxi drivers worse off. The issue is then, how to combine the need to assist taxi drivers with the need to provide inexpensive and accessible transit choices for all New York City citizens. Despite some criticism, the notion of surge pricing for taxis in New York City continues to be a point of contention, but, no practical changes to impose surge pricing for yellow cabs have been adopted in New York City yet.
For this reason, the main research question of this thesis will be:

- How can we integrate the idea of surge pricing in the fare price of taxi drivers?
- How to provide a transparent way to estimate the dynamic ride price even if the customer is not opting for a new ride-hailing service?

The ultimate goal of the research is to provide potential solutions to enhance the performance of the yellow cabs industry. The pain points that will be tackled will be:

2. On the drivers' side:
    a. Problem: lack of incentives & lack of profit with the usual pricing used.
    b. Solution: Dynamic pricing that incorporates the surge multiplier in order to earn more money at high demand hours, get encouraged to work more, and have a more effective use of the taxi fleet.
3. On the customers' side:
    a. Problem: Lack of transparency of the fare amounts, and non-existence of the latest technological advances in the sector (for a certain segment).
    b. Solution: An app that would allow users to estimate the price of a ride, and potentially compare that to the other solutions' prices. The solution could also provide a new feature that is nonexistent: booking a cab for later, which can serve an important use case especially for businesses.

In conclusion, this research will give a comprehensive understanding of the use of cloud services for dynamic price optimization in the context of ride-hailing

services, develop a model that adjusts the prices to surge multipliers, and to put in place an idea of an app that can answer the needs of the customers in full transparency.

4. State of the art

Dynamic pricing, also known as surge pricing or time-based pricing, is a frequently used method to maximize income and enhance overall customer experience in a variety of businesses,such as e-commerce, transportation, and hospitality. A substantial amount of work on dynamic pricing has been produced in the previous years.The goal of these researches is generally to create a realistic model for different markets and then apply machine learning techniques to it.

The use of sophisticated techniques such as machine learning, Monte Carlo models, Markov models, and deep learning to dynamically modify pricing depending on supply and demand is a fundamental component of this strategy. For instance, the **Monte Carlo simulation** is one of the most extensively utilized dynamic pricing approaches. This approach entails creating a large number of random samples in order to assess the system's probability distribution, and then utilizing this knowledge to optimize the pricing strategy. A research published in the Journal of Revenue and Pricing Management (Castillo, Juan Camilo, 2017) [3] employed Monte Carlo simulation to optimize a ride-hailing platform's pricing approach. According to the study, the Monte Carlo simulation-based technique increased revenue by an average of 15% when compared to the old pricing strategy. Machine learning is another technique that has also been widely used for dynamic pricing to assess enormous volumes of data and forecast future demand. This data may then be utilized to improve the pricing approach. Neural networks and decision trees are two examples of machine learning techniques that could be utilized for dynamic pricing.

On the one hand, **neural networks** is a machine learning technique that is modeled after the human brain and used to evaluate large amounts of data. In a research paper by S. Shakya, M. Kern, G. Owusu, C.M. Chin, entitled Neural network demand models and evolutionary optimisers for dynamic pricing [4], the authors used NN to estimate the demand for dynamic pricing, this method may be useful in the detection of patterns and trends in the data and forecast future demand. Unlike typical demand models, they make no assumptions regarding the link between

various components. They instead discover these associations from the data in order to represent relationships that are too complex for people or computer systems to discover. Because of this capability, neural networks are an excellent choice for modeling demand in dynamic pricing. In their paper, they also suggested employing evolutionary algorithms (EA) to optimize dynamic pricing challenges.
Decision trees, on the other hand, is an algorithm that predicts data by splitting it down into smaller and smaller sections in the form of nodes and leaves. This can help companies figure out which variables are the most influential on prices and which of these price ranges predict the highest sales. One of the most prominent examples of the application of using neural networks and decision trees for dynamic pricing is Uber. According to a blog post published by Uber in June 2017, this strategy increased income by up to 20% [5]. Another important model that has been widely used for surge pricing is the **Markov Model.** It is based on the Markov property, which claims that a system's future state is determined only by its current state and not the previous ones. This property may be considered as well suited to representing dynamic systems with temporal dependencies, which is the case of dynamic pricing. For instance, in the paper "Dynamic Pricing Competition with Unobservable Inventory Levels: A Hidden Markov Model Approach" [6], the authors propose the use of a hidden Markov model to address the problem of dynamic pricing competition with unobservable inventory levels. The model captures the dynamic interactions between companies in a competitive market, in which each firm's price decisions are impacted by its inventory level, which competitors cannot directly observe. The parameters are estimated using Bayesian learning.

On another note, it is important to note that dynamic pricing has also been prominently implemented in other sectors as well.
For instance, in recent years, **the retail industry** has increasingly used real-time pricing to optimize revenues while improving the consumer experience. This method entails altering prices based on variables such as demand, time, inventory, and competition prices. One of the primary benefits of dynamic pricing in retail is that it enables firms to adjust to changing market conditions, such as fluctuations in demand or changes in competition price, resulting in increased profitability. The increasing use of this strategy in retail has even incentivized Google to offer a

collection of cloud-based machine learning models for price optimization geared exclusively for that industry through Google Cloud Platform[7].
Moreover, **In the hospitality business,** such as hotels and resorts, dynamic pricing is utilized to optimize income and give more tailored price alternatives to visitors. This method involves altering accommodation prices based on demand, time, and availability. Prices, for example, may be higher during peak travel hours and lower during off-peak travel times. Because firms may better adapt to changing market conditions and give guests with more specialized price alternatives, dynamic pricing in the hospitality industry can result in increased profitability.

One case in point is the publication "Dynamic Pricing Strategies in the Hospitality Industry: An Empirical Study" (Rojas, 2018) [8], which defines "open pricing" as a dynamic pricing solution that employs big data-led revenue management systems (RMS) to establish prices in real-time without pre-set price ranges and barriers. According to the article, the primary benefit of open pricing is that it allows for more price adaptability to both demand and last room value, resulting in more value-based pricing that is closer to customers' willingness to pay. Finally, it concluded that real-time price setting, individual optimization per day and room type, increased dynamism in price and discounts, dynamic pricing across tariffs and distribution channels, and improved hotel positioning on online travel agencies are the major differences between traditional price discrimination and open pricing.

However, only a few publications and research works were conducted to predict the prices of the yellow taxis in a dynamic way. One of the most important articles in the subject is Shylaja S and Kannika Nirai Vaani M's article "A Machine Learning Framework for Profitability Profiling and Dynamic Price Prediction for New York City Taxi Trips" [13] . The latter presents a study of the taxi industry in New York City and proposes a machine learning framework for profitability profiling and dynamic price prediction. The authors employed a combination of machine learning approaches, such as regression analysis and decision trees, as well as data from the New York City Taxi dataset, to do this. They began with exploratory data analysis to better understand the patterns and trends in the data. They then implemented regression analysis to forecast the profitability of individual journeys based on variables such as time of day, pickup and drop-off locations, and weather. They also used decision trees to investigate the links

between the attributes and the goal variable (profitability) and to determine the most important predictors of profitability. The final results proposed is a dynamic pricing algorithm that could alter the fare in real-time depending on the projected profitability of each journey.

As for the deployment and implementation of the different models, there is an increasing use of cloud services for dynamic pricing optimization. . Cloud services, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), provide businesses with the ability to store and process large amounts of data in the cloud. This enables businesses to easily scale their resources, as well as access advanced machine learning and artificial intelligence (AI) tools, such as TensorFlow and PyTorch. The use of cloud services is starting to get adopted for dynamic pricing. As mentioned above, Google Cloud Platform (Google Cloud, Price Optimization using Vertex AI Forecast, 2021) has even provided a set of cloud-based machine learning models for price optimization [7]. Moreover, Total Data Science utilized AWS to build a machine learning model for dynamic price optimization in the retail business (Total Data Science, How Machine Learning is Helping in Providing Dynamic Pricing, 2020) [9],. According to the study, using cloud services enabled faster and more effective model training, resulting in smarter pricing strategies and more income.

II. Methodology

1.1 Model Selection

The first step of the project was collecting the data, preprocessing it and modeling it. This will set the ground for the deployment of the best model in later stages. In general, this phase can be summarized as follows:

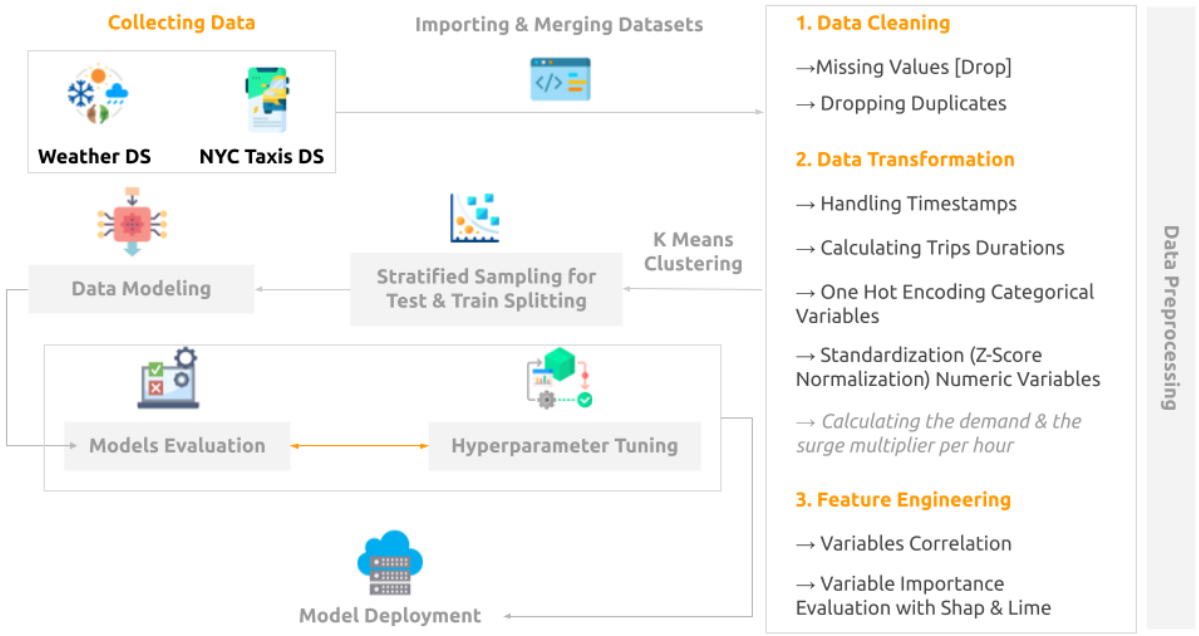


Figure 3: Model Selection Process [Generalized]

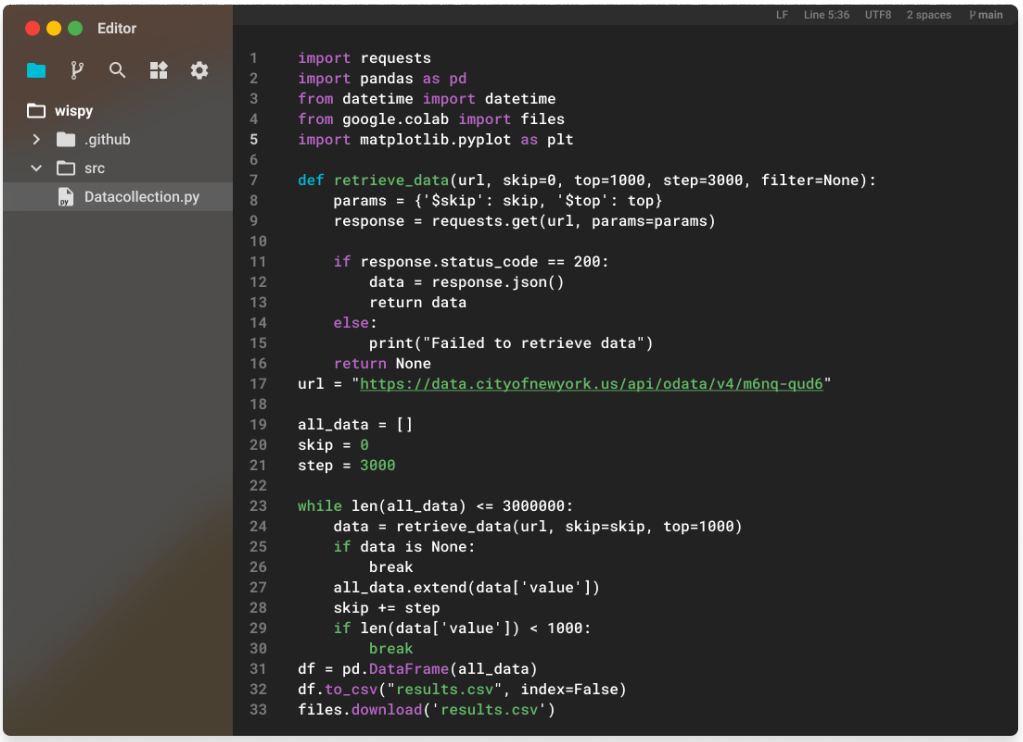
Since we are trying to introduce the idea of "surge multiplier" in the price estimation of the rides, we will be running two different models. In order to make our work more reliable, we will be looking at the same dataset collected, yet in two very different ways, as we will explain in **section 2** .

Section 1: Data Collection

- **Rides Dataset Collection:**

  The dataset "2021 Yellow Cab Trip Data" provides information on cab rides in New York City [10], collected, distributed and released by New York City's Taxi and Limousine Commission (TLC). It covers trip records generated by yellow cab Technology Service Providers' inputs (TSPs). Each record represents a single cab journey made in 2021 and contains information such as pickup and dropoff dates and times, pickup and dropoff locations, trip lengths, rates, rate kinds, payment types, and passenger counts. The dataset has 30.9 million rows and 18 columns, including VendorID, tpep pickup

datetime, tpep drop off 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, and congestion surcharge. The good thing about this dataset is that it is always updated. Indeed, it was most recently updated on April 29, 2022, and the metadata on May 10, 2022, and it offers important insights about the city's transportation patterns and trends, which may be used for transportation planning and research. In order to use the data from this dataset, we will collect from an API provided by the City of New York and filter using a skip and top parameter. The data will then be stored in a Pandas DataFrame and saved as a CSV file for further analysis, all done in python. The data will then be collected in chunks of 1000 rows at a time with a step size of 3000 and was repeated until a total of 3 million rows were collected. The resulting DataFrame was then saved as a CSV file for further analysis.



- **Weather Collection:**

  Visual Crossing, a supplier of meteorological data services, has gathered weather data for years and forecasted for months [11]. Hence, the dataset of historical weather information from 2021 is available for download in grid

format on their website and was gathered from a number of sources, including surface measurements, weather satellites, and numerical weather models. This site gives information on meteorological parameters such as temperature, precipitation, wind speed, and pressure. To guarantee accuracy and quality, the data was then processed, analyzed, and confirmed. The processed data is then made accessible in grid format for download, allowing users to quickly access the data for their individual needs. Visual Crossing's data gathering technique assures that the data is trustworthy and up to date, making it a valuable resource for individuals and groups. Hence, the weather dataset for 2021 was downloaded there in csv format.
The final result, the weather data for New York City for 2021 contains information on the data's date and time, as well as meteorological measurements including temperature, humidity, precipitation, wind, and cloud cover. Dew point, perceived temperature, UV index, solar radiation, and air pressure are also included. The dataset also includes information about dawn and sunset times, moon phase, and meteorological conditions such as snow, wind gusts, and precipitation type. A description and an emblem depicting the current weather conditions are also included in the data.

Section 2: Data Preparation

- **<u>New Static Price Dataset</u>**

  The first goal of our project is to predict the static/usual price of a fare, without any additional multipliers. Hence, we started by merging the two datasets that were integrated into a single one called “Static Price” using the "datetime" column as a connecting factor.

connecting factor

```
df_cabs.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 81003 entries, 0 to 81002
Data columns (total 52 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   __id                    81003 non-null  object
 1   vendorid                81002 non-null  float64
 2   datetime                81002 non-null  object
 3   tpep_dropoff_datetime   81002 non-null  object
 4   passenger_count         81002 non-null  float64
 5   trip_distance           81002 non-null  float64
 6   ratecodeid              81002 non-null  float64
 7   store_and_fwd_flag      81002 non-null  object
 8   pulocationid            81002 non-null  float64
 9   dolocationid            81002 non-null  float64
 10  payment_type            81002 non-null  float64
 11  fare_amount             81002 non-null  float64
 12  extra                   81002 non-null  float64
 13  mta_tax                 81002 non-null  float64
 14  tip_amount              81002 non-null  float64
 15  tolls_amount            81002 non-null  float64
 16  improvement_surcharge   81002 non-null  float64
 17  total_amount            81002 non-null  float64
 18  congestion_surcharge    81002 non-null  float64
 19  month                   81002 non-null  float64
 20  name                    80992 non-null  object
```

NYC taxi DS

```
 21  tempmax                 80992 non-null  float64
 22  tempmin                 80992 non-null  float64
 23  temp                    80992 non-null  float64
 24  feelslikemax            80992 non-null  float64
 25  feelslikemin            80992 non-null  float64
 26  feelslike               80992 non-null  float64
 27  dew                     80992 non-null  float64
 28  humidity                80992 non-null  float64
 29  precip                  80992 non-null  float64
 30  precipprob              80992 non-null  float64
 31  precipcover             80992 non-null  float64
 32  preciptype              66819 non-null  object
 33  snow                    80992 non-null  float64
 34  snowdepth               80992 non-null  float64
 35  windgust                66822 non-null  float64
 36  windspeed               80992 non-null  float64
 37  winddir                 80992 non-null  float64
 38  sealevelpressure        80992 non-null  float64
 39  cloudcover              80992 non-null  float64
 40  visibility              80992 non-null  float64
 41  solarradiation          80992 non-null  float64
 42  solarenergy             80992 non-null  float64
 43  uvindex                 80992 non-null  float64
 44  severerisk              0 non-null      float64
 45  sunrise                 80992 non-null  object
 46  sunset                  80992 non-null  object
 47  moonphase               80992 non-null  float64
 48  conditions              80992 non-null  object
 49  description             80992 non-null  object
```

Weather DS

The merged dataset hence captures all the weather information related to each taxi trip taken in 2021.

- **New Surge Multiplier Dataset**

  As explained in the introduction and state of art, ride hailing companies like Uber and Lyft, choose their own model for determining surge multipliers. Generally, it tends to capture the supply-demand dynamic. In our case, the parameters are not the same as we do not have neither control nor visibility over the number of taxis available. Moreover, our work aims not only to capture the demand, but also take into account other factors like weather conditions, day of the week/ month.. In order to have relevant data, we chose to work with the same “New Static Price Dataset”, yet, with some major modifications.

  1) **Calculating the demand per day and hour:**

     Before working on the surge multiplier, we started by calculating the number of trips per hour for each day in the dataset.

     To have a consistent dataset, the rest of the columns were averaged (passenger counts, windspeed, duration…) .

```
merged_df = pd.merge(Average_per_day_hour, Demand_per_day_hour, on=['Day', 'Month','Hour'])
```

```
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3261 entries, 0 to 3260
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Day                   3261 non-null   int64  
 1   Month                 3261 non-null   int64  
 2   Hour                  3261 non-null   int64  
 3   passenger_count       3261 non-null   float64
 4   trip_distance         3261 non-null   float64
 5   fare_amount           3261 non-null   float64
 6   extra                 3261 non-null   float64
 7   improvement_surcharge 3261 non-null   float64
 8   total_amount          3261 non-null   float64
 9   congestion_surcharge  3261 non-null   float64
 10  temp                  3261 non-null   float64
 11  feelslike             3261 non-null   float64
 12  humidity              3261 non-null   float64
 13  snow                  3261 non-null   float64
 14  windspeed             3261 non-null   float64
 15  winddir               3261 non-null   float64
 16  cloudcover            3261 non-null   float64
 17  duration              3261 non-null   float64
 18  num_trips             3261 non-null   int64  
dtypes: float64(15), int64(4)
memory usage: 509.5 KB
```

  2) **Adding the surge multiplier column:**

     One of the solutions that were thought of for our project was to make the surge multiplier directly correlated with the demand level. Generally, surge multiplier ranges between 1 and 2. Hence, in our case, the surge multiplier was calculated based on the quartiles of the "num_trips" column which represents the demand for taxis. The first

step was to use the pd.qcut() method to divide the "num_trips" values into 5 equally-sized groups or quartiles, based on their distribution in the dataset. The q=[0, 0.25, 0.5, 0.75, 0.9, 1] parameter indicates the percentile cutoffs for the quartiles, and the labels=[1, 1.25, 1.5, 1.75, 2] parameter assigns a corresponding value to each quartile range. Then, each quartile range was assigned a surge multiplier value. For example, the lowest quartile range (1) was assigned a value of 1, the second-lowest range (1.25) was assigned a value of 1.25, and so on, up to the highest range (2), which was assigned a value of 2.

```
merged_df['num_trips'].describe()

count    3261.000000
mean      873.813247
std       601.679385
min         1.000000
25%       285.000000
50%       975.000000
75%      1265.000000
max      2502.000000
Name: num_trips, dtype: float64
```

```
merged_df['surge_multiplier'].describe()

count    3261.000000
mean        1.399647
std         0.319951
min         1.000000
25%         1.000000
50%         1.250000
75%         1.500000
max         2.000000
Name: surge_multiplier, dtype: float64
```

- **<u>Unified Process: Data Preprocessing</u>**

  The methodology for preprocessing the data can be broken down into three main steps: Data Cleaning, Data Transformation, and Feature Engineering.

1. **Data Cleaning:**

   The first step involves cleaning the data to remove any errors or inconsistencies. This includes handling missing values and dropping duplicate rows. For missing values, the approach is to drop the entire row since there is not enough information to impute the missing values. For duplicates, the approach is to drop all but one instance of each duplicate.

2. **Data Transformation:**

   The next step is to transform the data to a format suitable for analysis. This includes handling timestamps, calculating trip durations, one-hot encoding categorical variables, standardizing numeric variables, and calculating the demand and surge multiplier per hour. Handling timestamps involves converting timestamp strings to a datetime format, and extracting the relevant time features such as the hour of the day. Trip durations can be calculated by subtracting the start time from the end time. One-hot encoding categorical variables involves converting categorical variables into

numerical dummy variables. Standardization involves scaling the numeric variables to a standard normal distribution

3. **Feature Engineering:**

   The final step is to perform feature engineering to create new variables that capture relevant patterns and relationships in the data. This includes evaluating variable correlation and assessing variable importance with Shap & Lime.

   Evaluating variable correlation involves identifying variables that are highly correlated with each other and may cause multicollinearity in the analysis. Assessing variable importance involves using machine learning models to determine the importance of each variable in predicting the target variable. Additionally, the use of Shap & Lime allows for a more interpretable understanding of how individual variables contribute to the model's predictions.

   Overall, this methodology for preprocessing the data aims to ensure that the data is clean, transformed, and engineered in a way that is suitable for analysis and allows for meaningful insights to be drawn.

Section 3: Data Modelling

1. **Data Splitting (Train & Test Sets)**

   In order to decide on the sampling technique to be used for splitting the dataset into train and test sets, we tested whether of not it can be clustered. This is why, the k-means clustering algorithm is applied to the input data with different numbers of clusters (k values) ranging from 1 to a specified maximum value (k_max). For each k value, the algorithm computes the within-cluster sum of squared distances (WCSS) and stores it in a list. The optimal number of clusters is determined by finding the "elbow" point in the WCSS plot. The elbow is defined as the k value that gives the maximum reduction in WCSS while adding one more cluster, which is equivalent to finding the minimum absolute difference between consecutive values in the WCSS plot. This is done using the best_k function. The k-means clustering algorithm is then applied again to the input data with the optimal number of clusters, and the resulting cluster centers, cluster values, and cluster counts are stored.

The results of the above showed that the dataset can be clustered into 6 clusters for new pricing dataset and 8 for new surge dataset, and that the train and test sets will be divided using the stratified sampling technique.

2. **Data Modeling**

   We used four different machine learning models for data modeling and prediction: decision tree, random forest, LSTM, and Monte Carlo simulation(with decision tree, random forest and LSTM).

   The decision tree model is implemented using the DecisionTreeRegressor class from the sklearn.tree module. The decision_tree function takes the training data (X_train and y_train) and an optional parameter max_depth that controls the maximum depth of the decision tree. The function returns the trained decision tree model.

   The random forest model is implemented using the RandomForestRegressor class from the sklearn.ensemble module. The random_forest function takes the training data (X_train and y_train) and two optional parameters: max_depth that controls the maximum depth of the decision trees in the forest and n_estimators that controls the number of decision trees in the forest. The function returns the trained random forest model.

   The LSTM model is implemented using the Sequential class from the tensorflow.keras module. The create_lstm_model function takes the training data (X_train and y_train) and the test data (X_test and y_test). The function defines an LSTM layer with 20 units followed by a dense layer with one unit and linear activation. The model is compiled with the adam optimizer, mean_squared_error loss function, and mae metric. The function also defines an early stopping callback that monitors the validation loss and stops training if it doesn't improve for 5 epochs. The model is trained for 50 epochs using a batch size of 32. The function returns the trained LSTM model and the training history.

   The Monte Carlo simulation model is implemented using the DecisionTreeRegressor and RandomForestRegressor classes from the sklearn.tree and sklearn.ensemble modules, respectively. The monte_carlo_simulation function takes the training data (X_train and y_train), the test data (X_test and y_test), the model type (forest, dt, or

lstm), and an optional parameter LSTM that is only used when the model type is lstm. The function selects the appropriate model based on the model type and fits it on the training data. Then, it predicts the target values on the test data and computes the sample mean and confidence bounds. The function returns the predicted target values, the sample mean, and the confidence bounds.

Hence, there were three calls to the monte_carlo_simulation function that use the trained models to perform a Monte Carlo simulation and compute the sample mean and confidence bounds for the predicted target values. The first call uses the random forest model, the second call uses the decision tree model, and the third call uses the LSTM model.

3. **Data Evaluation**

   The model evaluation methodology includes four main metrics to evaluate the performance of a given model:

   - Mean Squared Error (MSE): This metric measures the average squared difference between the predicted values and the true values. A lower MSE indicates that the model is better at predicting the target variable.
   - Mean Absolute Error (MAE): This metric measures the average absolute difference between the predicted values and the true values. A lower MAE also indicates that the model is better at predicting the target variable.
   - R-squared (R2) score: This metric measures the proportion of the variance in the target variable that is explained by the model. It ranges between 0 and 1, with a higher value indicating a better fit between the predicted values and the true values.
   - Cross-validation: This technique involves splitting the available data into several partitions or folds, using some of them to train the model and the remaining ones to test its performance. This helps assess the model's generalization ability by estimating how well it would perform on new, unseen data.

   It is important to note that manual models fine-tuning was performed until getting the best of each one.

Section 4: Results

1. **“Static Price Dataset”**

→ Feature Engineering:

For feature selection, we used different techniques such as correlation analysis, LIME (Local Interpretable Model-Agnostic Explanations), and SHAP (SHapley Additive exPlanations).

1.1- Correlation analysis:

We calculated the correlation between each feature and the target variable ('total_amount') and sorted them in descending order. The result shows that 'fare_amount', 'trip_distance', 'improvement_surcharge', 'congestion_surcharge', 'duration', 'pulocationid', 'Hour', and 'dolocationid' have the highest absolute correlation with the target variable.

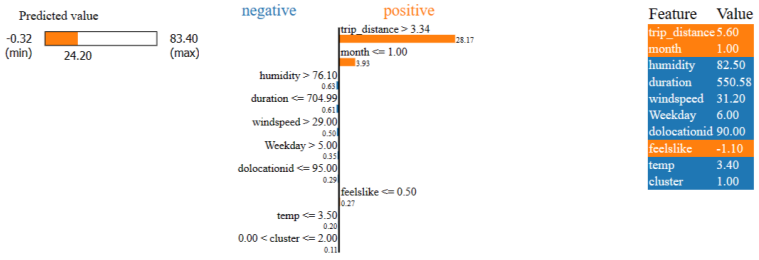
```
CorrelatinoRes = df_cabs.corr().abs()['total_amount'].sort_values(ascending=False)
```

```
CorrelatinoRes
```

```
total_amount             1.000000
fare_amount              0.976664
trip_distance            0.901369
improvement_surcharge    0.168341
congestion_surcharge     0.116466
duration                 0.082067
pulocationid             0.075935
Hour                     0.067078
dolocationid             0.040840
passenger_count          0.017018
extra                    0.014667
vendorid                 0.014171
Weekday                  0.010144
Day                      0.009795
Month                    0.007256
month                    0.007256
windspeed                0.006563
humidity                 0.006198
cloudcover               0.005888
feelslikemin             0.004398
feelslike                0.003822
winddir                  0.002538
temp                     0.002309
snow                     0.001213
Year                          NaN
Name: total_amount, dtype: float64
```

1.2- LIME:

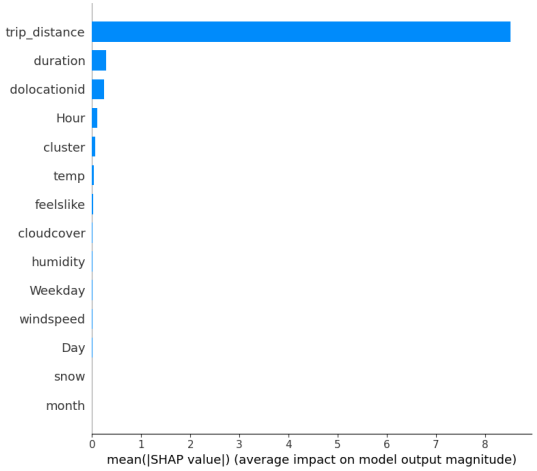
We created a LimeTabularExplainer object and used it to explain the predictions of a random row of data. The result shows the intercept, the predicted value for the target variable, and the contribution of each feature to the prediction.



The top two features that contribute to this prediction are "trip_distance" and "month," with values of 5.60 and 1.00, respectively. This suggests that the longer the

trip distance and the earlier in the year the ride takes place, the higher the predicted total amount will be. Other important features include "humidity," "duration," and "windspeed."

1.3- SHAP:



The code uses SHAP to calculate the Shapley values for each feature and plot a summary bar chart of their importance. The top three features according to Shapley values are "trip_distance," "duration," and "dolocationid." The highest positive Shapley value is for "trip_distance," indicating that as the trip distance increases, the predicted total amount increases as well. The highest negative Shapley value is for "duration," indicating that as the duration of the ride increases, the predicted total amount decreases

Overall, the three techniques seem to suggest that 'trip_distance', 'duration', and 'dolocationid' are the most important features in predicting the 'total_amount' variable, while the other variables have a lower impact.

- The variables that will be taken into account in our regression are: trip_distance, dolocationid, total_amount, month, temp, feelslike , snow, windspeed, cloudcover, Day, Hour, Weekday and duration.

→ Modeling Results:

Below, we will be comparing the performance of four different machine learning models: Decision Tree, Random Forest, LSTM, and Monte Carlo, using Mean Squared Error (MSE), Mean Absolute Error (MAE), R Squared, and Average Cross Validation Score.

				Monte Carlo +		
	Decision Tree	**Random Forest**	LSTM	Decision Tree	**Random Forest**	LSTM
Mean Squared Error	42.3	**22.28**	28.90	41.54	**23.48**	27.67
Mean Absolute Error	3.24	**2.28**	2.34	3.22	**2.31**	2.35
R Squared	70.77%	**84.60%**	80.03%	71.3%	**83.77%**	80.88%
Average Cross Validation Score	42.92%	**70.86%**				

The results show that Random Forest outperformed the other models in terms of MSE, MAE, R Squared, and Average Cross Validation Score. It had the lowest MSE and MAE, and the highest R Squared and Average Cross Validation Score. This is also seen with Monte Carlo applied to Random forest as well. LSTM had the second-best performance, followed by Decision Tree.

Overall, these results suggest that Random Forest is the most suitable model for the given problem, followed by Montecarlo with Random Forest.

2. **"Surge Multiplier Dataset"**

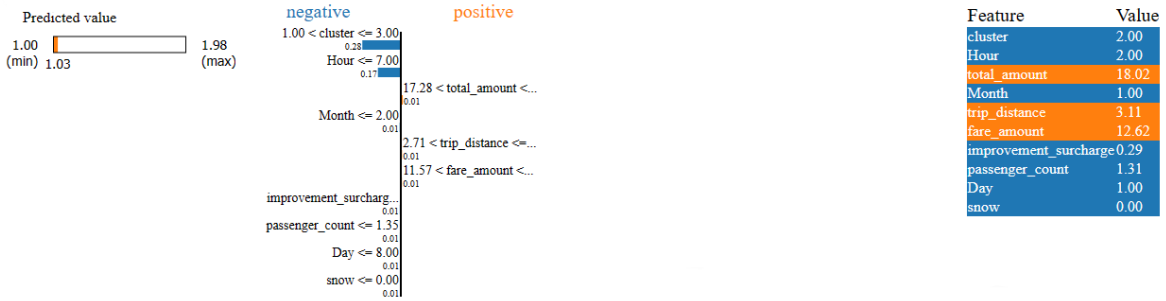
→ Feature Engineering:

2.1- Correlation analysis:

Based on the correlation analysis, the surge multiplier has a high positive correlation with the number of trips (0.97), followed by the hour of the day (0.41), and the congestion surcharge (0.41). On the other hand, the surge multiplier has a low positive correlation with features such as humidity (0.10), windspeed (0.09), and cloud cover (0.06).

2.2- LIME:

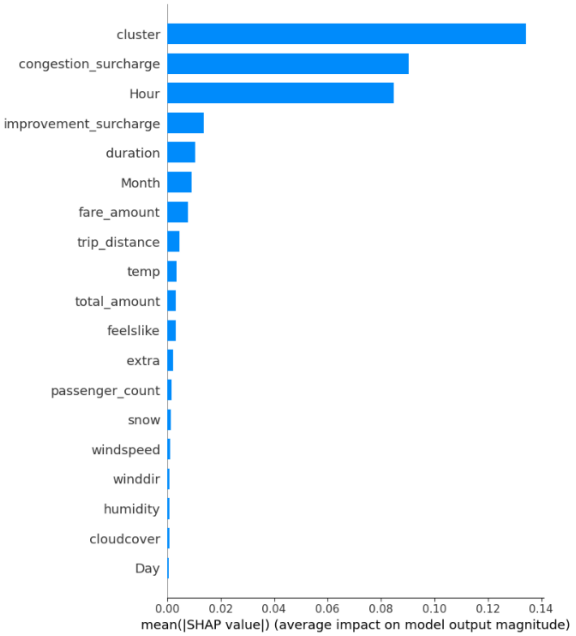
The LIME explainer shows the contribution of each feature to the prediction for a random row of data. The predicted value is slightly higher than the actual value. The Lime explainer also shows the contribution of different features towards the predicted value. For example, the hour of the day (0.28) and total amount (0.17) have the highest positive contribution towards the predicted surge multiplier value, while snow (0.01) and passenger count (0.01) have the least contribution.



2.3- SHAP:

The SHAP summary plot shows the average impact of each feature on the model output.The most important features are cluster (0.14), followed by

congestion surcharge and hour (0.11), and improvement surcharge, duration, month, and fare amount around 0.03. The rest of the features have a relatively lower importance, around 0.01.



Overall, the most important features for predicting surge_multiplier appear to be num_trips, Hour, congestion_surcharge, Month, and total_amount. However, the importance of each feature may vary depending on the specific model and dataset used. It is recommended to try multiple feature selection techniques and compare the results to select the most relevant features.

→ Modeling Results:

Below, we will be comparing the performance of four different machine learning models: Decision Tree, Random Forest, and Monte Carlo, using Mean Squared Error (MSE), Mean Absolute Error (MAE), R Squared, and Average Cross Validation Score.

			Monte Carlo +	
	Decision Tree	**Random Forest**	Decision Tree	**Random Forest**
Mean Squared Error	0.014	**0.006**	0.014	0.007
Mean Absolute Error	0.052	**0.051**	0.056	0.052
R Squared	86.00%	**93.24%**	85.43%	93.03%
Average Cross Validation Score	84.25%	**92.44%**		

1. Random Forest: The mean squared error is 0.006, the mean absolute error is 0.051, the R-squared value is 93.24%, and the average cross-validation score is 92.44%.
2. Monte Carlo + Random Forest: The mean squared error is 0.007, the mean absolute error is 0.052, the R-squared value is 93.03%, and the average cross-validation score is not provided.
3. Decision Tree: The mean squared error is 0.014, the mean absolute error is 0.052, the R-squared value is 86.00%, and the average cross-validation score is 84.25%.
4. Monte Carlo + Decision Tree: The mean squared error is 0.014, the mean absolute error is 0.056, the R-squared value is 85.43%, and the average cross-validation score is not provided.

The Random Forest model outperformed the other models in terms of all metrics, followed by Monte Carlo with random forest, while the Monte Carlo + Decision Tree model had the highest mean absolute error.

III. Model Deployment in Cloud

2.1 Deployment in GCP [Google Cloud Platform]

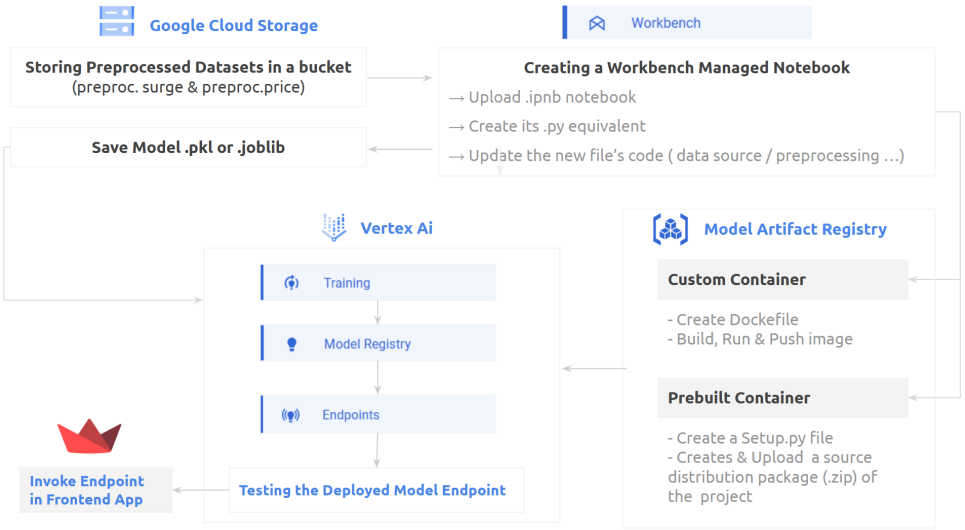


Figure 4: GCP Model Deployment Process

The methodology for deploying a model in GCP using the given steps can be outlined as follows:

1. Storing Preprocessed Datasets in a bucket: Before deploying the model, we downloaded the new locally preprocessed datasets (preproc.surge and preproc.price). Those are then stored in a bucket in Google Cloud Storage. This allows for easy access to the datasets during the model deployment process.
2. Creating a Workbench Managed Notebook: A managed notebook is created in Google Cloud's Workbench. The surge.ipnb and price.ipnb notebooks are uploaded to the managed notebook, and a .py equivalent is created. The code in the new file is updated to include the appropriate data source and preprocessing steps. We made sure to delete any preprocessing steps since we are now using the preprocessed dataset.
3. Saving the Model: Once the model has been developed, it needs to be saved. The model can be saved as a .pkl or .joblib file, which can be loaded later for inference.
    a. For when we used a custom Container: when a custom container was used for the deployment, a Dockerfile was created, specifying the environment and dependencies. The container is then built, run and

pushed to a container registry, through the pre build command prompt.

b. For when we used a Prebuilt Container: when a prebuilt container is used for the deployment, a setup.py file needs to be created. The file creates and uploads a source distribution package (.zip) of the project. One this appropriate setup is done, we can start the training with one of the relevant predefined containers.

4. Vertex AI: In Vertex AI, the model is trained, and then the Model Registry is used to create an endpoint for the model. This allows the model to be deployed and accessed from various applications.
5. Testing Deployed Model Endpoint: Once the model is deployed, it needs to be tested. This can be done by invoking the model's endpoint and checking the results.

A detailed documentation including screenshots from the different steps is provided in a separate document called ''Step by Step - How to Deploy a Model in Cloud'' for better understanding.

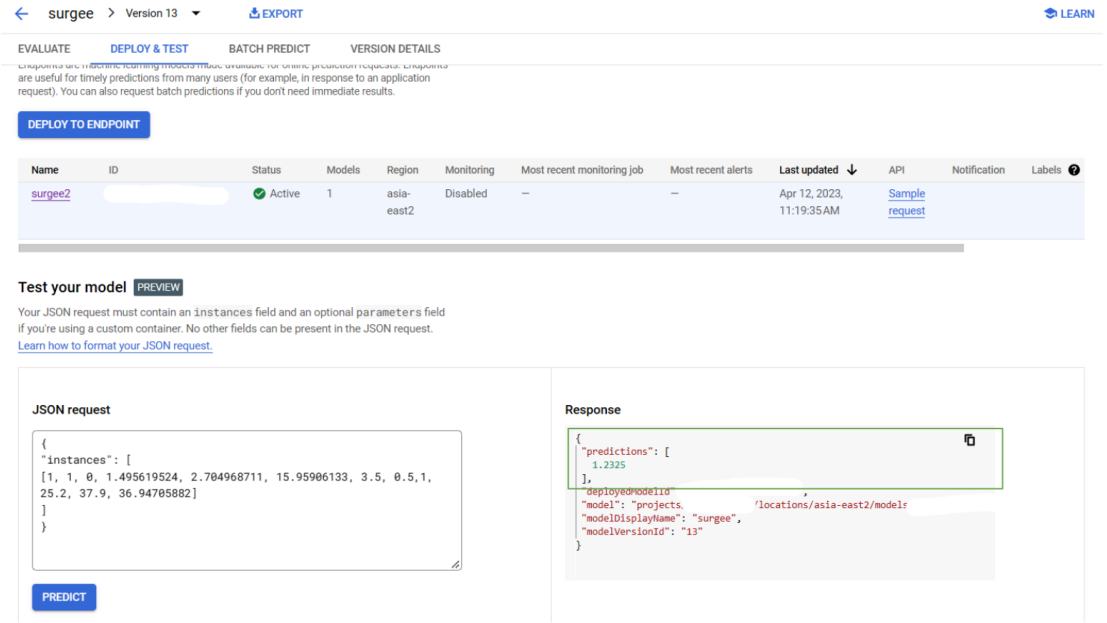


Figure 5: Result of the Testing of surge Model in GCP

2.2 Deployment in AWS [Amazon Web Services]

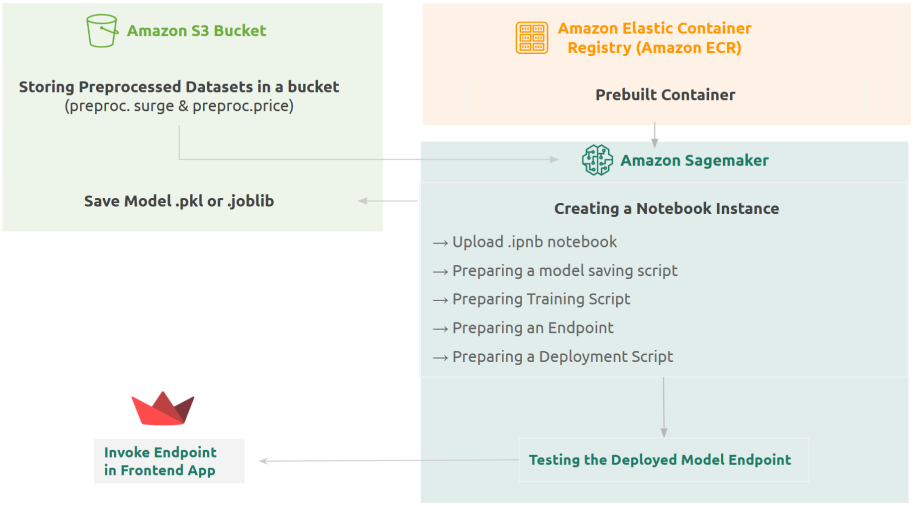


Figure 6: AWS Deployment Process

In order to compare the different service providers, we decided to try deploying our models in both GCP and AWS. For the latter, we opted for a code-oriented approach such that, the different steps of the process (training, endpoint deployment..) were coded instead of done with the AWS UI experience. Those are the steps for the deployment:

- Storing the preprocessed datasets in an S3 bucket named datasetsthesis2. The preprocessed datasets are preprocessed_surge.csv and preprocessed_price.csv.

- In Amazon Sagemaker, create a Notebook instance with the desired compute configuration and IAM role permissions.
- Upload the Jupyter notebooks files (.ipynb) for the two models and include a model saving script (joblib or pkl)to the Notebook instance.
- Include the training script: The script will define the model, its hyperparameters, and how the data is processed.

```
In [28]: from sagemaker.sklearn.estimator import SKLearn

sklearn_estimator = SKLearn(
    entry_point='script.py',
    role=get_execution_role(),
    instance_count=1,
    instance_type='ml.c5.2xlarge',  # Updated instance type
    framework_version='0.23-1',
    base_job_name='rf-scikit',
    hyperparameters={'n-estimators': 500,
                     'max_leaf_nodes': 16})
```

```
In [29]: sklearn_estimator.fit({'train':trainpath, 'test': testpath}, wait=False)

INFO:sagemaker:Creating training-job with name: rf-scikit-2023-04-07-18-47-47-849
```

```
In [30]: sklearn_estimator.latest_training_job.wait(logs='None')
artifact = m_boto3.describe_training_job(
    TrainingJobName=sklearn_estimator.latest_training_job.name)['ModelArtifacts']['S3ModelArtifacts']

print('Model artifact persisted at ' + artifact)
```

```
2023-04-07 18:47:48 Starting - Starting the training job....
2023-04-07 18:48:16 Starting - Preparing the instances for training........
2023-04-07 18:49:03 Downloading - Downloading input data....
2023-04-07 18:49:28 Training - Downloading the training image
2023-04-07 18:49:34 Training - Training image download completed. Training in progress.....
2023-04-07 18:49:54 Uploading - Uploading generated training model.
2023-04-07 18:50:05 Completed - Training job completed
Model artifact persisted at s3://sagemaker-eu-north-1-378412049928/rf-scikit-2023-04-07-18-47-47-849/output/model.tar.gz
```

- Prepare the endpoint configuration in the script.py file, which defines the runtime environment for the model, such as the Docker container image and the amount of compute resources. Prepare a deployment script that creates an endpoint with the desired configuration and deploys the trained model to it.

```
In [31]: import boto3
sm_client = boto3.client('sagemaker')
endpoints = sm_client.list_endpoints()
endpoint_configs = sm_client.list_endpoint_configs()
```

```
In [32]: from sagemaker.predictor import csv_serializer
predictor = sklearn_estimator.deploy(instance_type='ml.c5.2xlarge',initial_instance_count=1, serializer=csv_serializer)

INFO:sagemaker:Creating model with name: rf-scikit-2023-04-07-18-50-26-961
INFO:sagemaker:Creating endpoint-config with name rf-scikit-2023-04-07-18-50-26-961
INFO:sagemaker:Creating endpoint with name rf-scikit-2023-04-07-18-50-26-961

----!
```

- Test the deployed model endpoint.

```
In [33]: attributes = ['Day', 'Month', 'Hour', 'passenger_count', 'trip_distance', 'total_amount', 'temp', 'feelslike', 'snow', 'windspeed

predictor.predict(testX[attributes].values)

WARNING:sagemaker.deprecations:The csv_serializer has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.

Out[33]: array([1.0075, 1.9225, 1.2475, 1.1525, 1.7825, 1.8425, 1.5375, 1.745 ,
       1.4575, 1.19  , 1.1875, 1.03  , 1.575 , 1.5   , 1.4775, 1.42  ,
       1.1525, 1.775 , 1.5125, 1.6675, 1.7525, 1.7   , 1.6   , 1.3575,
       1.51  , 1.2225, 1.5925, 1.    , 1.3825, 1.    , 1.0075, 1.0025,
```

2.3 Comparing GCP and AWS Deployment Experiences

If we were to compare the difference of experiences between GCP and AWS, these would be the most important points:

- GCP and AWS: These are two of the leading cloud computing providers, offering a wide range of services including machine learning. They are both very popular and offer many similar services, although there are some differences in terms of pricing, customization, and control. Also, both GCP

and AWS provide various machine learning services, including tools for developing, training, and deploying machine learning models.

- Vertex AI and Amazon SageMaker are specific machine learning services offered by GCP and AWS, respectively. They are designed to provide a range of tools for developing and deploying machine learning models, including model training and deployment. The level of control that we have over the machine learning models especially for the training and deployment is different as, it was personally felt that AWS offered more levels of customization and control thanGCP for our example. Moreover, both GCP and AWS offer free usage for certain services, typically for a limited time or with a cap on usage. GCP provides $300 free credits over 90 days, while AWS offers a percentage usage per service over one month. The service providers also offer documentation and examples to help users utilize their ML services. GCP's documentation is extensive and includes well detailed videos, tutorials and documents, while AWS's documentation is relatively less extensive but includes pertinent examples.

	Google Cloud	aws
ML services	Vertex ai	Amazon SageMaker
Deployment Options	Cloud Run	Amazon SageMaker
Storage	Cloud Storage	S3
Customization and Control	Less granular control, more automated	More granular control over models
Methodology	GCP UI for deployment Jupyterlab for coding	Notebook Instances - all steps coded
Free Tier Pricing Model	300$ free credits over 90 days	% usage per service over 1 month
Documentation	Extensive Documentation (videos , documents..)	Relatively less extensive Documentation but pertinent examples

Figure 7: GCP VS AWS

IV. Real Time Simulation

Important Note:

Before starting the introduction of the front-end application developed to showcase our newly developed models, It is important to note that, the following step after testing the endpoint deployed is to Invoke it in the frontend App. However, we were not able to do that given that we signed up for the free tier and invoking the endpoint requires the activation of the billing options. As a result, we opted for a simpler solution which consists of uploading our saved models (.pkl) to our github repository, then unpickling them within our frontend python file.

For the development of our frontend application, we opted for a streamlit library which is an open-source Python library that allows users to quickly create interactive web applications and data dashboards using simple Python syntax.
For our models to run properly those are the input we need:

- **Trip input:** Day- Month- Hour- passenger_count - trip_distance - trip duration location id
- **Weather input:** temperature- feelslike- snow- windspeed- cloudcover-

The application will be used to redirect users to different pages in the application. The options in the list include "Feature Selection," "Algorithms Performances," and "Real Time Demo." The first two sections are mainly for explaining some technical aspects of our algorithms, however, we will be focusing mainly on the 3rd page, which is the real time demo of how our new dynamic pricing works. Below, we will explain how we are collecting the different required information for our models to run properly.
This is the link to the simulation application: **link**

Collecting Trip input:

- Trip distance , Trip duration, Departure Location id

  Once the user is on the main page, the application will

  - retrieve data from one of the New York City yellow taxi pickups APIs using the requests module. If the response from the API is successful (status code 200), the data will be converted to a JSON object, normalized using the json_normalize method, and then stored in a Pandas DataFrame. This API to gets the location IDs and coordinates

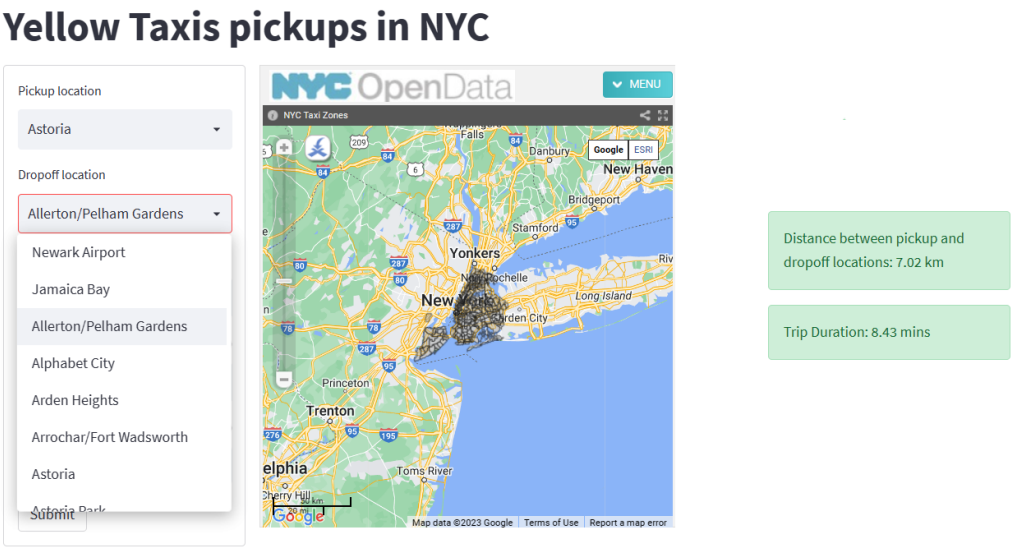
for New York City's boroughs and zones. It has the main source as the initial dataset that we retrieved earlier, hence the same regions, ids…

- The information is extracted from the JSON object, and then location IDs, zones, latitudes, and longitudes are stored in a dictionary, which is then used to create a Pandas DataFrame.

We also defined a function that calculates the distance between two sets of latitude and longitude coordinates using the Haversine formula. It also defines a function to estimate the duration of a trip based on the distance and an assumption that the average speed in general should be around 60 km/h.

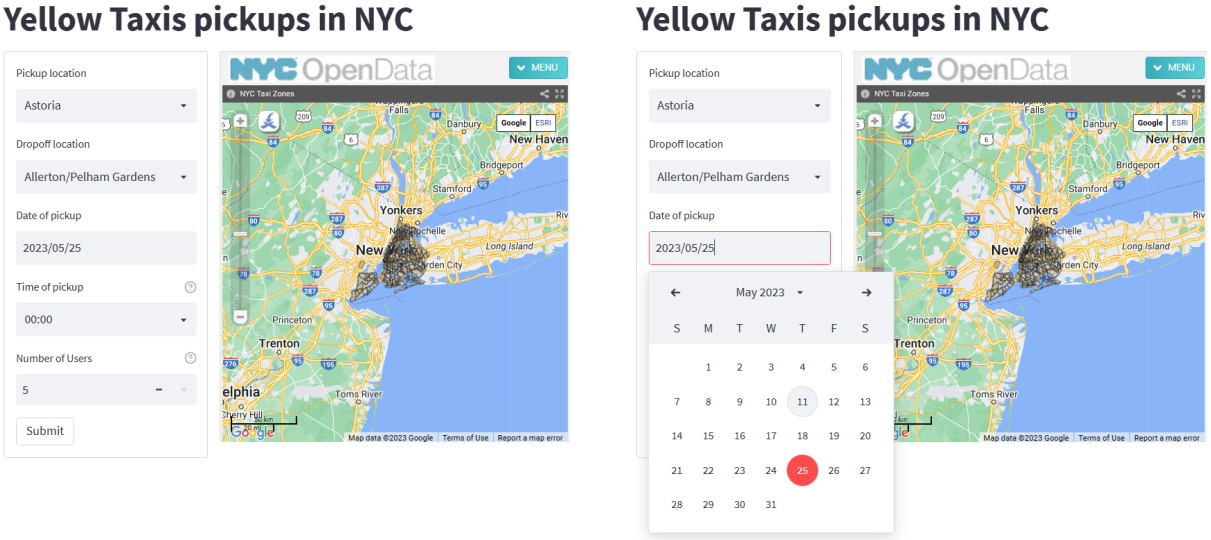
As a result, when the user will choose the start and end locations, we will go through the previously defined dictionary to fetch their ids , calculate the distance between them and the potential duration of the demand, given the assumptions.

Example:



- Day , Month , Hour, Passenger Count

The users are provided with a form that allows them to input information about their taxi ride, including date and time of pickup, and the number of passengers. Once the user submits the form, those information are used to retrieve weather data for the date of the trip, as will be explained below.



- Weather Information

  We retrieved weather data for the date of the trip from a separate dataset of historical weather data for New York City. Our approach consisted of filtering the weather data based on the user's input date and extracting the relevant weather features, including temperature, feels-like temperature, snow, wind speed, and cloud cover. Those are the weather inputs needed for our models to run.

Dynamic Surge Price of the trip:

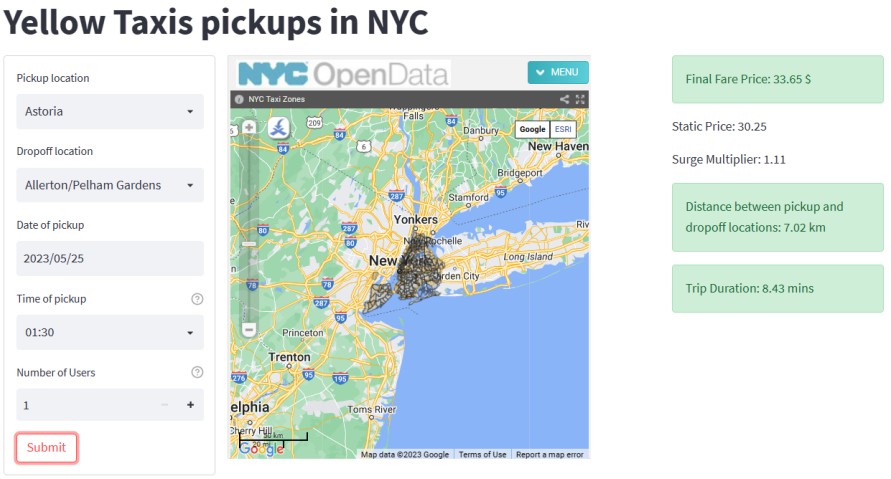
After getting all the information we need, we loaded the two machine learning models that have been pre-trained on NYC taxi data to estimate the cost of the taxi ride. The first model estimates the trip fare, and the second model estimates the surge multiplier. The final price that will be given to the customer is:

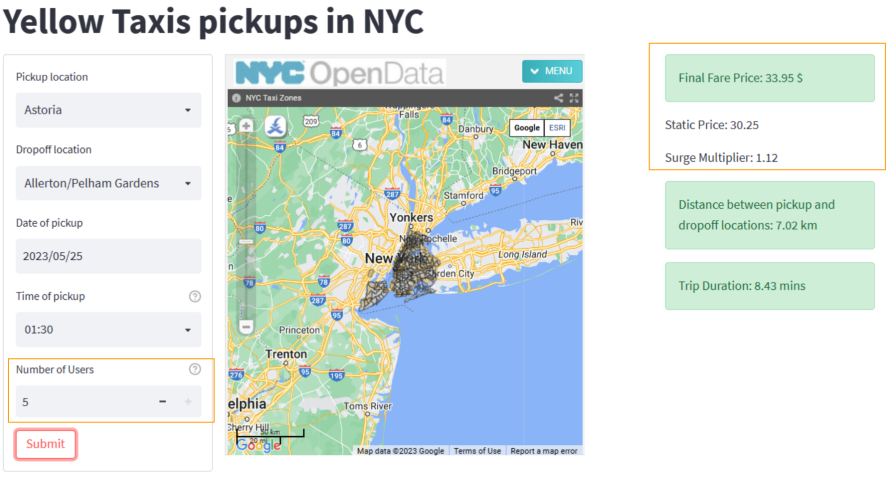
Dynamic Surge Price = Surge Multiplier [model 2] * **Basic Price** [model 1]

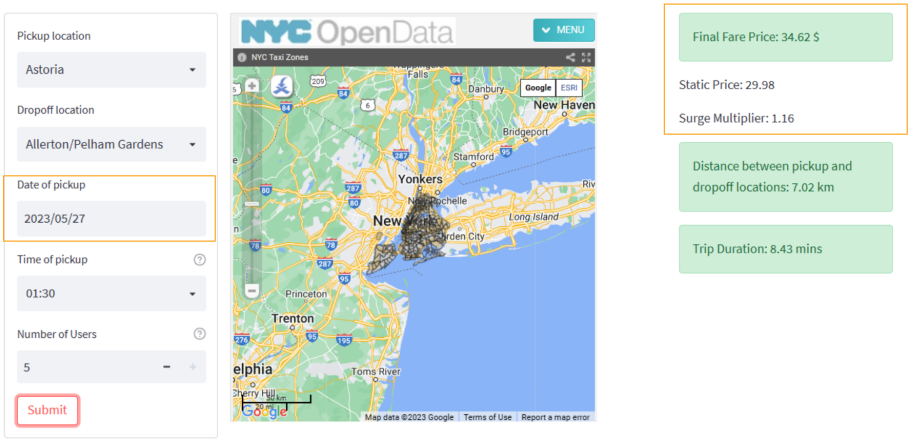
Once the ML models, it uses the input data to predict the fare and total amount charged for the taxi ride. The script then displays the estimated fare and total amount charged in a separate panel of the web application.

Overall, the app is a demonstration of the use of machine learning models and data analysis techniques to estimate the cost of a taxi ride, while including the notion of surge multiplier and taking into account weather conditions with other features, in New York City.

Testing the demo - Updating the number of users and the date of pickup







Return on Experience - Streamlit :

We opted for the open-source Python library Streamlit to build our app since it is one of the latest libraries published for building interactive web apps for data science and machine learning tasks. Indeed, after working with it, it is definitely a strong tool that, thanks to its user-friendly interface and intuitive design, makes the process of creating online applications simpler. Using straightforward Python scripts, we were able to quickly build applications, and deploy machine learning models in real-time. The most important feature is that it can be shared and easily deployed to the cloud using the Streamlit cloud platform. Additionally, it provides a simple method for connecting the GitHub repository, which makes it simpler for teams to make changes to the application through simple commits.

As a whole, using Streamlit in our project resulted in a favorable return on experience because of how simple it is to use and how well-written its documentation is. Moreover, their community is quite active, providing a wealth of resources, tutorials, and examples, making it simpler to start using the library and produce high-quality apps quickly.

V. Conclusion

In conclusion, this report presents the methodology used to collect, preprocess, and model data for dynamically estimating the prices of taxi rides, taking into account the surge multiplier effect,and including weather conditions in the prediction. The work was done on the available NYC taxi dataset, however, it is important to note that we opted for a generalized approach (function-first code..) which allows to use the same code for any taxi rides, and weather datasets all over the world.

For the sake of our project, we collected two datasets, one containing information about taxi rides and the other about weather conditions in the city. These datasets were merged to create a new static price dataset. To capture the surge multiplier effect, the number of trips per hour for each day in the dataset was calculated, and surge multipliers were added to the dataset based on the quartiles of the demand for taxis. This methodology lays the groundwork for the deployment of the best model in later stages of the project.

The data preprocessing involves three main steps: data cleaning, data transformation, and feature engineering. The goal is to ensure that the data is clean, transformed, and engineered in a way that is suitable for analysis and allows for meaningful insights to be drawn. We used k-means clustering algorithm to determine the optimal number of clusters for stratified sampling technique to divide the data into train and test sets. For data modeling, we used four machine learning models: decision tree, random forest, LSTM (only for pricing modeling), and Monte Carlo simulation. The model evaluation methodology includes four main metrics: Mean Squared Error, Mean Absolute Error, R Squared, and Average Cross Validation Score. The results showed that, even after fine tuning, the random forest model outperformed the other models in terms of all four metrics for both models, and therefore, it can be used for future prediction of the ride-hailing demand and surge multiplier.

Finally, we deployed the two different machine learning models in both Google Cloud Platform and Amazon Web Services. We highlighted the similarities and differences between the two service providers, including their machine learning services, pricing, and control levels. While both GCP and AWS offer similar services,

we found that AWS provides more customization and control options, which was one of the main learning points of the work.
For the sake of showing how ML can be useful in real life and can improve the user experience of customers and business profitability of drivers, we developed a real-time simulation application using the library “streamlit”that showcases how the newly developed machine learning models for estimating the cost of a taxi ride in New York City have been successfully implemented. Although the endpoint could not be invoked in the front-end application due to billing options not being activated, a simpler solution of uploading the saved models (.pkl) to the Github repository was utilized. The application collects various inputs such as trip distance, trip duration, departure location ID, date, time of pickup, number of passengers, and weather information to estimate the dynamic surge price of the trip using two pre-trained models

VI. Future Improvements

While the methodology presented is likely to provide a solid foundation for estimating the dynamic pricing of taxi rides in New York City, there are several areas for potential improvement. Those could include in the first place the activation of billing options to allow the endpoint to be invoked in the front-end application and expanding the data sources used for the models.
Moreover, while the real-time simulation application developed in this work provides a useful proof-of-concept for the implementation of machine learning models in a real-world setting, there are several areas for improvement in the user interface and functionality of the application. Future work could focus on improving the user experience of the application, adding additional features such as real-time traffic updates, integrating google maps api for a more accurate and reliable estimation of the distance and duration of the trip.
Overall, there is potential for future work to build upon the foundation laid out in order to continue improving the usefulness of machine learning models for reshaping taxi rides in urban areas.

VII References

[1] Matching and Dynamic Pricing in Ride-Hailing Platforms - Microsoft Research. (2018, May 1). Microsoft Research. Retrieved January 26, 2023, link

[2] Dynamic Pricing Explained: Machine Learning in Revenue Management and Pricing Optimization. (2019, April 24). AltexSoft. Retrieved January 26, 2023, link

[3] Automated topic modeling of tourist reviews: Does the Anna Karenina principle apply? (2020, October 13). Automated Topic Modeling of Tourist Reviews: Does the Anna Karenina Principle Apply? link

[4] S. Shakya, M. Kern, G. Owusu, C.M. Chin, Neural network demand models and evolutionary optimisers for dynamic pricing, Knowledge-Based Systems, Volume 29, 2012, Pages 44-53, link

[5] (2011, July 18). Neural network demand models and evolutionary optimisers for dynamic pricing. link

[6] Schlosser, Rainer & Richly, Keven. (2019). Dynamic Pricing Competition with Unobservable Inventory Levels: A Hidden Markov Model Approach. Communications in Computer and Information Science. 966. 15-36. 10.1007/978-3-030-16035-7_2. link

[7] Optimizing the price of retail products- by Leigha Jarett- published in codelabs.developers.google.com link

[8] (The Wheel of Dynamic Pricing: Towards Open Pricing and One to One Pricing in Hotel Revenue Management, 2022) link

[9] How Machine Learning Is Helping In Providing Dynamic Pricing , by Dr Bright (PhD Data Science), published in Total Data Science link

[10] 2021 Yellow Taxi Trip Data | NYC Open Data. (n.d.). 2021 Yellow Taxi Trip Data | NYC Open Data. Retrieved February 4, 2023, link

[11] Corporation, V. C. (n.d.). Weather Data Services | Visual Crossing. Weather Data Services | Visual Crossing. Retrieved February 4, 2023, link

[12]Rosenthal, Brian M. "New York Is Urged to Consider Surge Pricing for Taxis." The New York Times, The New York Times, 30 Jan. 2020, link

[13] S, S., & Nirai Vaani M, K. (2020). A machine learning framework for profitability profiling and dynamic price prediction for the New York City taxi trips. International Journal of Innovative Technology and Exploring Engineering, 9(5), 1-10. link

[14] Custom training overview | Vertex AI | Google Cloud. (n.d.). Google Cloud. link
[15] Vertex AI: Use custom prediction routines with Sklearn to preprocess and postprocess data for predictions | Google Codelabs. (n.d.). Google Codelabs. link
[16] Lukasz, O. (2022, November 8). Monitoring ML models with Vertex AI. Medium. link
[17] Dichiera, M. (2023, January 24). Step-by-step guide to build Personalized Airbnb Recommendation App with Streamlit. Medium. link
[18] Connect Streamlit to Google Cloud Storage - Streamlit Docs. (n.d.). Connect Streamlit to Google Cloud Storage - Streamlit Docs. link
[19] How to build, train, and deploy a machine learning model with Amazon SageMaker. (n.d.). Amazon Web Services, Inc. link

[20] Li, H. (2020, May 1). AWS Sagemaker Introduction & Tutorial. Medium. link

[21] Quickstart: Build and push a Docker image with Cloud Build | Cloud Build Documentation | Google Cloud. (n.d.). Google Cloud. link

List of Figures

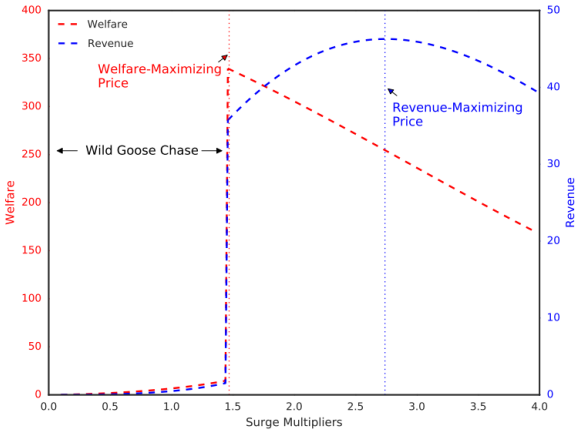


Figure1:how welfare and revenue change under dynamic pricing (DP) [Uber - San Francisco downtown]

Source: Automated Topic Modeling of Tourist Reviews: Does the Anna Karenina Principle Apply?, 2020

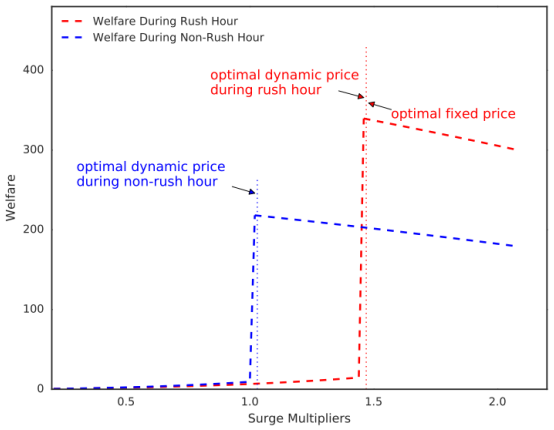


Figure2: Welfare under DP during rush and non-rush hour [Uber - San Francisco downtown]

Source: Automated Topic Modeling of Tourist Reviews: Does the Anna Karenina Principle Apply?, 2020

Figure 3: Model Selection Process [Generalized]

Figure 4: GCP Model Deployment Process

Figure 5: Result of the Testing of surge Model in GCP

Figure 6: AWS Deployment Process

Figure 7: GCP VS AWS