Building A Machine Learning To Predict Taxi Trip Duration Using Partial Data

Ammar Alyousfi (WQD170064)

21 May 2019

ABSTRACT

Taxis play an imperative role in the transportation system today; they are used by millions of people around the world every day to travel from one place to another. Improving taxi services has positive effects on taxi passengers and taxi transportation companies. One of the important things with regard to improving taxi services is trip-duration prediction where the duration of a taxi trip is predicted before the trip starts. With accurate predictions, passengers can plan their time more effectively and transportation companies can use their resources in a more efficient manner. In this project, we follow the data-science process and principles—using Python and many of data-science and machine-learning libraries—to build a machine-learning model that can predict taxi-trip durations with a good accuracy using partial data (i.e. no full-GPS-trajectories needed). In this project, we get the data, clean and prepare it, and visualize and explore its contents; then we apply feature engineering to create new useful features; after that, we apply numerous effective modeling techniques to build the predictive model of the study which we show that it achieves a good predictive performance; finally, we describe how we deploy this model as an open-source web application.

ABSTRAK

Teksi memainkan peranan penting dalam sistem pengangkutan hari ini; mereka digunakan oleh berjuta-juta orang di seluruh dunia setiap hari untuk bergerak dari satu tempat ke tempat lain. Memperbaiki perkhidmatan teksi mempunyai kesan positif ke atas penumpang teksi dan syarikat pengangkutan teksi. Salah satu perkara penting yang berkaitan dengan peningkatan perkhidmatan teksi adalah ramalan tempoh perjalanan di mana tempoh perjalanan teksi diramalkan sebelum perjalanan bermula. Dengan ramalan yang tepat, penumpang boleh merancang masa mereka dengan lebih berkesan dan syarikat pengangkutan dapat menggunakan sumber daya mereka dengan cara yang lebih efisien. Dalam projek ini, kami mengikuti proses sains data dan prinsip-prinsip menggunakan Python dan banyak akademik data dan sains pembelajaran komputer-untuk membina model pembelajaran mesin yang dapat meramalkan tempoh perjalanan teksi dengan ketepatan yang baik menggunakan data separa (iaitu tiada laluan penuh GPS-diperlukan). Dalam projek ini, kami mendapatkan data, membersihkan dan menyediakannya, dan menggambarkan dan meneroka kandungannya; maka kami menerapkan kejuruteraan ciri untuk mencipta ciri-ciri berguna yang baru; selepas itu, kami menggunakan banyak teknik pemodelan berkesan untuk membina model ramalan kajian yang kami menunjukkan bahawa ia mencapai prestasi ramalan yang baik; Akhirnya, kami menerangkan bagaimana kami menggunakan model ini sebagai aplikasi web sumber terbuka.

ACKNOWLEDGEMENT

I would like to express my gratitude and appreciation to my supervisor Associate Prof. Dr. Salimah Binti Mokhtar for her support, help, and guidance in the making of this project. I would also like to deeply thank my family for their great efforts and continuous support.

TABLE OF CONTENTS

1 INTRODUCTION **1**

1.1 Background 1

1.2 Problem Statement 2

1.3 Research Objectives 3

1.4 Research Questions 3

1.5 Expected Outcome 3

1.6 Scope of the Study 3

1.7 Significance of the Research 3

1.8 Organization of this Document 4

2 LITERATURE REVIEW **5**

2.1 Introduction 5

2.2 Related Work 5

2.3 Conclusion 16

3 RESEARCH METHODOLOGY **17**

3.1 Introduction 17

3.2 Datasets 17

3.2.1 Taking a Look at a Sample of the Datasets 20

3.2.2 Datasets Sizes 21

3.2.3 Philosophy Behind Using the Weather Datasets 22

3.2.4 New Limitation in the Taxi-Trips Datasets 22

3.2.5 Datasets Naming in Programming Code 22

3.3 Research Design 23

3.4 Tools 26

3.5 Evaluation Metric 27

3.6 Conclusion 27

4 DATA PREPARATION AND EXPLORATION 28

4.1 Introduction 28

4.2 Data Acquisition 28

4.3 Data Cleaning 28

4.3.1 Dealing with Missing Values 28

4.3.2 Checking for Duplicate Records 29

4.3.3 Removing Incorrect and Inconsistent Data 30

4.3.4 Data-Type Conversion 32

4.3.5 Unit Conversions 33

4.4 Creation of the Target Variable 34

4.5 Exploratory Data Analysis 34

4.5.1 Distribution of Trip-Duration Variable 34

4.5.2 Number of Taxi Trips Per Day of the Month 36

4.5.3 Number of Taxi Trips Per Hour of the Day 38

4.5.4 Distribution of the Number-of-Passengers Variable 40

4.5.5 Distribution of Trip-Distance Variable 42

4.5.6 Distribution of Rate-Code-ID Variable 43

4.5.7 Distribution of Store-and-Forward-Flag Variable 44

4.5.8 Distribution of Payment-Type Variable 45

4.5.9 Distribution of Total-Charge-Amount Variable 47

4.5.10 Number of Taxi Trips by Borough 48

4.5.11 Distribution of Daily-Temperature Variables 50

4.5.12 Distribution of Daily-Precipitation Variables 52

4.6 Expolratory Data Analysis: Exploring Relationships between Variables . 54
4.6.1 Relationship between Trip Duration and Trip Distance 54
4.6.2 Average Trip Duration Per Hour of the Day 58
4.6.3 Average Trip Duration Per Borough 59
4.6.4 Average Trip Duration Per Borough and Hour of the Day 59
4.6.5 Average Trip Duration Per Day of the Month 60
4.6.6 Relationship between Pickup-Borough and Pickup-Times 61
4.6.7 Relationship between Number of Passengers and Trip Duration . 64
4.6.8 Relationship between Rate-Code ID and Trip Duration 64
4.6.9 Relationship between Vendor ID and Trip Duration 65
4.6.10 Relationship between Store-and-Forward Flag and Trip Duration . 65
4.6.11 Relationship between Total Charge Amount and Trip Duration . . 66
4.6.12 Correlation between Variables 67
4.7 Conclusion 69
5 MODELING AND RESULT DISCUSSION 70
5.1 Introduction 70
5.2 Feature Selection 71
5.3 Feature Engineering 73
5.3.1 Extraction of Features from Pickup-Datetime 73
5.3.2 Adding Temperature Features 73
5.3.3 Adding Precipitation Features 74
5.3.4 Adding Borough and Service-Zone Features 74
5.3.5 Feature Encoding 75
5.3.6 Adding Origin-and-Destination-Boroughs Features 78
5.3.7 Removing Features that Have Zero Variance 78

5.4 Model Selection and Hyperparameter Optimization 79

5.5 Building an Ensemble Model 83

5.6 Training and Evaluating the Final Model 83

5.7 Feature Importance 85

5.8 Conclusion 86

6 DEPLOYMENT 87

6.1 Introduction 87

6.2 Deployment Process 87

6.2.1 Deployment to Google App Engine 87

6.2.2 Usage and Deployment of the Predictive Model 88

6.2.3 Input Data of the User 88

6.2.4 The Application Interface 89

6.3 Limitations 90

6.4 Conclusion 91

7 CONCLUSION 92

7.1 Limitations 92

7.1.1 Change in the Dataset of the Study 92

7.1.2 Computation Limitation 93

7.1.3 Time Limitation 93

7.2 Future Work and Recommendations 93

7.3 Study Code and Files 94

LIST OF FIGURES

2.1 A visualization of the framework implemented by Li et al (2017). Reprinted from 'Urban travel time prediction using a small number of gps floating cars' by Y. Li, D. Gunopulos, C. Lu, and L. Guibas, 2017, *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* 6

2.2 The model proposed by Jindal et al (2017) appears on the left of the figure. On the right appear two other models used for comparison. Reprinted from 'A unified neural network approach for estimating travel time and distance for a taxi trip' by I. Jindal, X. Chen, M. Nokleby and J. Ye, 2017, *arXiv preprint.* 8

2.3 Average RMSE for each day of the week (on the left side) and for each 30-minutes period of the day (on the right side) for the model proposed by Singh et al (2015). Reprinted from 'Taxi trip time prediction using similar trips and road network data' by A. D. Singh, W. Wu, S. Xiang, and S. Krishnaswamy, 2015, *2015 IEEE International Conference on Big Data (Big Data).* 12

3.1 New York City map with taxi zones highlighted. From NYC Taxi Zones by NYC OpenData, https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc. 19

3.2 A sample of the taxi-trips datasets 21

3.3 A sample of the helper location table 21

3.4 A sample of the weather datasets 21

3.5 Cross-Industry Standard Process for Data Mining (CRISP-DM). From Wikimedia. Retrieved from https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png. Copyright 2012 by Kenneth Jensen released under the Creative Commons Attribution-Share Alike 3.0 Unported license. 24

4.1 Code to check the number of missing values in the project datasets and its results 29

4.2 Code to check for duplicate records in the project datasets and its results . 29

4.3 Code used to check for errors in pickup and drop-off times and delete the erroneous records 30

4.4 Code used to check for errors in trip distances and delete the erroneous records 31

4.5 Code used to check for errors in trip charge amount and delete the erroneous records 31

4.6 Code used to check for errors in trip passenger-count and delete the erroneous records 32

4.7 Code used to convert some data fields to datetime format 32

4.8 Code used to replace the "T" values and convert certain data fields in weather datasets to float format 33

4.9 Code used to create the target variable in taxi-trips datasets 34

4.10 Histogram of trip_duration of training data 35

4.11 Histogram of trip_duration of test data 35

4.12 Number of trips in each day of March 2017 36

4.13 Weather data for some days of March 2017 37

4.14 Number of trips in each day of March 2018 37

4.15 Weather data for some days of March 2018 38

4.16 Number of taxi trips per hour in March 2017 39

4.17 Number of taxi trips per period of the day in March 2017 39

4.18 Number of taxi trips per hour in March 2018 40

4.19 Number of taxi trips per period of the day in March 2018 40

4.20 Distribution of the passenger-count variable for March-2017 data 41

4.21 Distribution of the passenger-count variable for March-2018 data 41

4.22 Histogram of trip distances in March 2017 42

4.23 Histogram of trip distances in March 2018 43

4.24 Number of trips for each rate code in March 2017 43

4.25 Number of trips for each rate code in March 2018 44

4.26 Distribution of store-and-forward-flag values in March 2017 45

4.27 Distribution of store-and-forward-flag values in March 2018 45

4.28 Number of trips for each payment type in March 2017 46

4.29 Number of trips for each payment type in March 2018 46

4.30 Distribution of trip total-charge-amount in March 2017 47

4.31 Distribution of trip total-charge-amount in March 2017 after removing an erroneous record 48

4.32 Number of taxi trips for each borough as an origin and a destination in March 2017 49

4.33 Number of taxi trips for each borough as an origin and a destination in March 2018 49

4.34 Daily temperature distribution in March 2017 50

4.35 Minimum, average, and maximum temperature for each day in March 2017 51

4.36 Daily temperature distribution in March 2018 51

4.37 Daily temperature distribution in March 2017 and March 2018 52

4.38 Minimum, average, and maximum temperature for each day in March 2018 52

4.39 Daily precipitation distribution in March 2017 53

4.40 Daily snow-fall distribution in March 2017 53

4.41 Daily precipitation distribution in March 2018 54

4.42 Daily snow-fall distribution in March 2018 54

4.43 Scatter plot between trip duration and trip distance in March 2017 (left) and in March 2018 (right) 55

4.44 2D histogram of trip duration and trip distance in March 2017 (left) and in March 2018 (right) 55

4.45 Distribution of taxi-trips speeds in March 2017 (left) and in March 2018 (right) 56

4.46 Code used to calculate MAD for taxi speeds in March 2017 57

4.47 Scatter plot between trip duration and trip distance in March 2017 (left) and in March 2018 (right) after removing trips with wrong speeds 58

4.48 Average trip duration by pickup hour-of-the-day in March 2017 (left) and March 2018 (right) 58

4.49 Average trip duration per borough in March 2017 (left) and March 2018 (right) 59

4.50 Average trip duration per borough per hour of the day in March 2017 . . . 60

4.51 Average trip duration per borough per hour of the day in March 2018 . . . 60

4.52 Average trip duration per day of the month in March 2017 (left) and in March 2018 (right) 61

4.53 Number of taxi trips in each borough in each hour of the day in March 2017 62

4.54 Number of taxi trips in each borough in each hour of the day in March 2018 63

4.55 Trip duration per the number of passengers in March 2017 (left) and March 2018 (right) 64

4.56 Trip duration distribution for each rate-code ID in March 2017 (left) and March 2018 (right) 65

4.57 Trip duration distribution for each vendor ID in March 2017 (left) and March 2018 (right) 65

4.58 Trip duration distribution for the cases of store-and-forward flag in March 2017 (left) and March 2018 (right) 66

4.59 Scatter plot between trip total-charge-amount and duration in March 2017 (left) and March 2018 (right) 66

4.60 Scatter plot between trip total-charge-amount and duration after removing outliers in March 2017 (left) and March 2018 (right) 67

4.61 Correlation between variables in taxi-trips datasets of March 2017 (left) and March 2018 (right) 68

4.62 Correlation between variables in weather datasets of March 2017 (left) and March 2018 (right) 68

5.1 The stages of the modeling process 70

5.2 The fields remaining in taxi-trips datasets after removing unwanted fields 72

5.3 The fields remaining in weather datasets after removing unwanted fields . 72

5.4 Code used to extract new features from pickup-datetime feature 73

5.5 Code used to add temperature-features from weather datasets 74

5.6 Code used to add precipitation-features from weather datasets 74

5.7 Code used to add borough and service-zone features using the helper location table 75

5.8 A sample of taxi-trips datasets after applying feature engineering 75

5.9 Feature conversion to numeric format 76

5.10 Code used to convert non-numeric features to numeric format 76

5.11 Code used to align the training and test datasets 77

5.12 Code used to perform PCA on one-hot-encoded location-ID features . . . 77

5.13 Example of pickup_borough and dropoff_borough features after encoding 78

5.14 Example of origin-and-destination-boroughs features 78

5.15 Code used to create origin-and-destination-boroughs features 78

5.16 Code used to delete features with zero variance 79

5.17 A visualization of k-fold cross-validation 80

5.18 Code used to perform k-fold cross-validation with LightGBM model . . . 81

5.19 Code used for hyperparameter optimization for LightGBM model 82

5.20 The operation of the ensemble model 83

5.21 The steps of the process of training and evaluating the final model of the project 84

5.22 Feature importances for LightGBM model 85

5.23 Feature importances for XGBoost model 85

6.1 The interface of the Flask web application 89

6.2 The user fills in the values for the trip 90

6.3 The prediction is calculated and displayed to the user 90

LIST OF TABLES

2.1 Test results of the model proposed by Li et al (2017) 7

2.2 Performance comparison between the model proposed by Jindal et al (2017) and other models for travel time estimation in seconds. 8

2.3 Performance comparison between the model proposed by Jindal et al (2017) and other models for travel distance estimation in miles. 9

2.4 The MAPE scores of the model proposed by Zhan et al (2013) for four periods in the seven days of the week 14

2.5 MSE scores of the AR model with the explanatory variables proposed by Nakata and Takeuchi (2004) 16

2.6 MSE scores of the state space model proposed by Nakata and Takeuchi (2004) 16

3.1 The description of data fields of taxi-trips data set 18

3.2 The description of data fields of weather data set 20

3.3 The number of rows and columns in the project datasets 21

3.4 Code variable names of the datasets used in the project 23

5.1 Fields to be removed from taxi-trips datasets and reasons of removing them 72

5.2 Selected models, their optimized parameters, and their cross-validation MAE score 82

LIST OF SYMBOLS AND ABBREVIATIONS

Abbreviation	Definition
MAE	Mean absolute error
DNN	Deep neural network
SVR	Support vector regression
RMSLE	Root mean squared logarithmic error
NYC	New York City
MNL	Multinomial logistic regression
LM	Levenberg-Marquardt
MAPE	Mean absolute percentage error
RMSE	Root mean square error
AR	Auto regression
MSE	Mean squared error
TLC	NYC Taxi and Limousine Commission
CRISP-DM	Cross-Industry Standard Process for Data Mining
EDA	Exploratory data analysis
PCA	Principal component analysis

CHAPTER 1: INTRODUCTION

1.1 Background

Nowadays, taxi transportation is one of the most popular ways of transportation in cities and urban areas. Taxis provide a convenient way for people to travel fast from one place to another with affordable costs. In the past few years, taxi transportation witnessed a substantial transformation to a new level with the rise of transportation network companies like Uber, Lyft, Grab, etc. which match passengers with drivers via mobile applications or websites.

Given the large prevalence of taxi transportation, many research studies have been done to improve taxi services. One of the prominent research topics related to taxi transportation is trip-duration prediction where researchers try to build models that are able to predict taxi trip duration before starting the trip. Trip duration is the time period between the start and the end of a taxi trip.

Predicting taxi trip duration successfully brings many benefits to different kinds of groups: it brings benefits to the passengers in taxi trips, to the many types of companies that offer taxi services, and to governmental and non-governmental organizations that aim to coordinate and improve transportational services in countries and cities.

For passengers, prediction of trip duration allows them to knwo in advance the time period a taxi trip will take from the desired origin to the desired destination. This enables passengers to plan their time more efficiently. For example, if someone has a meeting at a specific time of the day, he can use trip-duration prediction to know when to start his trip to the meeting location; this way, he will not start moving too early and consequently, he will not arrive too early and will not waste time waiting the meeting to start; also, he will not start moving late and consequently, he will not arrive late to the meeting.

For companies that offer taxi services, being able to predict taxi trip duration allows them to better manage and utilize their available taxicabs. This leads to a more operational efficiency, better resource management, and to more customer satisfaction with the service provided. For organizations that aim to improve transportational services, trip-duration

prediction can help them in creating or improving transportational plans and services by giving them the ability to test different plans with minimum time and cost. With trip-duration prediction, these organizations can improve the management of urban traffic. For example, they can find congestion points in traffic network and devise solutions to reduce the effects of these points.

Many approaches and methods have been used for trip-duration prediction and many machine-learning models have been used for the same purpose. We will review some of these approaches in the second chapter of this study and see where they achieve good results and where they fall short.

1.2 Problem Statement

There are a number of proposed solutions to the important problem of trip-duration prediction. Many of them depend on using historical GPS trajectories but not everyone has the capacity or means to store this huge trajectory data (Singh, Wu, Xiang, & Krishnaswamy, 2015). Many solutions also have limitations that lead to unstable predictions as in the work done by Zhan, Hasan, Ukkusuri, and Kamga (2013) which will be discussed in chapter 2 of this study. In addition to that, big transportation companies in the market have proprietary solutions for travel time prediction but these solutions are not published and the technical capabilities required to run these solutions are not known and it is probably not possible to have such capabilities for all entities that want to implement travel time prediction.

With the advent of big-data era and the unprecedented increase in the volume of data produced every day, new possibilities emerged. With big data and machine learning, many astounding applications have been developed recently. In this study, we will harness the power of big data and machine learning to build a model that can predict taxi trip duration accurately. We aim to build this model using a minimal set of features (partial data) that are easy to get and store—features like pickup location, pickup time, and drop-off location; GPS trajectories will not be needed for the model. The program used to build the model will be released under an open-source license to enable reproducibility, re-use, comparison, and modification.

1.3 Research Objectives

1. To apply and compare different machine-learning algorithms to find the best model or the best combination of models.

2. To deploy the best predictive model so anyone can use it to predict trip duration.

1.4 Research Questions

1. What are the most effective machine-learning models that are suitable to solve the research problem and that yield the best performance?

2. How to make the best model open-source and available for everyone to use for accurate taxi trip-time prediction?

1.5 Expected Outcome

1. Machine learning models that are able to accurately predict the trip duration of taxi trips in New York City.

2. The model will be deployed online in order to allow anyone to use it using a web interface.

1.6 Scope of the Study

The study will be based on New York City data. So the resulting model will work initially to predict trip duration in New York City. However, when similar data are available for other cities, the resulting model can be easily adapted to predict trip durations in those cities.

1.7 Significance of the Research

By building a machine-learning model that is able to accurately predict taxi trip duration, by releasing the code used under an open source license, and by deploying the resulting

model on the Internet, we offer an easy and free way for any transportation company or organization that wants to leverage big data to enhance their products and services.

The accurate predictions of the model will help small transportation companies in implementing predictive services to better manage their taxicabs and to provide their users with improved experiences. Being open source, the code used to build the predictive model can be easily compared to other approaches and can be improved and customized for any particular usage.

1.8 Organization of this Document

In this chpater, we introduced our work and provided preliminary information on the researched topic. In chapter 2, we will present material from the literature that provides background understanding of the study topic and shows works related to our study. In chapter 3, a proper description will be provided on the methodologies, data sets, and tools that will be used in this study. Then in chapter 4, we will describe the data-preprocessing techniques utilized to prepare the data for modeling and we will show the results of exploratory data analysis on the data and discuss those results to understand the data patterns and the relationships between its variables. After that in chapter 5, we explain how the machine-learning model is built based on principles that ensure a robust modeling process; we also state the modeling results, interpret them, and discuss their implications. Then in chapter 6, the model-deployment process is described. And finally in chapter 7, the conclusions of the project, the limitations, and future work are all discussed.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

In this chapter, we discuss previous related work from the literature where researchers have worked on topics similar to the topic of this study. For each study we review, we look at the methodology and the approach followed in the study; we also see the datasets and the model used to achieve the study objectives.

2.2 Related Work

In a study done by Li, Gunopulos, Lu, and Guibas (2017), dynamic traffic data was used for travel-time prediction. The authors chose to use dynamic data because travel time can vary significantly from hour to hour and from day to day due to different factors.

Unlike many other attempts for travel-time prediction which make use of thousands of GPS-equipped vehicles, the authors of this study try to implement a solution using GPS trajectories but with a small number of vehicles. A data collected in a project called Gotcha is used. This implementation has some difficulties since not all roads have GPS-trajectory data at all times, which creates sparsity in the data. Path-based approach was used to deal with these difficulties; in this appraoach, the route is divided into multiple parts then the travel-time for each part is predicted as will be explained below.

The framework implemented in this study is summarized in Figure 2.1.

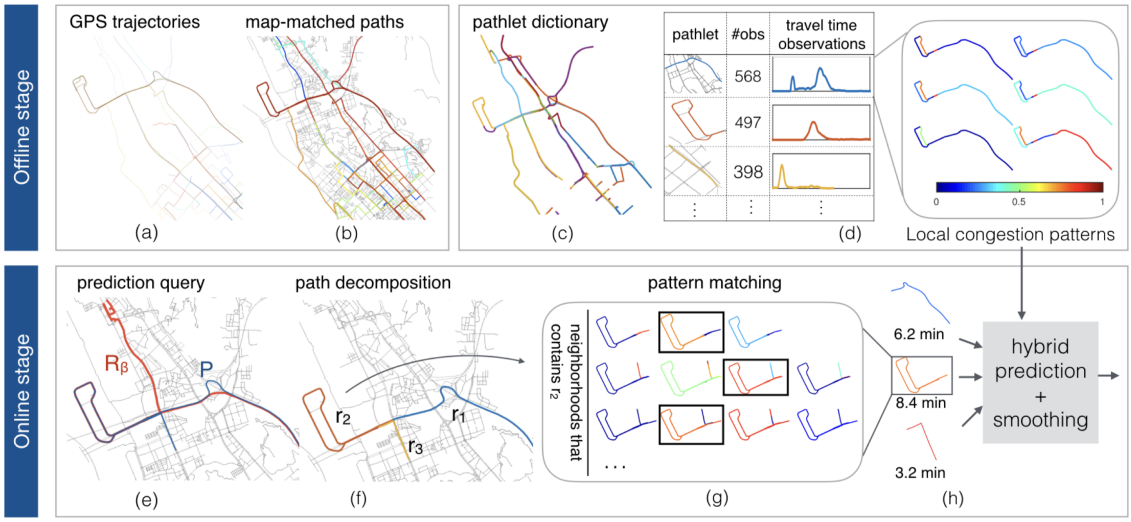

Figure 2.1: A visualization of the framework implemented by Li et al (2017). Reprinted from 'Urban travel time prediction using a small number of gps floating cars' by Y. Li, D. Gunopulos, C. Lu, and L. Guibas, 2017, *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.*

First, in steps a and b, the trajectory data gets processed and the map-matched paths are computed. Then in steps c and d, a dictionary of popular paths (pathlets) is created. Part d of the figure shows six patterns of a pathlet that are colored based on the congestion level where red means high-degree congestion and blue indicates the non-existence of congestion. Finally, in steps e through h, when the system receives a new input path, the path is divided into parts as shown in part f of the figure. Then each part is matched against the most recent congestion patterns; for example, as shown in part g, the r_2 part of the input path is matched with three patterns enclosed by black boxes; these patterns are used to calculate the predicted travel time of r_2. If no recent congestion patterns are found, the predicted travel time is calculated using historical data based on the time of r_2. Similar process is followed for r_1 and r_3 and then the final prediction is computed by combining the predicted values of the path parts and applying some optimizations (hybrid prediction and smoothing).

To test the implemented system, GPS trajectories of Shenzhen city in China were used. These trajectories were obtained by deploying 10-15 taxis for 11 months. For the test, the map of Shenzhen were divided into four regions, and for each region, 300 trajectories were selected as test data and the other trajectories as training data. Table 2.1 shows the test results using Mean Absolute Error (MAE) and Mean Relative Error metrics. Pre-

dictPM++ refers to the optimized model of the study.

Table 2.1: Test results of the model proposed by Li et al (2017)

Region ID	ρ^R	Baseline MRE	Baseline MAE	PredictPM++ MRE	PredictPM++ MAE	% improved
1	0.218	0.213	6.739	**0.148**	4.700	30.3%
2	0.124	0.155	2.959	0.151	2.891	2.43%
3	0.556	0.198	7.357	0.185	6.654	6.33%
4	0.628	0.238	8.472	0.201	7.310	15.36%
Mean	0.381	0.201	6.382	0.171	5.389	13.60%

Note. Reprinted from 'Urban travel time prediction using a small number of gps floating cars' by Y. Li, D. Gunopulos, C. Lu, and L. Guibas, 2017, *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.*

We can see that the average MAE for the optimized model is 5.389. The authors says that the duration of the trips whose trajectories were obtained varies between 10 and 75 minutes. This means that on average, the optimized model prediction has an error of ~5 minutes for trips of duration 10–75 minutes. The authors states that the average query processing time of the proposed model is 0.7 seconds.

In a study by Jindal, Chen, Nokleby, and Ye (2017), a machine learning model was built to predict the travel time and the travel distance of taxi trips. The model consists of two deep neural networks (DNN): the first DNN predicts the travel distance and then this prediction combined with other data form the input to the second DNN which predicts the travel time of the trip. Each DNN consists of a neural network with three layers. Stochastic gradient descent algorithm was used to train both DNNs.

The dataset used in this study is a dataset released by New York City Taxi & Limousine Commission. This dataset contains the following information for each taxi trip: the origin coordinates (longitude and latitude), the destination coordinates (longitude and latitude too), pickup date and time, drop-off date and time, trip duration, trip distance, etc.

As described above, the data used in this study includes location (langitude and latitude) data which was acquired with GPS technologies. GPS data is prone to some errors due to atmospheric conditions, multipath effect, clock errors, and also due to NYC tall buildings and crowded regions. To reduce the effects of these errors, the location data was discretized into 2-dimensional square cells. Time of day was also discretized into 1-dimensional cells.

The model proposed in this study which consists of two DNNs as described earlier was evaluated and compared to many other models including a linear-regression model that predicts the travel time of the trip, a linear-regression model that predicts the distance of the trip, a DNN model that predicts the travel time of the trip, and a DNN model that predicts the distance of the trip. Figure 2.2 shows the latter two models on the right side and the proposed model on the left side.

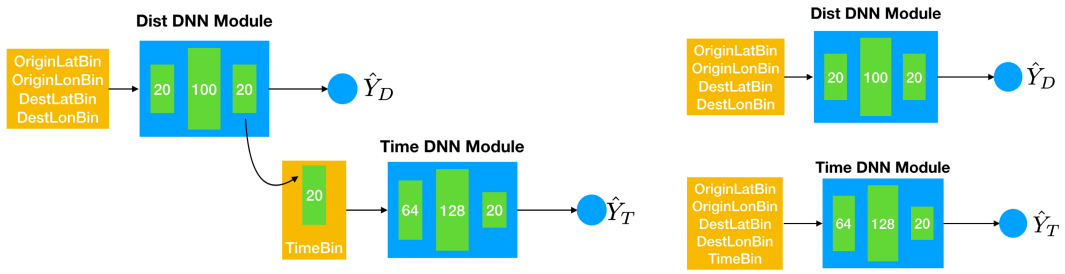

Figure 2.2: The model proposed by Jindal et al (2017) appears on the left of the figure. On the right appear two other models used for comparison. Reprinted from 'A unified neural network approach for estimating travel time and distance for a taxi trip' by I. Jindal, X. Chen, M. Nokleby and J. Ye, 2017, *arXiv preprint.*

To evaluate model performances, multiple metrics were used: Mean Absolute Error (MAE), Mean Relative Error, Median Absolute Error, Median Relative Error, and the coefficient of determination. Also, the dataset was divided into a training and testing parts with a ratio of 80:20. The evaluation results are shown in Table 2.2 and Table 2.3. Table 2.2 shows the performance comparison for travel time estimation in seconds while Table 2.3 shows the performance comparison for travel distance estimation in miles. ST-NN in the tables refers to the proposed model (shown on the left side of Figure 2.2); LRT and LRD refer to the linear-regression models; and TimeNN and DistNN refer to the DNN models shown on the right side of Figure 2.2.

Table 2.2: Performance comparison between the model proposed by Jindal et al (2017) and other models for travel time estimation in seconds.

	R^2 Coefficient	MAE	MRE	MedAE	MedRE
LRT	-1.84	724.14	1.01	638.52	1.10
TimeNN	0.713	158.29	0.221	100.242	0.182
ST-NN	0.75	145.9	0.20	91.48	0.16

Note. Reprinted from 'A unified neural network approach for estimating travel time and distance for a taxi trip' by I. Jindal, X. Chen, M. Nokleby and J. Ye, 2017, *arXiv preprint.*

Table 2.3: Performance comparison between the model proposed by Jindal et al (2017) and other models for travel distance estimation in miles.

	R^2 Coefficient	MAE	MRE	MedAE	MedRE
LRD	-0.397	3.109	1.045	2.549	1.224
DistNN	0.95	0.21	0.07	0.077	0.0418
ST-NN	0.943	0.27	0.09	0.112	0.06

Note. Reprinted from 'A unified neural network approach for estimating travel time and distance for a taxi trip' by I. Jindal, X. Chen, M. Nokleby and J. Ye, 2017, *arXiv preprint.*

As can be seen in the tables, the proposed model outperforms other models in travel time estimations. However, for travel distance, the proposed model is better than the regression model but slightly worse than DistNN model. The authors explain this by stating that the ST-NN model compromises in performance for the estimation of travel distance in order to achieve better performance for the estimation of travel time.

The proposed model in the study has some limitations: as the travel time increases the model becomes less accurate in predicting the travel time correctly; the model generates predictions with MAE error of 8-10 minutes for long trips. Moreover, the model performs better when predicting travel time during weekdays; on weekends, in the later periods of the day, the model becomes less able to capture the temporal patterns.

In another study by Gui and Yu (2014), a machine-learning model named selective forgetting extreme learning machine—which was proposed by Zhang and Wang (2011)—was used to predict trip travel time based on historical trip data. The authors recognize that there are different factors that can affect travel time such as weather and time of the day. However, they assume that this kind of information is implied in the historical trip data; they also focus on short-term travel time forecasting only.

The data used by the model is extracted from taxi trace data. A taxi's trace data is chronological data records sampled at a fixed time interval. Each record contains the taxi ID which identifies the taxi, GPS coordinates of the taxi at the sampling time, speed of the taxi at the sampling time, orientation of the taxi at the sampling time, the meter state which reflects whether there is a passenger in the taxi or not at the sampling time, and the sampling time.

The data used with the model was extracted from this trace data. Each instance in the extracted data describes a taxi trip with the following data: the start region of the trip, the

destination region, the start time of the trip, and the duration of the trip. The start time of the trip was discretized into one-hour intervals. For example, 08:10 a.m. and 08:20 a.m. were both represented as 08:00 a.m.

The model passes through two calibration phases. In the offline phase, historical trip data are utilized to construct the initial model. Then, in the online phase, new coming trip data is used to modify the initial model to make it able to respond to the dynamics of the traffic.

The dataset used in this study contains GPS trajectory data obtained from more than 30,000 taxis in Beijing, China. The data was collected for one month (30 days). The data of the first 20 days was used as training dataset for the offline phase; the data of the remaining 10 days was used for the online calibration phase.

Because the data doesn't contain instances for each origin-destination pair at all times, the authors chose origin-destination pairs that were traversed by a specified minimum number of taxis to make sure that enough data is available for the experiment. The authors report that their model has a maximum of 19.48% percentage error.

In a study performed by Singh, Wu, Xiang, and Krishnaswamy (2015), multiple techniques were combined to predict the travel time of a taxi trip. The authors base their study on the observation that most trips have three segments: the first is from the starting location to the highway; the second is on the highway; and the third is from the highway to the end location.

The model proposed in the study consists of two parts: finding similar trips and finding frequent sub-routes using road network data.

To train the model, two stages were implemented: finding similar trips and sub-routes clustering. First, for finding similar trips, similarity was based on spatial and temporal factors: two trips were considered similar if they started and ended at adjacent locations and if they had similar traffic conditions. So to cluster trips in similar groups, spatial clustering was implemented firstly to find trips that are spatially similar. Then temporal clustering was applied to each of the resulting spatial clusters to find trips that had similar traffic conditions within each spatial cluster. This was done by dividing the day into 48

parts of 30 minutes each then representing each part with the average trip time for trips that took place during that part. Then two parts are considered similar regarding traffic conditions if their average trip times are close.

Second, for sub-routes clustering, the route of each trip is estimated using OpenStreetMap which is an open-source world map. After that, the trips of each spatial-temporal cluster were clustered to obtain the sub-routes among them. Then, each trip is mapped to these sub-routes such that each trip can be represented using the three segments mentioned above: a segment from the start point to highway, a segment along the highway, and a segment from highway to the end point. A Support Vector Regression (SVR) model was trained on these segments of each trip to predict the duration of the trip.

MiniBatchKMeans algorithm which was proposed by Sculley (2010) was chosen in this study for clustering because it can scale to deal with the big amount of data used in the study. It can be used with a large number of clusters too. And it was chosen by the authors also because they needed an algorithm that can be used for prediction to know the cluster of new query trips.

Trip-time Prediction of new trips happens as follows: the model finds the spatial-temporal cluster to which this new trip belongs. Then, the route of this trip is calculated using OpenStreetMap. After that, the trip duration is predicted using the sub-routes and the SVR model that corresponds to the spatial-temporal cluster of the trip.

The dataset used in the study contained 12+ million trips in Singapore. The data was collected over 30 days. For each trip, the dataset specifies the origin, destination, starting time, and end time. 100,000 trips from the dataset were used for testing and the others for training. RMSE metric was used to measure the performance of the model.

Figure 2.3 shows the average RMSE for each day of the week (on the left side) and for each 30-minutes period of the day (on the right side). It can be seen that model performance varies considerably with the day of the week and the time of the day. The authors stated that their model was not able to capture trip duration for trips on Friday especially Friday's evening.

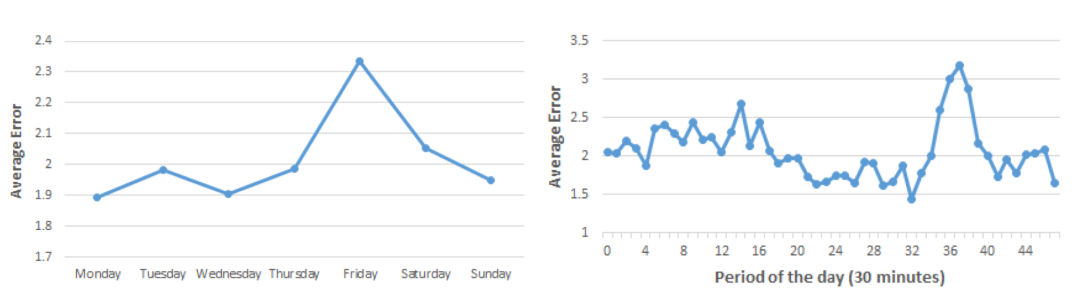

Figure 2.3: Average RMSE for each day of the week (on the left side) and for each 30-minutes period of the day (on the right side) for the model proposed by Singh et al (2015). Reprinted from 'Taxi trip time prediction using similar trips and road network data' by A. D. Singh, W. Wu, S. Xiang, and S. Krishnaswamy, 2015, *2015 IEEE International Conference on Big Data (Big Data)*.

In another study conducted by Lam, Diaz-Aviles, Pascale, Gkoufas, and Chen (2015), the authors presents machine-learning models to predict the travel time and destination of taxi trips. We will be interested in their work on travel time prediction. The study aimed to improve taxi dispatch system by predicting travel time and destination of taxi trips based on data describing the initial trajectories of these trips.

The dataset used in the study contains information on ~1.7 million taxi trips performed by 422 taxis in Porto, Portugal. For each trip, the dataset specifies the taxi ID, origin call ID which is a unique identifier for the phone number which was used to demand the taxi service, starting time of the trip, a list containing GPS coordinates for each 15 seconds of the trip, and other information.

Many features were extracted from the dataset to feed the model. Features were extracted using trip matching (finding similar trips): for a given trip, a group of 10 trips that are most similar in their trajectories to the trajectory of the given trip were used to extract features for that trip. Some of the extracted features are Haversine distance between the given trip and the similar trips and features related to kernel regression (which is a smooth version of K-Nearest Neighbors regression). Other features were extracted from the trajectory of the given trip like: distance traveled, direction (moving outside or inside the city), first and last GPS locations, average speed and acceleration for the last 10, 20, 50, 100, and 200 meters, and other features.

To train their model, the authors created a training set containing 13301 trips out of the 1.7 million trips in the original dataset using the feature extraction methodology described

above. Many models were proposed and evaluated in the study: they were ensemble models consisting of many models: Gradient Boosted Regression Trees, Random Forest regressor, and Extremely Randomized Trees regressor. The ensemble method used was stacking. On a test set that the models had never seen, the best model achieved a Root Mean Squared Logarithmic Error (RMSLE) of 0.51327.

In a study by Zhan, Hasan, Ukkusuri, and Kamga (2013), the authors present a model to estimate average travel time in an hourly fashion using big amounts of partial data.

The data used in the study was collected by New York City Taxi and Limousine Commission. It describes a large number of taxi trips in New York city. For each trip, there is information on the starting and ending locations, distance travelled, trip time, and fare. The data doesn't provide GPS trajectory data for the whole trip; it just provides the starting (origin) and ending (destination) locations.

For building the model, the authors start by processing the data. Because the tall buildings in New York City (NYC), there were some errors in the origin and destination GPS locations of the trips. To solve this problem, data mapping was used to modify the origin and destination locations such that the new locations are on actual roads in NYC. Moreover, given the origin and destination of the large number of trips in the dataset, a set of possible paths for taxi trips is constructed; the set consists of paths in NYC that a trip might have followed to travel from its origin point to its destination point. The shortest 20 paths were selected for each origin-destination pair using Yen's k-Shortest Path algorithm (Yen, 1971).

To estimate the travel time of a trip, the travel time of each possible path of the trip is multiplied by the probability of the driver choosing that path, then the results of these multiplications are summed to represent the estimated travel time of the trip. Thus, travel time estimation becomes the estimation of path travel times that minimize the least square error between the expected and observed (actual) travel times. Multinomial logistic regression (MNL) model was used to calculate the probability of a driver choosing a certain path from the set of possible paths. The MNL model takes into account trip time and distance to estimate the probability of choosing a certain path. The MNL model is used in the study with a limitation: it is used with the assumption that drivers seek to minimize

the travel time and distance in their trips which allow them to take more trips and generate more revenue.

A modified version of Levenberg-Marquardt (LM) optimization method [Nocedal and Wright, 2006; Fletcher, 1971] was used to solve the non-linear least square problem of the study. LM can be used to solve least square curve-fitting problems and it outperforms the simple gradient descent method according to the authors.

A rectangular area of 1370 × 1600 m in Midtown Manhattan was selected as the study region. The network of the study region was also extracted; it contains 311 road segments. All records from the original dataset that fall inside the study region were selected and split in hourly segments in order to be used to estimate hourly average trip time.

The authors used Matlab with parallel computing to implement their algorithms. They stated that the process mentioned earlier of computing the 20 shortest paths for each origin-destination pair takes a considerable amount of time. They also say that the LM method takes 15 minutes on Intel i7 processor to solve the optimization problem.

To test their model, the authors chose a week's data from NYC taxi dataset. Travel time was predicted for four hourly periods: 9:00–10:00, 13:00–14:00, 19:00–20:00, and 21:00–22:00. Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE) were used to evaluate the model. The MAPE scores for the four periods and the seven days of the week is shown in Table 2.4.

Table 2.4: The MAPE scores of the model proposed by Zhan et al (2013) for four periods in the seven days of the week

	9:00–10:00	13:00–14:00	19:00–20:00	21:00–22:00
Monday	29.51	24.22	26.27	21.87
Tuesday	29.63	25.59	23.33	22.20
Wednesday	41.32	34.97	28.73	24.40
Thursday	27.28	27.92	28.54	21.05
Friday	27.76	27.04	25.17	22.26
Saturday	16.84	24.58	27.14	21.61
Sunday	25.44	23.70	22.72	19.87

In another study by Nakata and Takeuchi (2004), the authors propose a model that is able

to predict travel time of a vehicle on a section of a road (link) taking into account traffic conditions. To get total travel time, travel times corresponding to all sections of the road can be added together.

The data used in the study is probe-car data which was collected in Nagoya, Japan in 2002 using around 1500 taxis. Probe-car is a car equipped with sensors used to collect data. Data was collected from the probe-cars periodically at a specified interval. Each record of the data contains car location, the state of some devices in the car like the brake and the wiper, and the time. Using this probe-car data, datasets that describe trip data for each road section (link) were created. Each row in this dataset specifies the car ID, the time at which the car entered the link, the time at which the car exited the link, and the difference between those two times which represent the travel time of the car on the link. After that, these datasets - that each one of them contains trip data of a certain link - were used to create a time series for each link. The time-series contains aggregated trip data for each 15 minutes of the day. Each record in the time series specifies the day of the week, a time index which is a number between 0 and 95 where 0 represents the first 15 minutes of the day (0:00–0:15) and 95 represents the last 15 minutes of the day (23:45–0:00), average travel time, variance, and the number of trips. The resulting time series are called probe time series.

A probe time series was created for 10 links in Negoya from 12/January to 31/March with removing data of Saturdays and Sundays. As a result, each one of the 10 links had a probe time series with ~5000 records.

Two models were used for travel time prediction in this study: Auto Regression (AR) model and state space model. AR model assumes that the data is stationary. However, traffic data is not stationary because it contains periodic sequences with 7 days and 24 hours. To solve this issue, seasonal adjustment was applied using travel time table. Moreover, to improve travel time prediction of a certain link, travel times of other links were used as explanatory variables (features). Minimum Description Length method was used to select the explanatory variables that are effective for prediction.

The second model, state space mode, can handle non-stationary data without problems. It decomposes time-series data into the following components: a trend component, a

stationary component, a seasonal component, and an observation noise component.

To evaluate the models, cross-validation technique was employed on the probe time series described earlier. The metrics used were Mean Squared Error (MSE) and its square root (RMSE). Table 2.5 shows MSE scores for links 103, 104, 105, and 106 using AR model with the explanatory variables mentioned above. Table 2.6 shows the MSE scores for all links using the state space model but with each record representing 60 minutes instead of 15 minutes; this was done to reduce computational cost.

Table 2.5: MSE scores of the AR model with the explanatory variables proposed by Nakata and Takeuchi (2004)

Link	103	104	105	106
MSE	40.9	73.1	145	212

Table 2.6: MSE scores of the state space model proposed by Nakata and Takeuchi (2004)

Link	101	102	103	104	105	106	107	108	109	110
MSE	154	105	25.9	49.2	92.2	141	148	104	138	88.6

2.3 Conclusion

In this chapter, we looked at different approaches to the trip-duration-prediction problem; we saw that many different datasets and many diverse models were used to achieve the prediction objectives. The information obtained in this chapter will be utilized in the following chapters when we design and build our predictive model.

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

In this chapter, the main data to be used in this study is introduced and defined . Also, the research design and process are broadly explained. In addition to that, the tools to be used in this study will be listed and defined.

3.2 Datasets

In this study, we will use multiple datasets as follows: we will mainly use taxi data provided by NYC Taxi and Limousine Commission (TLC). This data describes taxi trips that took place in New York City. Table 3.1 shows the information provided by the dataset for each taxi trip based on the data documentation provided by NYC Taxi and Limousine Commission (NYC Taxi and Limousine Commission, 2018).

Table 3.1: The description of data fields of taxi-trips data set

Data Field	Description
VendorID	The identification code of TPEP[1] provider that supplied the information of this trip
tpep_pickup_datetime	The date and time when the taxi meter was started
tpep_dropoff_datetime	The date and time when the taxi meter was stopped
Passenger_count	The number of passengers in this trip
Trip_distance	The distance travelled in this trip in miles
PULocationID	The ID of the zone in New York City where the meter was started
DOLocationID	The ID of the zone in New York City where the meter was stopped
RateCodeID	The rate code used for the trip where 1 denotes Standard rate, 2 denotes JFK, 3 denotes Newark, 4 denotes Nassau or Westchester, 5 denotes Negotiated fare, and 6 denotes Group ride
Store_and_fwd_flag	When the taxi is not connected to the Internet, it stores the trip information and when it connects again, it sends this information to the vendor. In this case, this flag will have the value of "Y". Otherwise—in normal cases—the flag has the value of "N"
Payment_type	A code that specifies the method used for payment by the passenger. where 1 denotes Credit card, 2 denotes Cash, 3 denotes No charge, 4 denotes Dispute, 5 denotes Unknown, and 6 denotes Voided trip
Fare_amount	The fare of the trip which is calculated using the duration and distance of the trip
Extra	The amount of extra charges like rush-hour and overnight charges
MTA_tax	A $0.5 tax that is added automatically based on the metered rate in use
Improvement_surcharge	A $0.3 surcharge that began being collected in 2015
Tip_amount	The amount of tip paid by the passenger in case of paying with credit card only; cash tips are not included
Tolls_amount	The amount of money paid for tolls during the trip
Total_amount	The total amount that should be paid by the passenger

Taxi data is provided by TLC in monthly fashion. For each month since 2009 and until

[1]Taxi Passenger Enhancement Program

2018, TLC provides a CSV file that contains data describing trips that took place in that month. In this study, we will use trip data of March 2017 and March 2018 such that the data of March 2017 will be used to train the machine-learning model and the data of March 2018 will be used to test the performance of the model.

With taxi-trips datasets comes a small helper table that maps the location IDs in `PULocationID` and `DOLocationID` fields in taxi-trips datasets (see Table 3.1) to information about the corresponding location. For each location ID, the table specifies the name of the location zone, the name of the location borough[2], and the name of the location service-zone. In the table, there are 262 unique zones that are located inside 7 unique borough. Figure 3.1 shows a map of New York City with the zone borders on the map.

Figure 3.1: New York City map with taxi zones highlighted. From NYC Taxi Zones by NYC OpenData, https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc.

In addition to the taxi-trips data sets, data sets that describe the weather in New York City for the taxi-trips days (days of March 2017 and March 2018) were used. The weather data is obtained from the website of National Weather Service which is a US governmental agency that provides weather data and forecasts among other related services. Table 3.2 shows the information provided by the weather data sets for each day based on the data documentation provided by National Weather Service (National Weather Service, n.d.).

[2]a town or district which is an administrative unit.

Table 3.2: The description of data fields of weather data set

Data Field	Description
Maximum Temperature	The maximum temperature of the day
Minimum Temperature	The minimum temperature of the day
Average Temperature	The average temperature of the day
Departure Temperature	The difference between the average temperature of the day and the 30-year normal temperature for that day
HDD	A measure of the required amount of heating or cooling for a building in the day taking 65 F degrees as a baseline
CDD	
Precipitation	The total precipitation of the day measured in inches taking into account all forms of precipitation. When small amounts of precipitation that are not enough to measure fall, this field shows a value of "T".
New Snow	Total snowfall in the day measured in inches
Snow Depth	Snow depth on the ground at 12:00 in UTC time zone measured in inches

Thus, four datasets will be used in this project: two for training the machine-learning model and two for testing that model. For training, the first dataset to be used is the main dataset which is the dataset that describes taxi trips in March 2017; the second dataset is the dataset that describes the weather in March 2017. Similarly for testing, the first dataset is the taxi-trips dataset of March 2018 and the second dataset is the weather dataset for March 2018 as well.

3.2.1 Taking a Look at a Sample of the Datasets

Figure 3.2 shows a few records from the taxi-trips dataset of March 2017; the taxi-trips dataset of March 2018 has the same structure but with different values. Figure 3.3 shows a few records of the helper location table that comes with taxi-trips datasets. And Figure 3.4 shows a few records of the weather dataset of March 2017; weather dataset of March 2018 also looks similar and has the same structure with different values.

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID
0	2	2017-03-09 21:30:11	2017-03-09 21:44:20	1	4.06	1	N	148	48
1	2	2017-03-09 21:47:00	2017-03-09 21:58:01	1	2.73	1	N	48	107
2	2	2017-03-09 22:01:08	2017-03-09 22:11:16	1	2.27	1	N	79	162
3	2	2017-03-09 22:16:05	2017-03-10 06:26:11	1	3.86	1	N	237	41
4	2	2017-03-31 06:31:53	2017-03-31 06:41:48	1	3.45	1	N	41	162

payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount
1	14.0	0.5	0.5	3.06	0.0	0.3	18.36
2	11.5	0.5	0.5	0.00	0.0	0.3	12.80
1	10.0	0.5	0.5	2.82	0.0	0.3	14.12
1	12.0	0.5	0.5	3.99	0.0	0.3	17.29
2	12.0	0.5	0.5	0.00	0.0	0.3	13.30

Figure 3.2: A sample of the taxi-trips datasets

	LocationID	Borough	Zone	service_zone
0	1	EWR	Newark Airport	EWR
1	2	Queens	Jamaica Bay	Boro Zone
2	3	Bronx	Allerton/Pelham Gardens	Boro Zone
3	4	Manhattan	Alphabet City	Yellow Zone
4	5	Staten Island	Arden Heights	Boro Zone

Figure 3.3: A sample of the helper location table

	Date	Maximum Temperature	Minimum Temperature	Average Temperature	Departure Temperature	HDD	CDD	Precipitation	New Snow	Snow Depth
0	2017-03-01	70	54	62.0	23.9	3	0	0.12	0.0	0
1	2017-03-02	64	35	49.5	11.2	15	0	0.00	0.0	0
2	2017-03-03	39	24	31.5	-7.1	33	0	0.00	0.0	0
3	2017-03-04	30	17	23.5	-15.3	41	0	0.00	0.0	0
4	2017-03-05	37	14	25.5	-13.6	39	0	0.00	0.0	0

Figure 3.4: A sample of the weather datasets

3.2.2 Datasets Sizes

Table 3.3 shows the number of rows (records) and columns (fields or features) of the four datasets of this project.

Table 3.3: The number of rows and columns in the project datasets

Dataset	Number of rows	Number of columns
Taxi trips of March 2017	10,294,628	17
Taxi trips of March 2018	9,430,376	17
Helper location table for taxi-trips datasets	265	4
Weather of March 2017	31	10
Weather of March 2018	31	10

3.2.3 Philosophy Behind Using the Weather Datasets

The goal of using the weather datasets is to test if they help to achieve better modeling performance. In this project, we are using *daily* weather data (e.g. average temperature in the day, total precipitation in the day, etc.) as shown in 3.2. This daily data becomes available at the end of each day. We are now able to use this data because we will train and test our machine-learning models using past data (data of March 2017 and March 2018). If we notice that the weather data offer performance improvements, we can utilize weather data to predict duration for future trips that haven't started yet as follows: when predicting the duration for a trip, we can use the current weather data by retrieving this data from weather APIs which are provided by many organizations like OpenWeather (https://openweathermap.org/api) and Dark Sky (https://darksky.net/dev).

3.2.4 New Limitation in the Taxi-Trips Datasets

As mentioned above, the main datasets in this project are taxi-trips datasets released by NYC Taxi and Limousine Commission (TLC). Previously, these datasets specified the exact pickup and drop-off locations using longitude and latitude coordinates. Recently, however, TLC decided to stop collecting and sharing those exact location coordinates for security and privacy reasons. Instead, they provide the zone of pickup and drop-off locations where they divided New York City into 265 zones.

This recent change might limit the accuracy of predictions produced by the machine-learning model of this study since a part of the data that describes the taxi trips is now missing. It is worth mentioning that previous research done using the datasets provided by TLC used the datasets that contain the exact location coordinates, which might give them advantage in model performance.

3.2.5 Datasets Naming in Programming Code

In the Python code used in this project, we will refer to the datasets of this project as described in Table 3.4 that shows the variable name in the code for each dataset. Each one of the variables listed in the table is a Pandas DataFrame. Pandas DataFrame is a

two-dimensional data structure like a spreadsheet or SQL table where columns can have different data types.

Table 3.4: Code variable names of the datasets used in the project

Dataset	Variable Name in Code
Taxi-trips data for March 2017 which will be used to train the model	data
Taxi-trips data for March 2018 which will be used to test the model	test_data
Helper location table for taxi-trips datasets	zones_lookup_table
Weather data for March 2017 which will be used to train the model	weather_data
Weather data for March 2018 which will be used to test the model	weather_test_data

3.3 Research Design

In this research study, the Cross-Industry Standard Process for Data Mining (CRISP-DM) will be used as a guide to apply the data science process using appropriate data in order to build and deploy the desired predictive model. Figure 3.5 shows a diagram describing the CRISP-DM process.

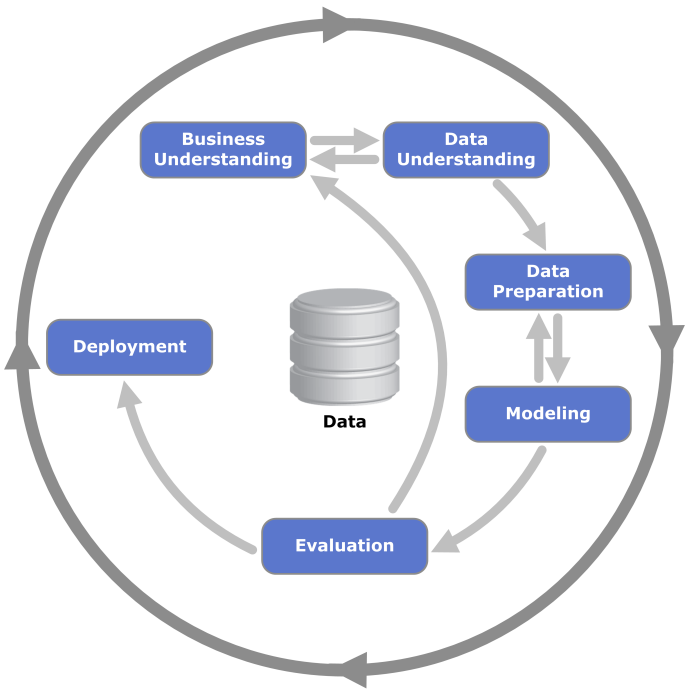

Figure 3.5: Cross-Industry Standard Process for Data Mining (CRISP-DM). From Wikimedia. Retrieved from https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png. Copyright 2012 by Kenneth Jensen released under the Creative Commons Attribution-Share Alike 3.0 Unported license.

In the following paragraphs, we will generally explain how we will follow the CRISP-DM stages in this study:

Business understanding In this stage, we define the business problem to be solved, what business benefits will the completion of this project bring, and the significance of the project. This was done in chapter 1 of this study.

Data understanding In this stage, we collect the research data that is required to do the project, we seek to comprehend the data in order to utilize it in the most effective way, and we explore the data using relevant and suitable visulaizations to get a better understanding of the data variables and the relationships between them. This stage is important because it enables us to achive maximum utilization of the available data.

Data preparation In this stage, data is prepared for modeling. Preparation includes data cleaning where the problems of missing values, data inconsistency, and outliers are fixed. Preparation also includes data transformation where data variables are

transformed in a way that makes the variables more suitable for machine-learning modeling; many types of transformations can be applied according to the nature of the variable and the modeling requirements: binning/bucketing, data normalization, data scaling, log transformation, encoding of categorical variables, etc. Moreover, data preparation covers a component that can improve model performance substantially which is feature engineering; in feature engineering, new features are created based on existing features in a way that allows the predictive model to exploit the data more in order to extract more hidden patterns from the data. In this stage also, the data is divided into training set and testing set in order to build and test the model in the next stage.

Modeling In this stage, the machine-learning model is built using the best combination of hyperparameters; to find the best set of hyperparameters, multiple hyperparameter-optimization techniques can be applied including grid search, random search, and Bayesian optimization. In this stage, the model is trained using the training set produced in the previous stage. Two approaches can be followed to train the model: 1) dividing the training set into two parts, one for training and the other for validation. 2) using cross-validation technique.

Evaluation The model produced in the previous stage is evaluated and tested in this stage. To evaluate the model, we generate predictions using the model on the testing set that was produced in data-preparation stage and then, we compare these predictions against the actual values to determine the goodness and the performance of the model. Many metrics can be used in this comparison process. Some of the well known metrics in regression problems are mean absolute error (MAE), root-mean-square error (RMSE), and R^2. In this study, we will focus on MAE metric.

Deployment After we finish building and evaluating the model, we move to deploy the model in this stage. The resulting model is deployed on the Internet where a web interface is used to access the model and generate predictions for data entered by users. In this stage also, the code used in all previous stages is prepared and published under an open source license to allow others to use, test, and modify it; this

step is important to achieve research reproducibility where other researchers can easily verify the results claimed in this study.

3.4 Tools

In order to achieve the objectives of this study and obtain results, we need the right set of tools. First, regarding the programming language, we will use Python. Python is one of the leading—if not the leading— open-source programming languages in the data science field. The tools that will be used in this study include:

- Pandas: a Python library for data analysis and data manipulation. The operations are performed on dataframes, the main Pandas data structure.
- Jupyter Notebook: an interactive-computing tool that allows writing and running Python code while showing the output of the code in the same file. All the code used in this project is available as Jupyter notebooks.
- Numpy: a Python library that provides many mathematical and linear-algebra operations on multi-dimensional arrays.
- Matplotlib: a Python library for plotting and visualization.
- Seaborn: a Python library for plotting and visualization that provides high-level commands to draw statistical graphics.
- Scikit-learn: a Python library for machine learning. It provides implementations of many machine-learning algorithms and related functions.
- XGBoost: a library that implement an efficient gradient-boosting machine-learning algorithm. It provides an implementation in Python and other languages as well.
- LightGBM: a gradient-boosting library from Microsoft that uses fast tree-based approach. It provides Python implementation too.
- CatBoost: a gradient boosting on decision trees library from Yandex. It offers Python implementation also.

- Google Cloud Platform: a platform for cloud computing services provided by Google. The services of interest are AI Platform Notebooks and App Engine.
- Flask: a web-application framework that can be used to deploy the model of this study.

3.5 Evaluation Metric

As mentioned earlier in 3.3, the metric that will be used in this study to evaluate the machine-learning models is the mean-absolute-error (MAE) metric. Given a vector y which contains the actual true values and a vector $\hat{y}$ that contains the predicted values, MAE can be calculated using the following formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

where n is the number of items in each of y and $\hat{y}$. MAE metric is easy to interpret as the average error the model makes in doing predictions; it has the same unit (e.g. minutes, kg, etc.) as the data.

3.6 Conclusion

In this chapter, we defined the main data in this study with information about each field in the data. After than, we introduced the data-science process that we will follow in this study with a general description of each of its stages. Finally, we provided a list of the tools that will be used in the implementation of this study.

CHAPTER 4: DATA PREPARATION AND EXPLORATION

4.1 Introduction

In this chapter, we will describe the steps taken to make the data of this study ready to be used for modeling. Moreover, we will run an exploratory data analysis on the data to get a proper understanding of the data which will enable us to create new features (fields) that can enhance the performance of the machine-learning model in the modeling phase.

The data that will be prepared and explored is the data described in Research Methodology chapter, which are the taxi-trips data and the weather data.

4.2 Data Acquisition

The taxi-trips data used in this research was downloaded from the website of NYC Taxi and Limousine Commision. As mentioned earlier, we will use the data of March 2017 and March 2018 in this study. The combined size of the data is 1.74 gigabytes. The combined number of rows of the data is 19,725,004 rows.

The weather data used in this research was downloaded from National Weather Service website. It covers the days of March 2017 and March 2018. The size of this weather data is 3 kilobytes, and it contains a row for each day in the two months.

4.3 Data Cleaning

In order to have accurate results, the data we use must be complete, correct, and consistent. In other words, we must ensure that the data used is a high-quality data. The steps taken to clean the data are described below.

4.3.1 Dealing with Missing Values

To ensure that our data is complete, we must check if it contains any missing values, and if it does, we should fill in these missing values using appropriate methods.

Upon checking the number of missing values in the datasets of the study, we found that our data sets do not contain any missing value. Figure 4.1 shows the code used to check the number of missing values in the project datasets and the results of running the code.

```
data.isna().sum().sum()
```

```
0
```

```
test_data.isna().sum().sum()
```

```
0
```

```
weather_data.isna().sum().sum()
```

```
0
```

```
weather_test_data.isna().sum().sum()
```

```
0
```

Figure 4.1: Code to check the number of missing values in the project datasets and its results

4.3.2 Checking for Duplicate Records

By checking the number of duplicate records in the project datasets, it was found that the dataset of March-2017 taxi trips contains 27 duplicate records out of the total 10,294,628 records. Duplicate records should be deleted when they exist due to errors or when they represent the exact real-world objects. However, in our case, the duplicate records in the taxi-trips dataset might be describing different taxi trips that have started and ended at the same times and locations, especially that locations in the dataset are specified as zone IDs not as exact coordinates as described in 3.2.4 earlier. Hence, we will not remove these 27 duplicate records.

No duplicate records were found for the other datasets. Figure 4.2 shows the code used to check for duplicate records in the project datasets and the results of running the code.

```
data.shape[0] - data.drop_duplicates().shape[0]
```

```
27
```

```
test_data.shape[0] - test_data.drop_duplicates().shape[0]
```

```
0
```

```
weather_data.shape[0] - weather_data.drop_duplicates().shape[0]
```

```
0
```

```
weather_test_data.shape[0] - weather_test_data.drop_duplicates().shape[0]
```

```
0
```

Figure 4.2: Code to check for duplicate records in the project datasets and its results

4.3.3 Removing Incorrect and Inconsistent Data

To get accurate predictions from the machine-learning model, we need to make sure that the data we feed to the model is free of incorrect or inconsistent values. Toward that end, we will run many logical checks on our datasets to spot errors and correct them.

Removing records where pickup and drop-off times are the same or where pickup time is after drop-off time In taxi-trips datasets, when the pickup and drop-off times are identical, this means that there is something wrong with trip because it is not possible that the trip starts and ends at the same minute and the same second. Also, when the specified pickup time of a trip occurs after the drop-off time, there is clearly an error in this case because drop-off always occur after picking up the passenger.

When checking the number of records where one of the two aforementioned error cases occurs, we find that there are 10,283 records with such errors in the training taxi-trips dataset (in `data` variable), and 7,777 records with such errors in the test taxi-trips dataset (in `test_data` variable). Figure 4.3 shows the code used to check for the errors and delete the erroneous records.

```
data[data.tpep_pickup_datetime >= data.tpep_dropoff_datetime].shape[0]
```

```
10283
```

```
data = data[data.tpep_pickup_datetime < data.tpep_dropoff_datetime]
```

```
test_data[test_data.tpep_pickup_datetime >= test_data.tpep_dropoff_datetime].shape[0]
```

```
7777
```

```
test_data = test_data[test_data.tpep_pickup_datetime < test_data.tpep_dropoff_datetime]
```

Figure 4.3: Code used to check for errors in pickup and drop-off times and delete the erroneous records

Removing records where trip distance is less than or equal to zero In taxi-trips datasets, when the trip distance is less than or equal to zero, this means that this trip-record should be deleted because either the trip did not occur (when trip distance is zero for example) or trip data was entered incorrectly (when trip distance is a negative number for example).

When checking the number of records where trip distance is less than or equal to zero, we find that there are 56,387 records with such errors in the training taxi-trips dataset

(in `data` variable), and 53,407 records with such errors in the test taxi-trips dataset (in `test_data` variable). Figure 4.4 shows the code used to check for the errors and delete the erroneous records.

```
data[data.trip_distance <= 0].shape[0]
```

```
56387
```

```
data = data[data.trip_distance > 0]
```

```
test_data[test_data.trip_distance <= 0].shape[0]
```

```
53407
```

```
test_data = test_data[test_data.trip_distance > 0]
```

Figure 4.4: Code used to check for errors in trip distances and delete the erroneous records

Removing records where total charge amount of the trip is less than or equal to zero

In taxi-trips datasets, when the total amount of money charged to the passanger in a trip is less than or equal to zero, this indicates an error in the record of this trip, and this erroneous record should be deleted.

When checking the number of records where total charged amount is less than or equal to zero, we find that there are 4,203 records with such errors in the training taxi-trips dataset, and 4,452 records with such errors in the test taxi-trips dataset. Figure 4.5 shows the code used to check for the errors and delete the erroneous records.

```
data[data.total_amount <= 0].shape[0]
```

```
4203
```

```
data = data[data.total_amount > 0]
```

```
test_data[test_data.total_amount <= 0].shape[0]
```

```
4452
```

```
test_data = test_data[test_data.total_amount > 0]
```

Figure 4.5: Code used to check for errors in trip charge amount and delete the erroneous records

Removing records where passenger count is less than one In taxi-trips datasets, when the passenger count in a trip is less than one (e.g. a passenger count of 0), this indicates an error in this trip record, and this erroneous record should be deleted.

When checking the number of records where passenger count is less than one, we find that there are 66 records with such errors in the training taxi-trips dataset, and 69,135 records

with such errors in the test taxi-trips dataset. Figure 4.6 shows the code used to check for the errors and delete the erroneous records.

```
data[data.passenger_count < 1].shape[0]
```

```
66
```

```
data = data[data.passenger_count >= 1]
```

```
test_data[test_data.passenger_count < 1].shape[0]
```

```
69135
```

```
test_data = test_data[test_data.passenger_count >= 1]
```

Figure 4.6: Code used to check for errors in trip passenger-count and delete the erroneous records

The total number of removed erroneous records from taxi-trips dataset of March 2017 is 70,941 records which is 0.69% only out of the total 10,294,628 original records. And for the taxi-trips dataset of March 2018, the total number of removed erroneous records is 134,771 which is 1.43% only out of the total 9,430,376 original records.

4.3.4 Data-Type Conversion

To facilitate the analysis process, we need to make sure that each data field in the project datasets is represented in the most appropriate format. When we take a look at the data types of the fields of taxi-trips datasets and weather datasets, we see that date and date-time fields are represented as text strings instead of actual datetime items. Specifically, these fields are `tpep_pickup_datetime` and `tpep_dropoff_datetime` from taxi-trips datasets and `date` field in weather datasets. To convert these fields to datetime format, we use the Pandas function `astype()`. Figure 4.7 shows the code used in this conversion process where "M" denotes a datetime format.

```
# for taxi-trips datasets
data.tpep_pickup_datetime = data.tpep_pickup_datetime.astype('M')
data.tpep_dropoff_datetime = data.tpep_dropoff_datetime.astype('M')

test_data.tpep_pickup_datetime = test_data.tpep_pickup_datetime.astype('M')
test_data.tpep_dropoff_datetime = test_data.tpep_dropoff_datetime.astype('M')


# for weather datasets
weather_data.Date = weather_data.Date.astype('M')
weather_test_data.Date = weather_test_data.Date.astype('M')
```

Figure 4.7: Code used to convert some data fields to datetime format

Another issue regarding data types is that the `Precipitation`, `New Snow`, and `Snow`

`Depth` fields in the weather datasets are represented as text strings instead of being represented as numbers. This is because these fields contain the value "T" for some rows in the datasets. "T" in these cases means "Trace". So for example, when `Precipitation` field has the value "T" for some day, this means that the precipitation in that day was in small amounts which made it not measurable.

We want to replace "T" with a numeric value which will allow us to represent the three aforementioned fields in an appropriate numeric format. By checking the minimum values for these fields in weather datasets, we confirmed that the minimum value for all of them is 0. For `Precipitation` fields, the next minimum value after 0 is 0.01, so we will replace "T" with a tenth of this value which is 0.0001. Similarly for `New Snow` fields, the next minimum value after 0 is 0.2, so we will replace "T" with 0.002. Finally, for `Snow Depth` fields, the next minimum value after 0 is 1, so we will replace "T" with 0.01. Figure 4.8 shows the code used to replace the values and then convert the fields to float format.

```
weather_data['Precipitation'] = weather_data['Precipitation'].replace(to_replace='T', value=0.0001).astype('f')
weather_data['New Snow'] = weather_data['New Snow'].replace(to_replace='T', value=0.002).astype('f')
weather_data['Snow Depth'] = weather_data['Snow Depth'].replace(to_replace='T', value=0.01).astype('f')

weather_test_data['Precipitation'] = (
    weather_test_data['Precipitation'].replace(to_replace='T', value=0.0001).astype('f'))
weather_test_data['New Snow'] = weather_test_data['New Snow'].replace(to_replace='T', value=0.002).astype('f')
weather_test_data['Snow Depth'] = weather_test_data['Snow Depth'].replace(to_replace='T', value=0.01).astype('f')
```

Figure 4.8: Code used to replace the "T" values and convert certain data fields in weather datasets to float format

4.3.5 Unit Conversions

In taxi-trips datasets, there is the field `trip_distance` which specifies the distance travelled during the trip in miles. We want to convert the values of this field so they are in kilometers instead of miles because kilometers is a more universal unit than mile and therefore, it is easier to interpret. To do so, we multiply the values in this field with 1.609.

For the same purpose, the temperature fields in weather datasets are measured in Fahrenheit. We want to convert them to be in Celsius by applying the following formula.

$$C = \frac{F - 32}{9} \times 9$$

Also, in these datasets, the `Precipitation`, `New Snow`, and `Snow Depth` fields are mea-

sured in inches. We will convert them to be in Centimeters by multiplying their values by 2.54.

4.4 Creation of the Target Variable

The main goal of this project is to build a machine-learning model to predict taxi trip duration. This means that the target that the model will predict is the trip duration. Although trip duration is implicitly specified in taxi-trips datasets (through `tpep_pickup_datetime` and `tpep_dropoff_datetime` variables), there is no explicit variable that specifies trip duration. Thus, we need to create this variable using the two aforementioned variables. Figure 4.9 shows the code used to create the taget variable in taxi-trips datasets. The target variable, `trip_duration`, is created by calculating the difference in minutes between pickup and drop-off times.

```
data['trip_duration'] = (data.tpep_dropoff_datetime - data.tpep_pickup_datetime).dt.total_seconds() / 60

test_data['trip_duration'] = (test_data.tpep_dropoff_datetime - test_data.tpep_pickup_datetime
                              ).dt.total_seconds() / 60
```

Figure 4.9: Code used to create the target variable in taxi-trips datasets

4.5 Exploratory Data Analysis

After cleaning the training and test datasets in the previous sections, we can now explore these datasets using statistical methods and effective visualizations to get a better understanding of the data, the distribution of the data, and the relationships between different variables in the data. After doing data exploration, we can decide on applying some transformations and feature engineering on the data to improve the modeling performance. Moreover, data exploration can help us to spot errors and outliers in the data so we can fix these errors before modeling to get more accurate predictions.

4.5.1 Distribution of Trip-Duration Variable

We start by exploring the distribution of the target variable which specifies trip duration. One of the most effective ways to visualize data distribution is by using histograms. Fig-

ure 4.10 shows the histogram of `trip_duration` for training data (i.e. for March 2017 data).

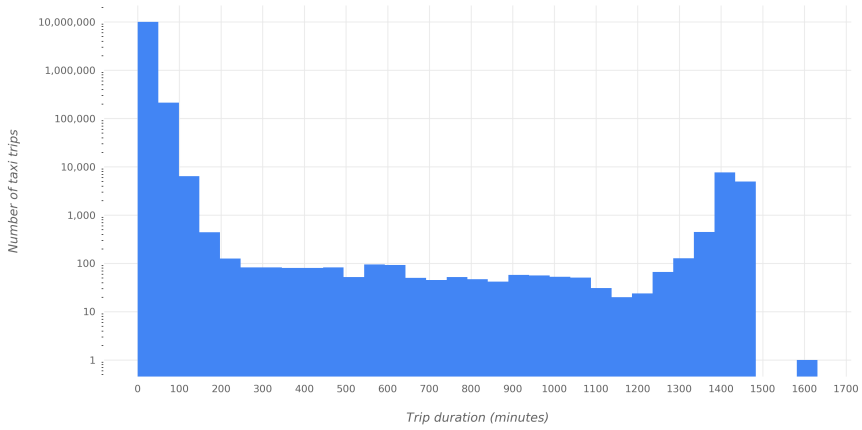

Figure 4.10: Histogram of trip_duration of training data

We can see from the figure that the vast majority of taxi trips in March 2017 had a trip duration less than 50 minutes (notice that the y axis is *log-scaled*). We can also see that there are some trips with longer trip durations but there is a single trip with a duration of ~1600 minutes. By retrieving the data of this trip, we found that the distance travelled in this trip was 26.2 km only. This clearly indicates that there is an error in this trip record. Thus, we delete this record.

Similarly, Figure 4.11 shows the histogram of trip_duration for test data (i.e. for March 2018 data).

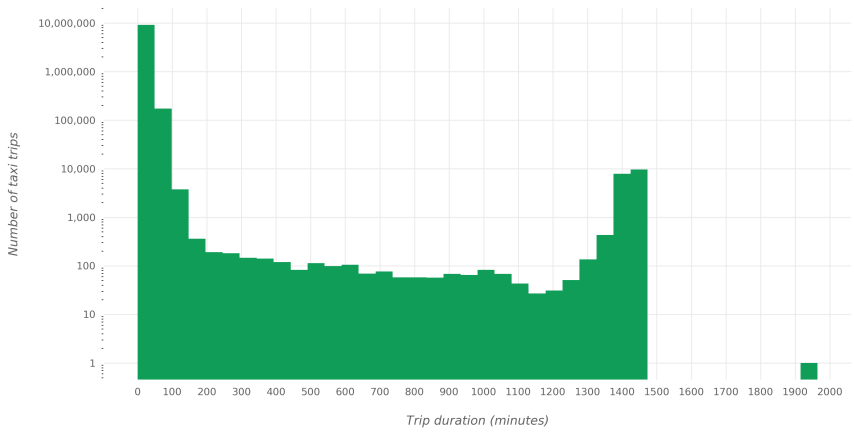

Figure 4.11: Histogram of trip_duration of test data

As with training data, it can be seen that the vast majority of trips in March 2018 had a

trip duration less than 50 minutes and that there are some trips with longer durations. We also notice a single trip in the plot with duration of ~1900 minutes. By retrieving the data of this trip, we found that the distance travelled in this trip was 3.9 km only. This clearly indicates that there is an error in this trip record. Thus, we delete this record.

4.5.2 Number of Taxi Trips Per Day of the Month

Next, we investigate the number of trips per day of the month to see if the number of trips differs significantly from day to day and to find the reason of such difference to utilize it in the modeling process. Figure 4.12 shows a plot of the number of taxi trips for each day in March 2017. The plot was produced by extracting the day component from `tpep_pickup_datetime` field, which specifies the pickup date and time of taxi trips, then counting the number of trips in each day.

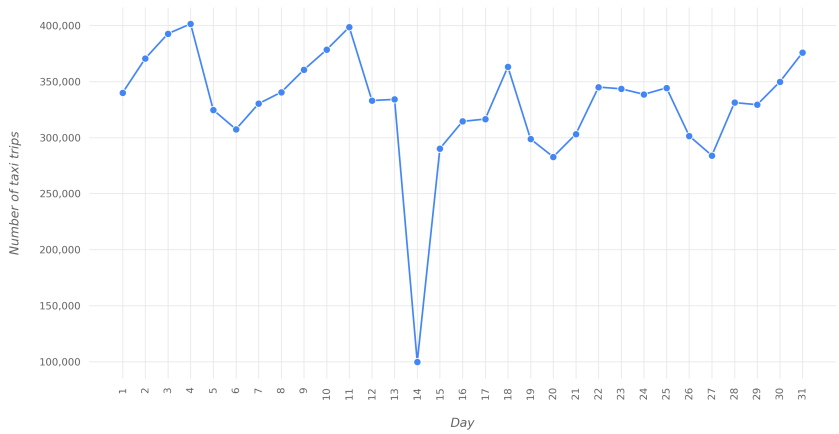

Figure 4.12: Number of trips in each day of March 2017

We can see that the number of trips approximately ranges between 300,000 and 400,000 trips per day except for 14-March where the number of trips drops sharply to 100,000 trips. To investigate the reason of this decline, we searched the weather dataset of March 2017 to retrieve the weather data of 14 March 2017 and some days before and after it to see if there were exceptional conditions in 14 March. Figure 4.13 shows the weather data for the aforementioned three days.

Date	Average Temperature	Precipitation	New Snow
2017-03-11	-5.000000	0.0000	0.000000
2017-03-12	-4.444444	0.0000	0.000000
2017-03-13	-2.222222	0.0000	0.005080
2017-03-14	-3.055556	5.0038	19.303999
2017-03-15	-5.000000	0.0000	0.000000
2017-03-16	-0.277778	0.0000	0.000000
2017-03-17	3.333333	0.0000	0.000000

Figure 4.13: Weather data for some days of March 2017

From that figure, we can see the significant difference in weather between 14 March and other days: the precipitation and snowing in 14 March occured in much greater amounts than other days. This heavy snowing is a plausible reason for the decline in number of trips on 14 March.

Similarly, Figure 4.14 shows the number of taxi trips for each day in March 2018.

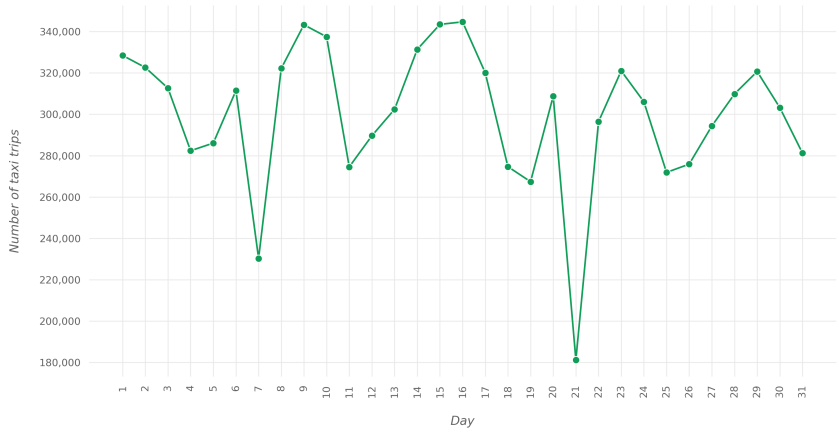

Figure 4.14: Number of trips in each day of March 2018

We can see that for most days of the month, the number of trips ranges roughly between 270,000 and 340,000 trips per day. However, we notice a decline in the number of taxi trips on the 21th of the month where the number of taxi trips drops to ~180,000 trips. Moreover, there is another less-sharp decline on the 7th of the month where the number of taxi trips is ~230,000. Like what was done above, we look at the weather conditions for these two days and the days surrounding them. Figure 4.15 shows the weather conditions for the two decline days and some days before and after them.

Date	Average Temperature	Precipitation	New Snow
2018-03-18	1.944444	0.000000	0.000000
2018-03-19	4.444444	0.000000	0.000000
2018-03-20	1.388889	0.000254	0.000000
2018-03-21	1.666667	2.514600	20.827999
2018-03-22	5.000000	0.152400	0.508000
2018-03-23	5.555556	0.000254	0.000000
2018-03-24	6.111111	0.000000	0.000000

Date	Average Temperature	Precipitation	New Snow
2018-03-04	3.888889	0.000000	0.00000
2018-03-05	3.333333	0.000000	0.00000
2018-03-06	4.166667	0.000254	0.00000
2018-03-07	2.222222	3.581400	8.12800
2018-03-08	2.222222	0.000254	0.00508
2018-03-09	2.222222	0.000000	0.00000
2018-03-10	1.944444	0.000000	0.00000

Figure 4.15: Weather data for some days of March 2018

From the figure, we can see the remarkable difference in snowing amount between 21 March and its surrounding days. For 7 March, the snowing amount is also more than its surrounding days but the difference is smaller than that of 21 March. We can see that the greater the difference in weather conditions (in this case, snowing), the greater the drop in the number of taxi trips.

The correlation we just observed between weather conditions on a day and the number of taxi trips on that day gives us an indication of the performance improvement that can be achieved by incorporating weather data in the modeling process.

4.5.3 Number of Taxi Trips Per Hour of the Day

After exploring the distribution of the number of taxi trips on month days, we now move to explore taxi trips distribution on the hours of the day: we want to see the number of trips performed during a certain hour throughout the month. For example, we want to see the number of taxi trips performed between 2:00 p.m. and 3:00 p.m. in *all* days of the month.

Figure 4.16 shows the number of trips per hour for March 2017. For example, the first point in the graph shows the number of trips performed between 00:00 (i.e. 12:00 a.m.) and 01:00 (i.e. 01:00 a.m.) over the month.

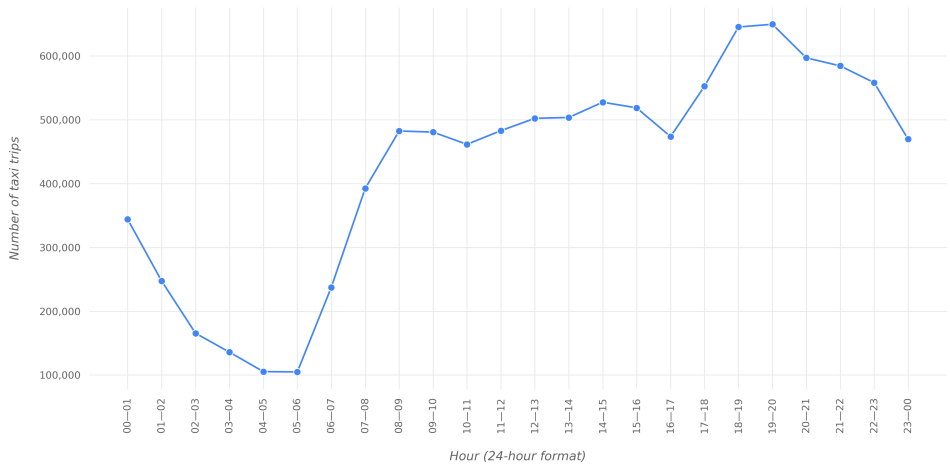

Figure 4.16: Number of taxi trips per hour in March 2017

The disparity in number of trips between hours of the day can be clearly seen in the graph. It can be seen that there are fewer trips after midnight until early morning, then starting at 6:00 a.m., the number of trips increases, and then increases even more at 6:00 p.m. To visualize these increments in the number of trips from period to period, Figure 4.17 shows the number of trips per period of the day where the day is divided into three periods with equal length (i.e. each period is an 8-hours period) starting at 12:00 a.m.

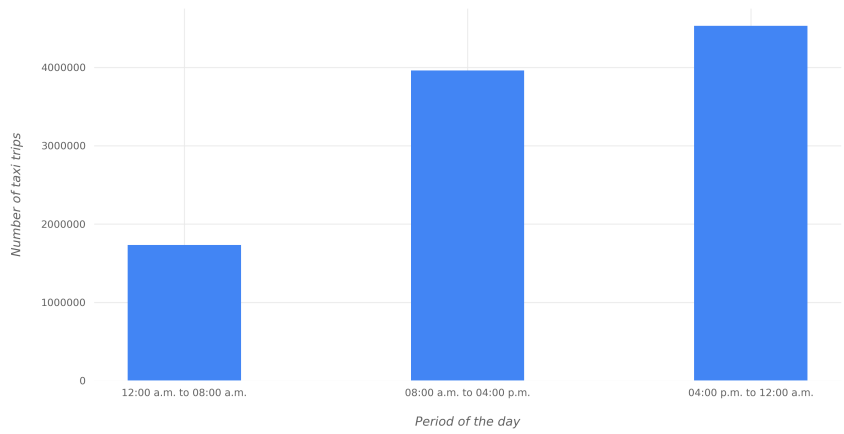

Figure 4.17: Number of taxi trips per period of the day in March 2017

Similarly for test data of March 2018, Figure 4.18 shows the number of trips per hour of the day and Figure 4.19 shows the number of trips per period of the day. It can be noticed that the number of trips also increases from period to period as time passes.

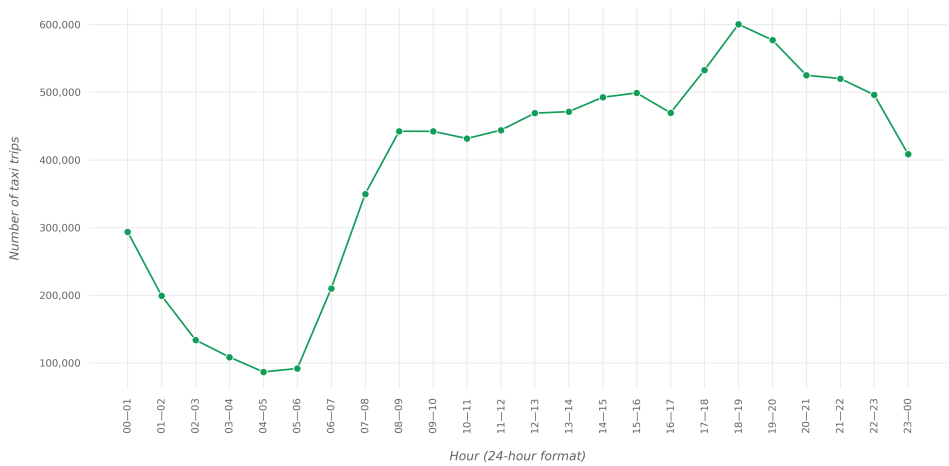

Figure 4.18: Number of taxi trips per hour in March 2018

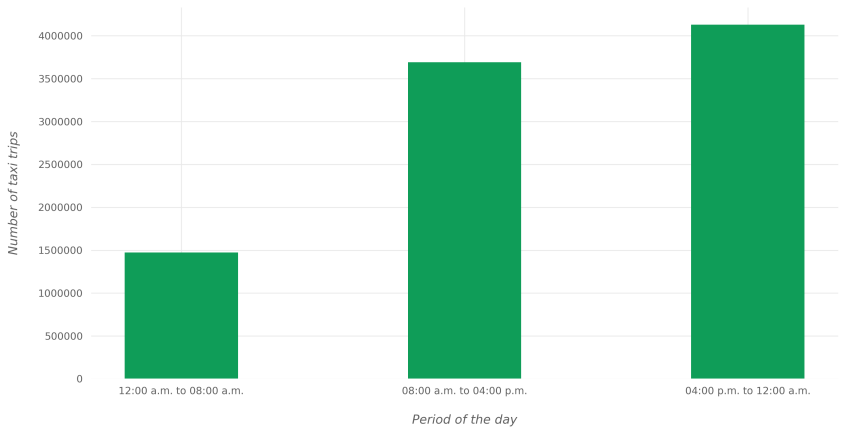

Figure 4.19: Number of taxi trips per period of the day in March 2018

By looking at the distribution of the number of trips over day hours in March of 2017 and 2018, we can see that the the time between 6:00 p.m. and 8:00 p.m. represents the peak time where more taxi trips occur than other times in the day. Additionally, we can see that the time between 4:00 a.m. and 6:00 a.m. represents a slack time where less taxi trips occur than other times of the day. Taking the time and the period of the trip into account in the modeling phase seems crucial to get high-accuracy predictions.

4.5.4 Distribution of the Number-of-Passengers Variable

To see the number of taxi trips that had one passenger, two passengers, three passengers, etc., we plot the distribution of the `passenger_count` variable as a bar plot. For

March 2017 taxi-trips data, Figure 4.20 shows this distribution, and for March 2018, the distribution is displayed in Figure 4.21.

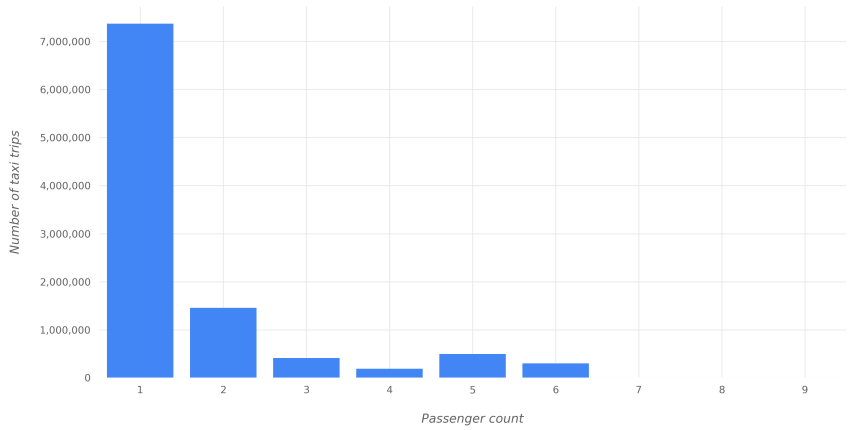

Figure 4.20: Distribution of the passenger-count variable for March-2017 data

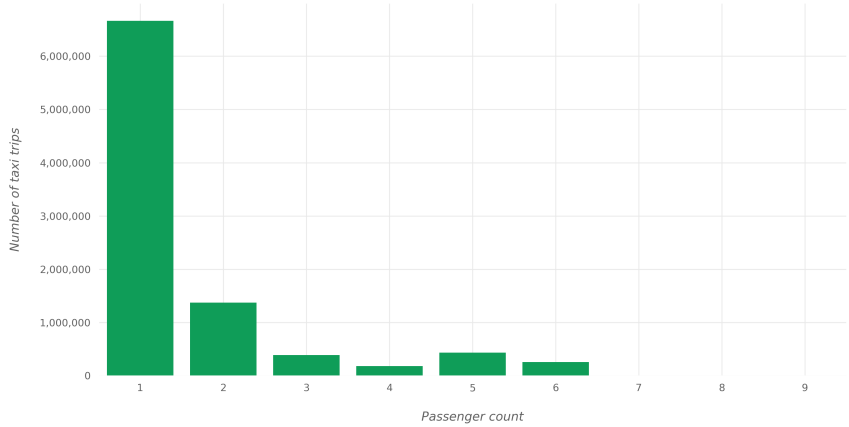

Figure 4.21: Distribution of the passenger-count variable for March-2018 data

We can see that for both months, the majority of taxi trips had only one passenger (72.9% of March-2017 trips and 71.7% of March-2018 trips) and that some trips had two passengers (14.3% of March-2017 trips and 14.8% of March-2018 trips). We notice also that few trips had three, four, five, or six passengers and very few trips had seven, eight, or nine passengers; in fact, only 0.0004% of trips in March 2017 and 0.0003% of trips in March 2018 had seven, eight, or nine passengers.

4.5.5 Distribution of Trip-Distance Variable

Trip distance is the total distance travelled by the taxi in the trip in kilometers, starting at the pickup location and ending at the drop-off location. Figure 4.22 shows the distribution of trip distances of March 2017; the distribution is displayed using a histogram plot (on the left side of the figure) and using a strip plot (on the right side of the figure). In the strip plot, each point represents a trip.

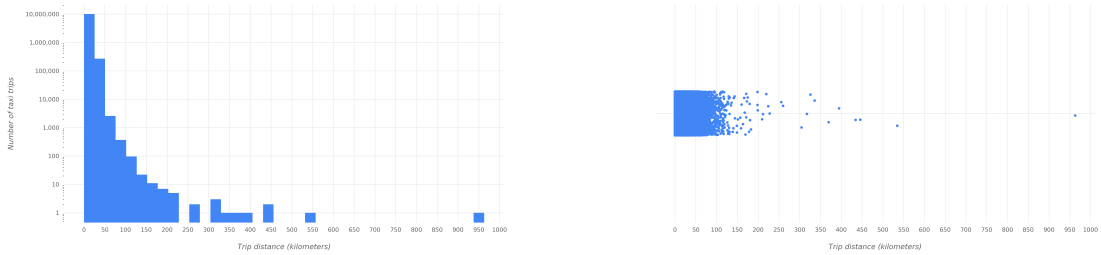

Figure 4.22: Histogram of trip distances in March 2017

From the graph we can note that for the vast majority (97.3%) of taxi trips, a distance less than 25 km was travelled (notice that the y axis is log-scaled). It can also be seen that there are some trips (2.7% of trips) with a travel distance between 25 km and 225 km. We can notice in the graph that there is a trip that is distant from other trips in the histogram with a travelled-distance of ~950 km. To investigate further about this trip, we retrieved some data about it. The data retrieved specifies that in this trip, a distance of 963 km was travelled in 20.7 minutes only. This indicates a clear error in the record of this outlier trip and thus, this record was deleted.

In a similar fashion, Figure 4.23 shows the distribution of trip distances for March 2018. The figure shows that for the vast majority (97.3%) of taxi trips in March 2018 also, a distance less than 25 km was travelled. It shows also that there are some trips (2.7% of trips) with a travel distance between 25 km and 225 km.

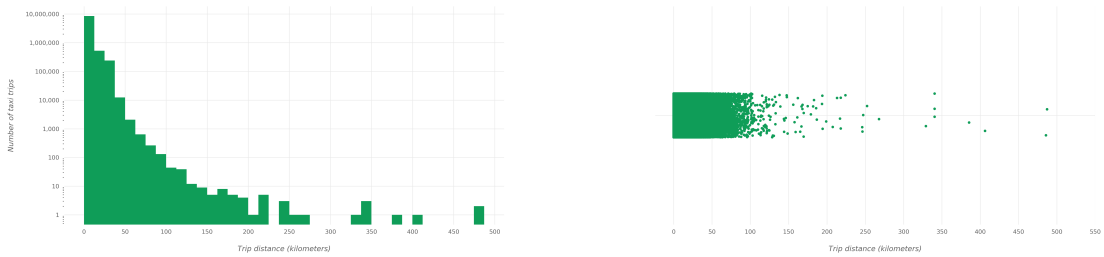

Figure 4.23: Histogram of trip distances in March 2018

4.5.6 Distribution of Rate-Code-ID Variable

Rate code is used to indicate how the trip fare is calculated depending on whether the trip destination is inside New York City for example. For the meaning of each rate code, see Table 3.1 Figure 4.24 shows the number of taxi trips for each rate code ID in March 2017.

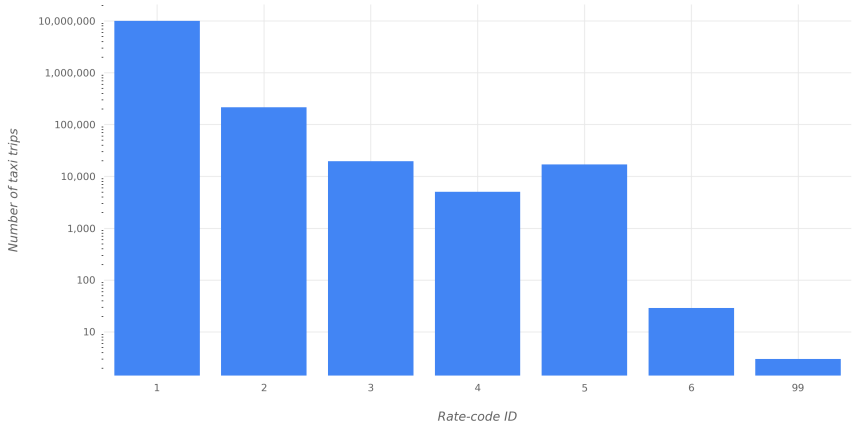

Figure 4.24: Number of trips for each rate code in March 2017

The figure shows that the vast majority (97.5%) of trips had a rate-code ID of 1 which denotes standard rate (notice that the y axis in the plot is log-scaled). Other trips had different rate-code IDs but we notice that there is a rate-code ID in the plot that is not mentioned in the documentation of the dataset (see Table 3.1) and its value is not consistent with the rest IDs. This ID is 99; it should indicate a missing rate-code value for some trip records. Further investigation in the dataset revealed that there are only three records in the dataset with this rate-code ID. The values of other fields for these three records seem correct; nothing peculiar can be seen in them.

In a similar manner, Figure 4.25 shows the number of taxi trips for each rate code ID in March 2018.

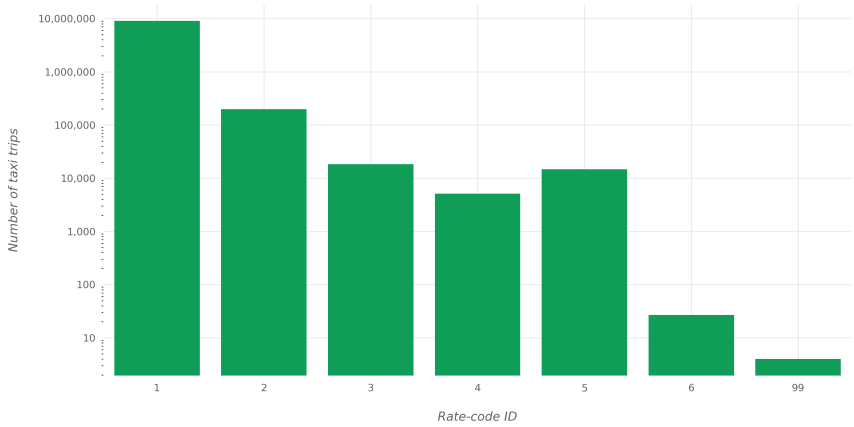

Figure 4.25: Number of trips for each rate code in March 2018

As with March-2017 data, the vast majority (97.5%) of trips had a rate-code ID of 1. And we notice also the ID of 99. For March-2018 data, we found four records with this ID and the values of other fields for these four records seem correct too.

4.5.7 Distribution of Store-and-Forward-Flag Variable

The store-and-forward-flag field in taxi-trips datasets has two possible values: “N” and “Y”. If a taxi trip has “Y” as the value for this field, it means that the taxi was not connected to the Internet and that the taxi will store trip information and send it when it connects to the Internet again. For the opposite case, when a taxi trip has the value “N” for this field, it indicates the normal situation where the taxi was connected to the Internet. Figure 4.26 shows the percentage of taxi trips for each of the store-and-forward-flag values for the data of March 2017.

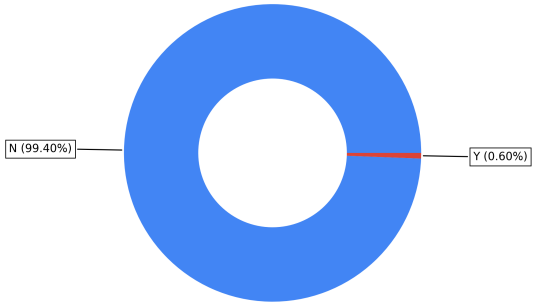

Figure 4.26: Distribution of store-and-forward-flag values in March 2017

The figure shows that for almost all trips, the value of the store-and-forward flag was “N” which means that the taxi was connected to the Internet and trip information was sent directly.

Figure 4.27 shows the distribution of the store-and-forward-flag values for the data of March 2018. We can conclude from the figure that almost all trips in March 2018 also had the value “N” for the store-and-forward-flag field.

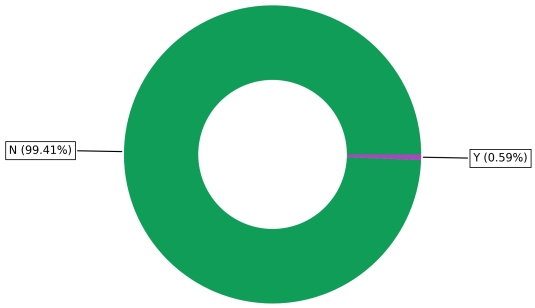

Figure 4.27: Distribution of store-and-forward-flag values in March 2018

4.5.8 Distribution of Payment-Type Variable

Payment-type field in taxi-trips datasets indicates the method used by the passenger for taxi-fare payment. The possible values of this field and their meanings were listed in

Table 3.1. Figure 4.28 shows the number of trips where each payment type was used in March 2017.

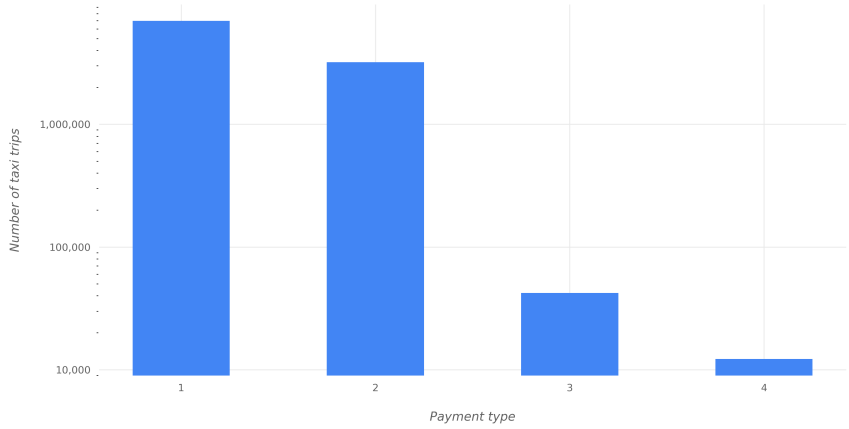

Figure 4.28: Number of trips for each payment type in March 2017

From the figure we can see that in a large number of trips (68.2%), the payment method used was method 1 which means credit-card method by referring to Table 3.1. We also note that in a significant number of trips (31.3%), the payment method used was the one coded as 2 which stands for cash payment method. A small portion of trips (0.5%) had payment types 3 and 4 which stand for "no charge" and "dispute" respectively.

Likewise, Figure 4.29 shows the number of trips where each payment type was used in March 2018. We can see from the figure that a large number of trips (69.8%) used payment type 1, a considerable portion (29.7%) used type 2, and a tiny portion (0.5%) used types 3 and 4.

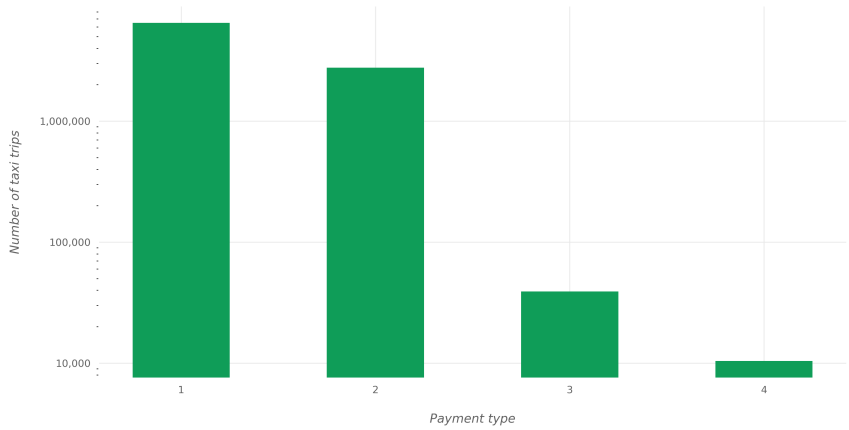

Figure 4.29: Number of trips for each payment type in March 2018

4.5.9 Distribution of Total-Charge-Amount Variable

The field named `total_amount` in taxi-trips datasets specifies the total amount charged to the passenger in a taxi trip in USD currency. Figure 4.30 shows the distribution of this field values in March 2017 using a histogram plot.

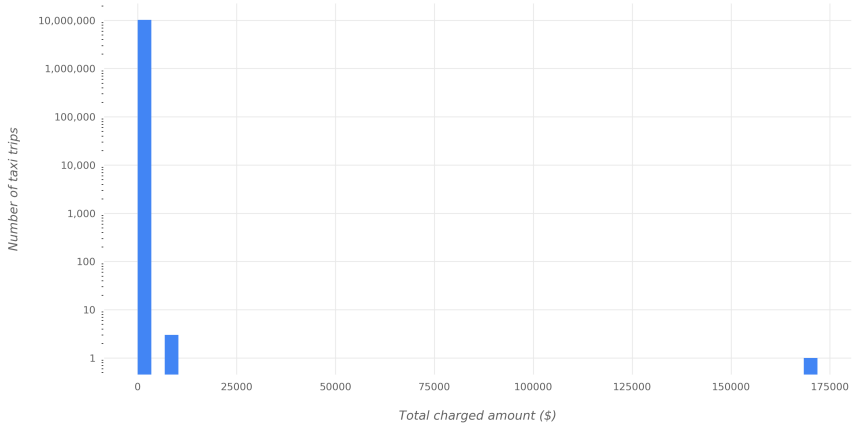

Figure 4.30: Distribution of trip total-charge-amount in March 2017

The first thing we notice in the graph is the single trip in the far right of the graph; we can see that this trip has a completely unrealistic charge amount which is more than USD 150,000. By retrieving the record of this trip from the dataset, we knew that only 6 km were travelled in this trip in around 22 minutes for a total charge amount of USD 171,863.58. This indicates an obvious error in the record of this trip. Thus, it was deleted. Figure 4.31 shows the distribution of total-charge-amount variable after removing that record.

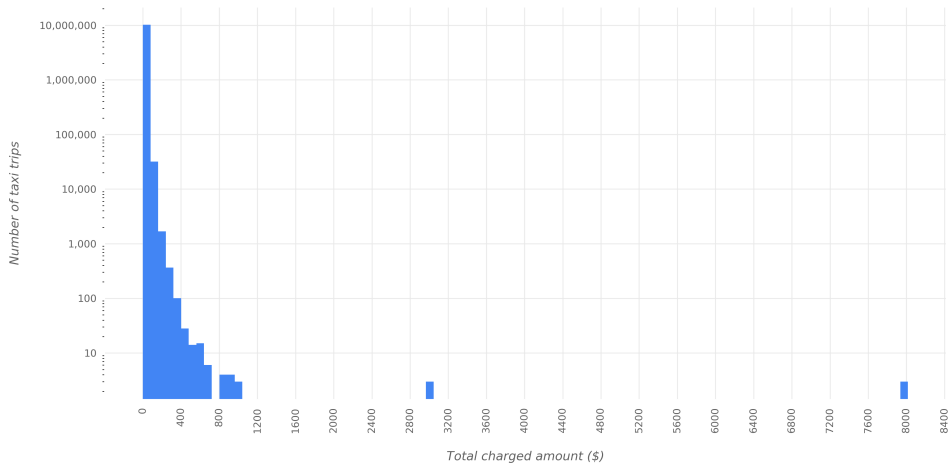

Figure 4.31: Distribution of trip total-charge-amount in March 2017 after removing an erroneous record

From the figure we can see that the vast majority (99.7%) of trips in March 2017 had a charge amount less than USD 80. We can also see that there are six trips with charge amounts noticeably larger than the rest of trips: two of them with a charge amount of USD ~3,000 and two with USD ~8,000. By investigating into the trips of USD ~8,000 charge amount, we see that those trips have trip durations between 4 and 23 minutes and trip distances between 2 and 7 km. It is illogical that such trips cost USD ~8,000. And by investigating into the trips of USD ~3,000 charge amount, we see that their durations range between 9 and 17 minutes and their travel-distances range between 3 and 6 km. This also means that it is illogical for these trips to have a charge amount of USD ~3,000. As a result, these six trip records were deleted.

4.5.10 Number of Taxi Trips by Borough

Using the helper location table which is provided with taxi-trips datasets, we can map location IDs in taxi-trips datasets to the borough of the location as discussed in 3.2. Figure 4.32 shows the number of taxi trips for each borough in March 2017: it shows the number of taxi trips in which the borough was the origin (the pickup location) and the number of taxi trips in which the borough was the destination (the drop-off location).

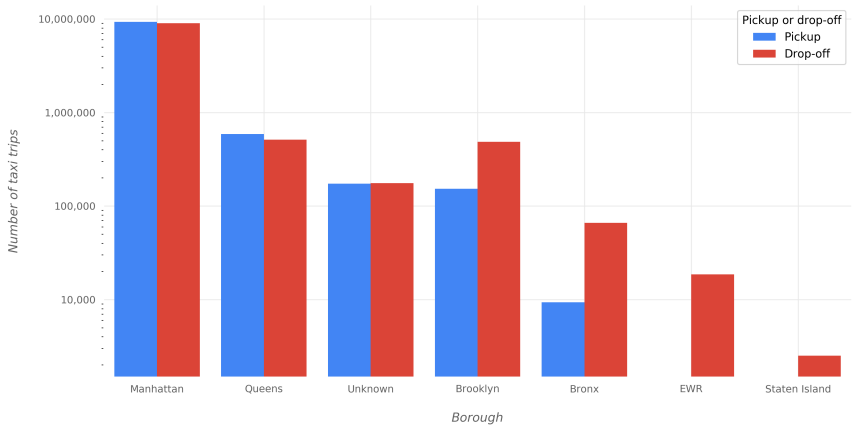

Figure 4.32: Number of taxi trips for each borough as an origin and a destination in March 2017

Similarly, Figure 4.33 shows the number of taxi trips for each borough in March 2018.

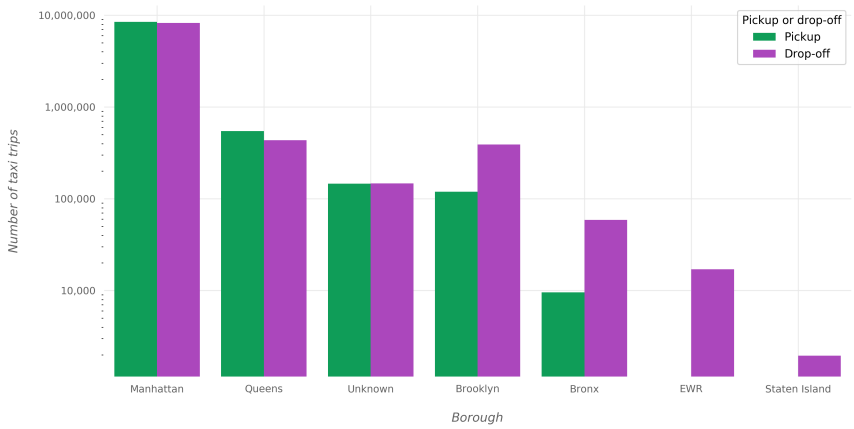

Figure 4.33: Number of taxi trips for each borough as an origin and a destination in March 2018

We can see that the vase majority of trips both in March 2017 and March 2018 had Manhattan borough as the pickup borough; the vast majority of trips also had Manhattan as the drop-off borough. We can also see that there is a borough named "Unknown"; it seems that the actual borough is missing for trips that have this borough as pickup or drop-off borough. Only 1.9% of trips in March 2017 and 1.8% of trips in March 2018 had "Unknown" as the pickup or drop-off borough. Finally, we note that some boroughs—EWR and Staten Island—are much more drop-off destinations than being pickup origins. They were the pickup borough for some trips but we can't see that in the graphs because the number of these trips is tiny.

4.5.11 Distribution of Daily-Temperature Variables

The weather datasets in this project specify the minimum, average, and maximum temperatures for all days of the two months of interest in this project which are March 2017 and March 2018. Figure 4.34 shows the distributions of minimum, average, and maximum temperatures over the days of March 2017 using box plots. Along with the box plots, there are strip plots which are the grey dots that appear on top of the box plots; the grey dots represent actual daily temperatures: each dot represents the temperature of a day in March 2017.

Note: For box plots, many people use the value $1.5 \times \text{IQR}$ to determine the whiskers positions. However, throughout this report, when we use box plots, the whiskers extend such that they include all data points as shown in Figure 4.34.

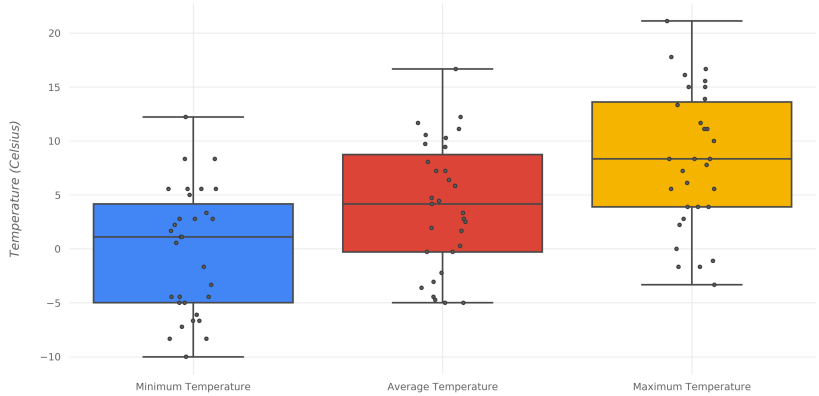

Figure 4.34: Daily temperature distribution in March 2017

From the figure we can see that in March 2017, the minimum temperature in a day ranges between -10 °C and ~12 °C, the average daily temperature ranges between -5 °C and ~17 °C, and the maximum daily temperature ranges between ~-3 °C and ~21 °C. We can deduce that the minimum temperature in the whole March 2017 was -10 °C while the maximum temperature was ~21 °C.

Figure 4.35 shows the minimum, average, and maximum temperature for each day of March 2017. We can see from the figure that the temperature varies from day to day. Sometimes the variation in temperature from one day to the next is significant; for example, the average temperature drops from ~10 °C on the 2nd of the month to ~0 °C on the

3rd of the month. We expect that this variation will prove useful in predicting trip duration in the modeling process later.

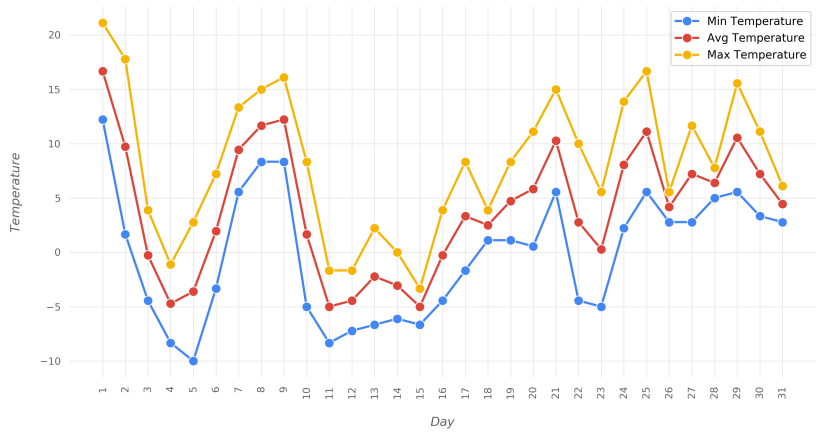

Figure 4.35: Minimum, average, and maximum temperature for each day in March 2017

Similarly, Figure 4.36 shows the distributions of minimum, average, and maximum temperatures over the days of March 2018.

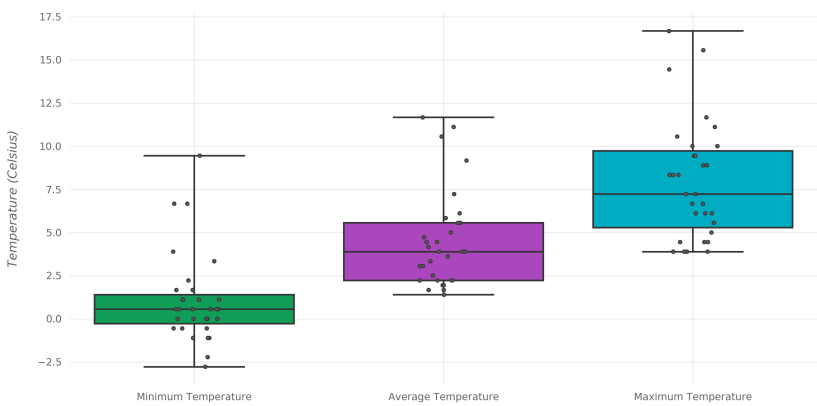

Figure 4.36: Daily temperature distribution in March 2018

From the figure we can see that in March 2018, the minimum temperature in a day ranges between ~-2.5 °C and ~9 °C, the average daily temperature ranges between ~1 °C and ~12 °C, and the maximum daily temperature ranges between ~4 °C and ~17 °C. We can deduce that the minimum temperature in the whole March 2018 was ~-2.5 °C while the maximum temperature was ~17 °C. By comparing the temperature ranges in March 2018 to March 2017, we can see that the ranges in March 2018 are narrower. Figure 4.37

enables a visual comparison between the temperature ranges in March 2017 and March 2018.

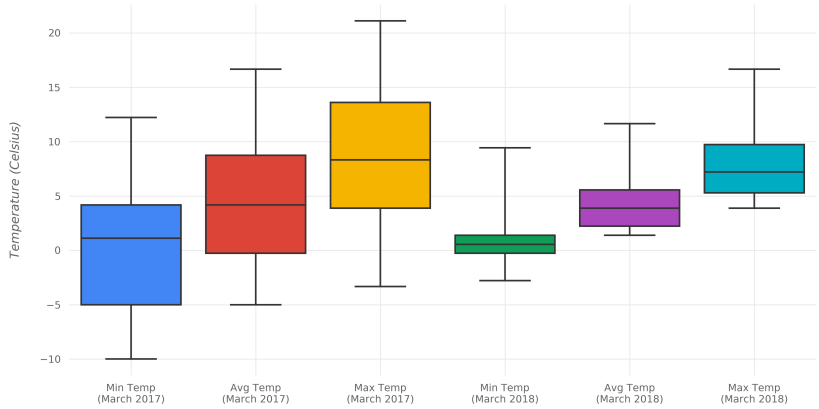

Figure 4.37: Daily temperature distribution in March 2017 and March 2018

Figure 4.38 shows the minimum, average, and maximum temperature for each day of March 2018. From the figure we can notice that there are also variations in temperature from day to day although to a lesser degree than March 2017 (see Figure 4.35).

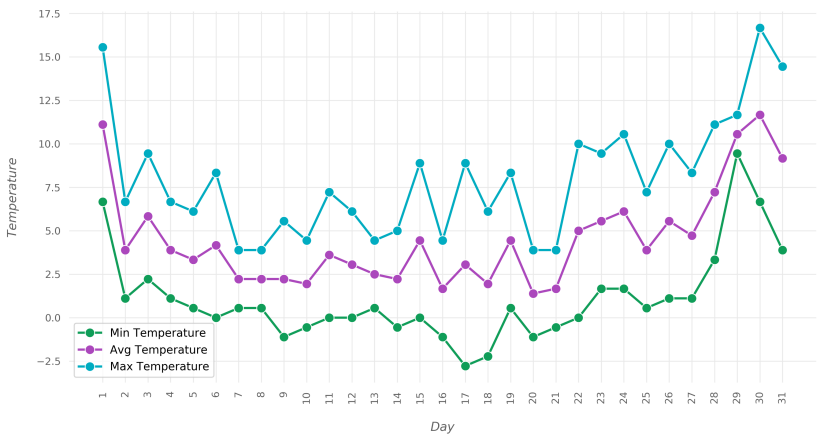

Figure 4.38: Minimum, average, and maximum temperature for each day in March 2018

4.5.12 Distribution of Daily-Precipitation Variables

Along with the temperature, the weather datasets in this project specify the precipitation for all days of the two months we are studying in this project which are March 2017 and March 2018. Figure 4.39 shows the distributions of preciptation over the days of March

2017 using a histogram plot (on the left) and a box plot combined with a strip plot (on the right).

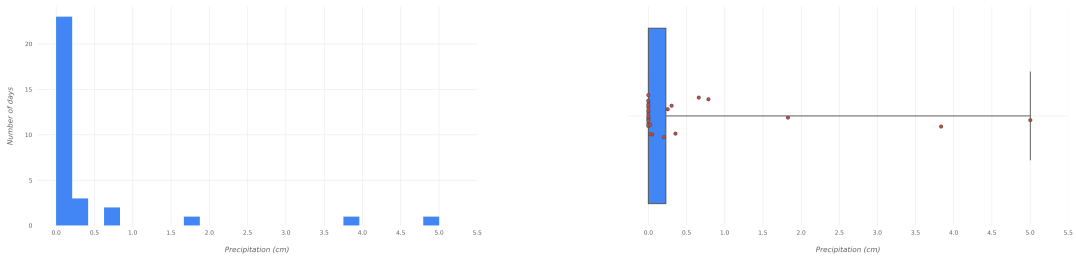

Figure 4.39: Daily precipitation distribution in March 2017

We can see that precipitation in most days of the month is zero or close to zero. We also can see that there are some days in the month with relatively high degree of precipitation. We expect that these high degrees of precipitation will affect the durations of taxi trips in the corresponding days. Figure 4.40 shows the distribution of snowing over the days of March 2017 using a histogram to the left of the figure and a strip plot on the right side of the figure.

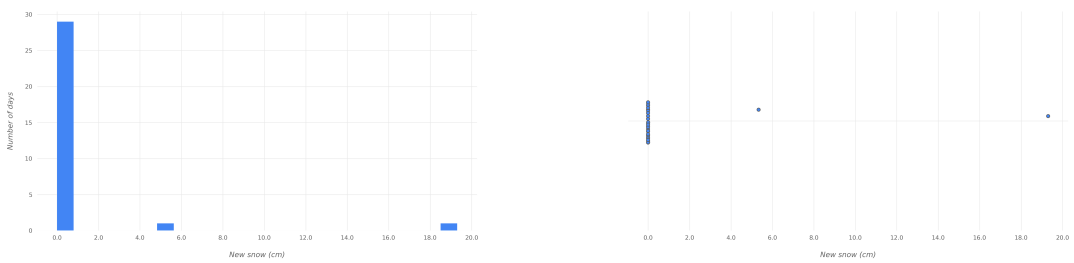

Figure 4.40: Daily snow-fall distribution in March 2017

We can see that for almost all days of March 2017, there was no snowing; snowin occured on two days of the month only. Snowing, especially in big amounts, affects traffic conditions and trip duration. Thus, we expect that this data about snowing levels will increase the machine-learning-model ability to generate accurate trip-duration predictions.

In a similar way, Figure 4.41 shows the distribution of preciptation and Figure 4.42 shows the distribution of snowing over the days of March 2018.

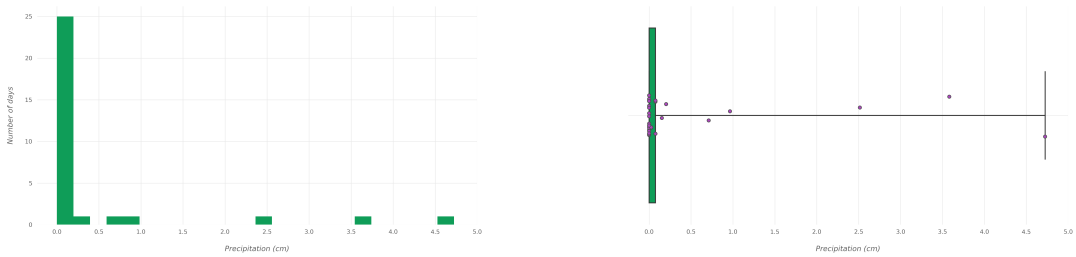

Figure 4.41: Daily precipitation distribution in March 2018

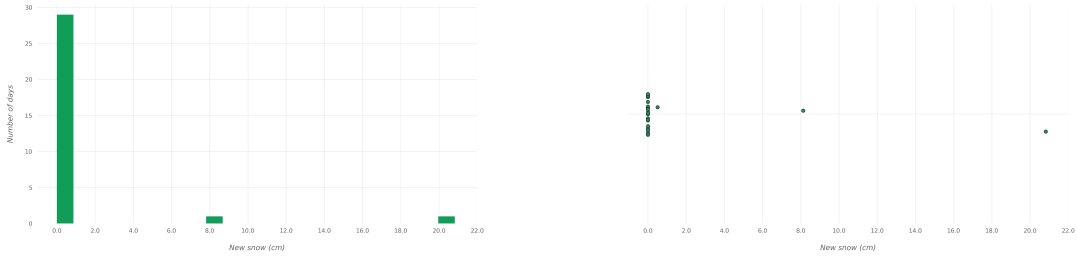

Figure 4.42: Daily snow-fall distribution in March 2018

4.6 Expolratory Data Analysis: Exploring Relationships between Variables

4.6.1 Relationship between Trip Duration and Trip Distance

We start by exploring the relationship between the trip-distance variable and the target variable which specifies trip duration. Figure 4.43 shows a scatter plot between the two variables in March 2017 and in March 2018. Since scatter plot is not ideal for a large dataset with a lot of points which is the case with our dataset (more than 10 million records), we will also use a 2D histogram (or 2D density plot) which counts the number of data points in a particular area of the 2D space. Figure 4.44 shows the 2D histogram of trip-distance and trip-duration variables in March 2017 and March 2018 with two marginal plots on the top and on the left of the plot; the marginal plots show the histogram for each variable separately.

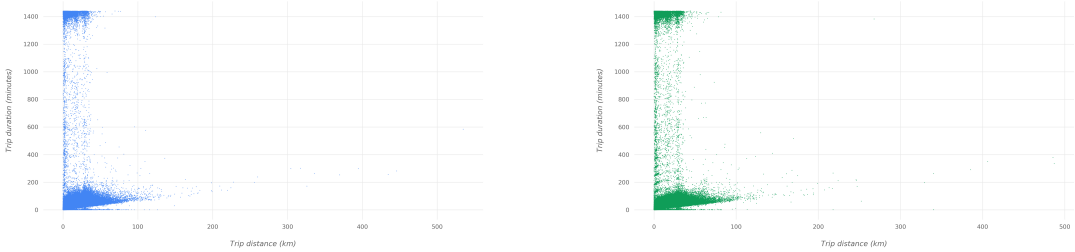

Figure 4.43: Scatter plot between trip duration and trip distance in March 2017 (left) and in March 2018 (right)

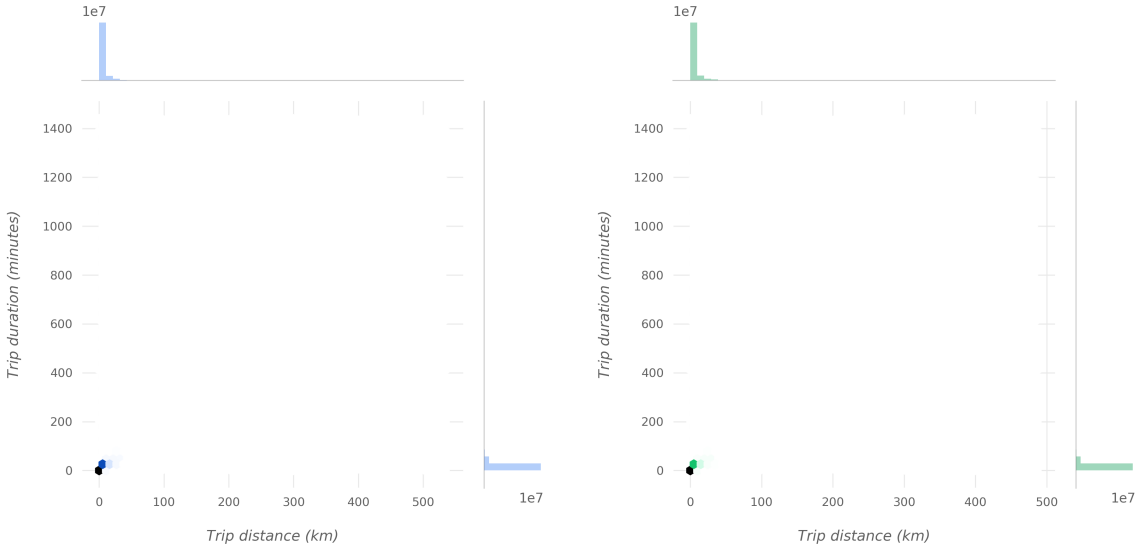

Figure 4.44: 2D histogram of trip duration and trip distance in March 2017 (left) and in March 2018 (right)

By looking at Figure 4.43, we see that in some trips, the distance travelled is not small (e.g. more than 100 km) but the trip duration is ~0. Moreover, we see trips with extremely long durations like 1400 minutes (which is more than 23 hours). We also see that for many data points (i.e. taxi trips), the trip duration changes while the trip distance does not change. Changes in trip duration for the same trip distance can be justified by the changes in traffic conditions; for example, in some parts of the day, when the traffic jam is at its maximum, taxis move more slowly and trip duration increases. But in the scatter plot, we see that for trips with the same trip distance, the duration differs hugely; we see one trip with a duration of 200 minutes and another trip with the same distance but with a duration of 1400 minutes; we can see that many data points in the scatter plot have the same issue but we can't determine the percentage of these data points out of the total data points using the scatter plot. For that purpose, we look at Figure 4.44 to see that most data

points concentrate at the bottom-left of the plot—the darker the hexagon is, the more data points it represents; we see that few points (i.e. few trips) have a duration more than ~50 minutes; we also see that trip distance and duration tend to have some linear correlation.

To filter out the incorrect data points that appear in Figure 4.43, we will inspect the average speed for taxi trips. We have two possible error situations: 1) When a trip with a short distance has a very long duration. In this case, the average speed will be very low—for example, a speed of 0.5 kilometers/hour. 2) When a trip with a certain distance has a relatively very short duration. In this case, the average speed will be too high. Thus, by inspecting the average speed of taxi trips, we will be able to capture both error cases and consequently filter the erroneous data records out.

When we look at the distribution of the speeds of taxi-trips in Figure 4.45 where a box plot combined with a strip plot were used, we can see that there are trips with completely unrealistic speeds; we see trips with a speed of 1,000 km/h and 10,000 km/h. On the other hand, we see trips with extremely low speeds; we see trips with a speed of 0.1 km/h and 0.01 km/h. To filter out trips with wrong speeds, we need to set a lower speed limit and an upper speed limit such that trips whose average speeds fall between these two limits are kept and trips whose average speeds fall below the lower limit or above the upper limit get filtered out.

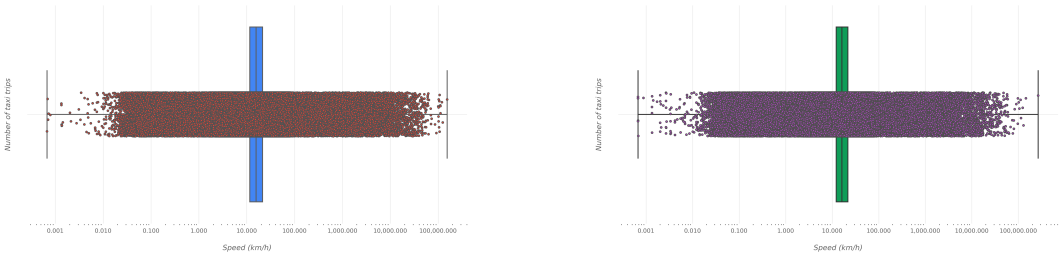

Figure 4.45: Distribution of taxi-trips speeds in March 2017 (left) and in March 2018 (right)

For the upper speed limit, according to New York State Legislature (2019), the maximum allowed speed in New York state is 65 miles per hour which is equivalent to 105 km/h. Based on that, we can say that most taxi trips in New York City should have speeds less than 105 km/h since this is the maximum allowed speed regardless of the location inside New York state. Thus, we will set 120 km/h as our upper speed limit which is the

maximum allowed speed plus 15 km/h margin.

For the lower limit, we will use median-absolute-deviation method as described by Leys, Ley, Klein, Bernard, and Licata (2013) to determine it. Median absolute deviation is a robust method to detect outliers in data. The lower limit will be $M - 2.5 \times MAD$ where M is the median speed, MAD is the median absolute deviation, and 2.5 is the threshold chosen. We calculate MAD based on its implementation in SciPy 1.3.0 by The SciPy Developers (n.d.) as shown in Figure 4.46.

```
speed_mad = np.median(np.abs(data['speed (km/h)'] - np.median(data['speed (km/h)'])))
```

Figure 4.46: Code used to calculate MAD for taxi speeds in March 2017

By applying the aforementioned equation for both March-2017 and March-2018 datasets, we get the value for the lower speed limit as 3.95 km/h for March 2017 and 4.7 km/h for March 2018. These are reasonable values because we don't expect a taxi trip to have an *average* speed less than that.

After determining the lower and upper speed limits, we remove data points whose speeds don't fall between these limits. A total of 97,035 records (only 0.95% of records) will be removed from March-2017 dataset and a total of 95,942 records (only 1.03% of records) will be removed from March-2018 dataset.

After removing those erroneous records, Figure 4.47 shows the scatter plots between trip distance and trip duration. By comparing these plots to those in Figure 4.43, we can see that removing the erroneous records improved the quality of the data by alleviating the problems discussed previously. However, we can notice in each plot in 4.47 a single point that is distant from other points and has a very large duration (~1400 minutes which is ~ 23 hours); this point will be considered as an outlier also and will be removed.

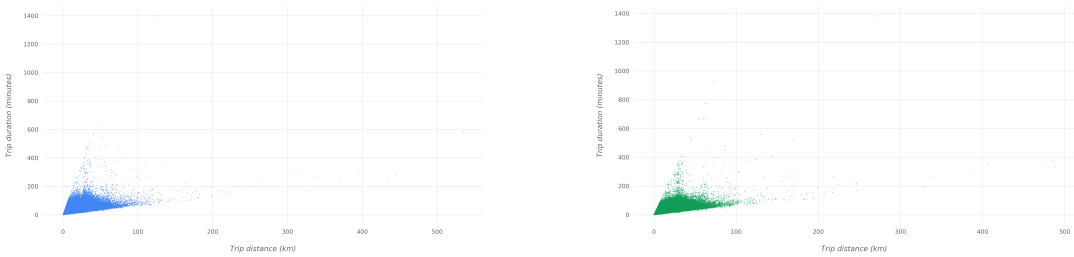

Figure 4.47: Scatter plot between trip duration and trip distance in March 2017 (left) and in March 2018 (right) after removing trips with wrong speeds

4.6.2 Average Trip Duration Per Hour of the Day

To see how trip duration is related to the time of the day in which the trip takes place, we plot a line plot showing the average trip duration for each hour of the day. Figure 4.48 shows this plot for March-2017 and March-2018 data. For example, in the left-side plot, the first point from left represents the average trip duration for trips in March 2017 whose pickup hour was between 12:00 a.m. and 1:00 a.m.

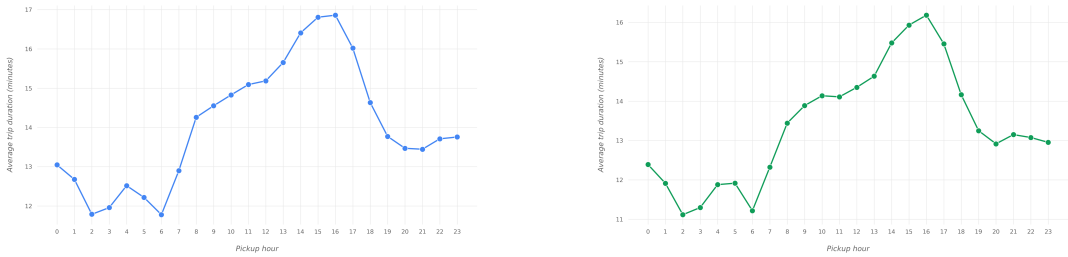

Figure 4.48: Average trip duration by pickup hour-of-the-day in March 2017 (left) and March 2018 (right)

From the figure we can see that there is a significant variation in average trip time from hour to hour; this is clearly seen if we compare the average trip duration for trips with pickup-hour between 6:00 a.m. and 7:00 a.m.—which equals ~12 minutes for March 2017 and ~11 minutes for March 2018—against the average trip duration for trips with pickup-hour between 3:00 p.m. and 4:00 p.m.—which equals ~17 minutes for March 2017 and ~16 minutes for March 2018. This figure indicates the importance of using pickup hour in predicting trip duration.

4.6.3 Average Trip Duration Per Borough

Is there a relationship between trip duration and the pickup or drop-off borough? Does trip duration change based on the destination borough? To answer these questions, we draw a bar plot to show the average trip duration for each borough both as a pickup borough and as a drop-off borough. Figure 4.49 shows this plot for both March 2017 and March 2018.

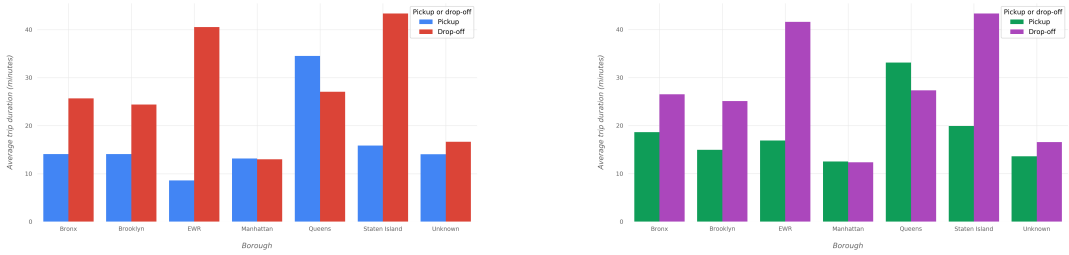

Figure 4.49: Average trip duration per borough in March 2017 (left) and March 2018 (right)

We can see that average trip duration differs greatly from some boroughs to other boroughs. For example, in March-2017 plot, we can see that trips whose pickup-borough was Queens had much greater average trip duration (~35 minutes) than trips whose pickup-borough was EWR (~8 minutes).

4.6.4 Average Trip Duration Per Borough and Hour of the Day

After we explored average trip duration per hour of the day and per borough, we now explore average trip duration per borough per hour of the day. Figure 4.50 shows a group of line plots that demonstrate this exploration for March 2017 and Figure 4.51 shows the same but for March 2018. Each figure shows average trip duration per borough per hour, one time with boroughs as pickup boroughs and another time with boroughs as drop-off boroughs.

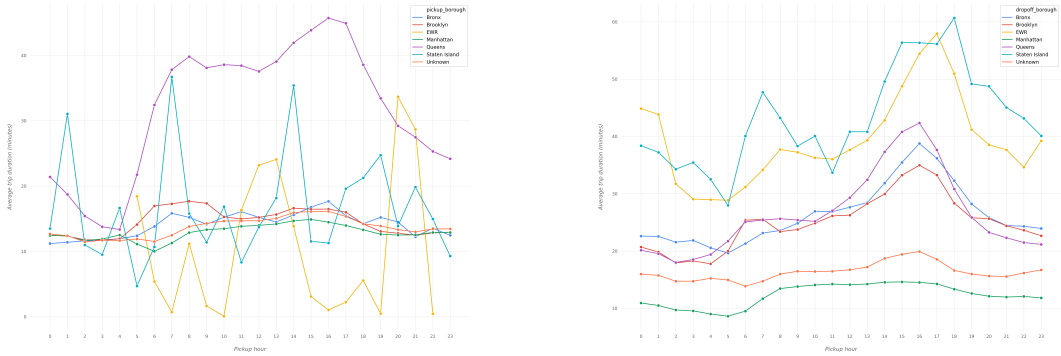

Figure 4.50: Average trip duration per borough per hour of the day in March 2017

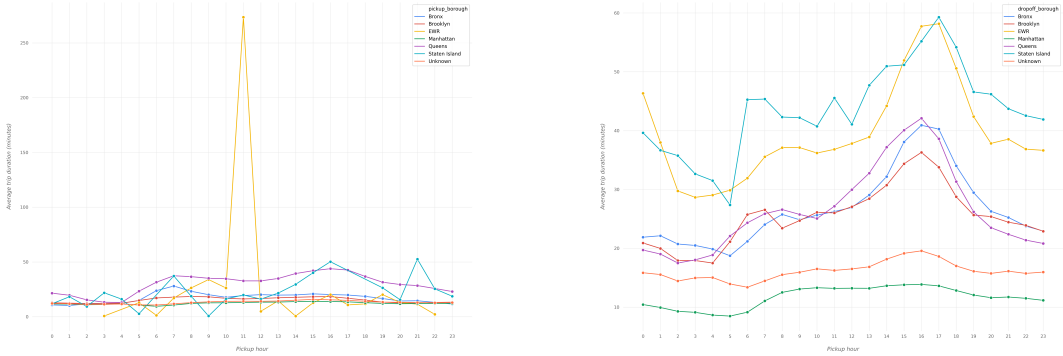

Figure 4.51: Average trip duration per borough per hour of the day in March 2018

These figures show how average trip time differs from borough to borough and from hour to hour for each borough We can see that for some boroughs like EWR and Staten Island, average trip duration varies from hour to hour in greater degrees than other boroughs.

4.6.5 Average Trip Duration Per Day of the Month

Is average trip duration constant or almost constant for all days of the month? To get the answer for this question, we plot the average trip duration for each day in March 2017 and in March 2018. Figure 4.52 shows such a plot for each month. The figure shows weekend days in New York City (Saturday and Sunday) in different color than weekdays.

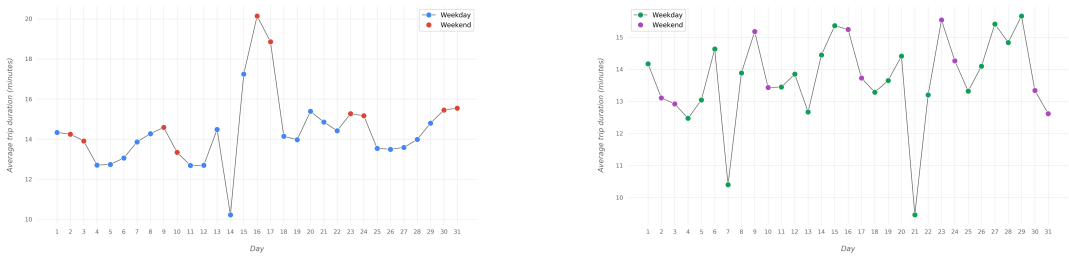

Figure 4.52: Average trip duration per day of the month in March 2017 (left) and in March 2018 (right)

The figure clearly shows that average trip duration varies between the days of each month; we see that March 2018 had larger variations between days than March 2017. Also, we note that the line plots of the two months are not very similar; they don't follow the same patterns. When we calculate the Pearson correlation coefficient between the average trip durations of the days of March 2017 and the average trip durations of the days of March 2018, we get a correlation coefficient of 0.18; this means that there is some positive correlation but it is not strong.

4.6.6 Relationship between Pickup-Borough and Pickup-Times

In this part of the exploration, we want to see the most popular pickup times in each borough. Figure 4.53 shows a group of plots where each plot shows the number of taxi trips in each hour in March 2017 for one of the boroughs. Figure 4.54 shows the same but for March 2018.

Figure 4.53: Number of taxi trips in each borough in each hour of the day in March 2017

Figure 4.54: Number of taxi trips in each borough in each hour of the day in March 2018

We can see that the popular pickup hours differs from one borough to another within the

same month; we also notice that the most popular pickup hours are similar for the same borough in the two months. For example, in March 2017, we can interestingly see that the two most popular pickup-hours for Bronx borough are 7 a.m. and 8 a.m. (early morning), for Brooklyn borough are 11 p.m. and 12 a.m. (late night), for Manhattan are 6 p.m. and 7 p.m. (evening), and for Queens are 5 p.m. and 10 p.m.

4.6.7 Relationship between Number of Passengers and Trip Duration

Figure 4.55 shows the distribution of trip duration for each number of passengers in March 2017 and March 2018. The distribution is shows using violin plots which are plots similar to box plots; they utilize a kernel density estimation of the underlying distribution; when the violin is wider, this means that more data points are included in that wide area and vice versa. There is a box plot inside each violin where the white dot indicates the median of the data to convey more meaning. From the figure we can see that as passenger count increases, trips tend to have longer durations. This indicates the benefit of using the passenger-count feature in modeling.

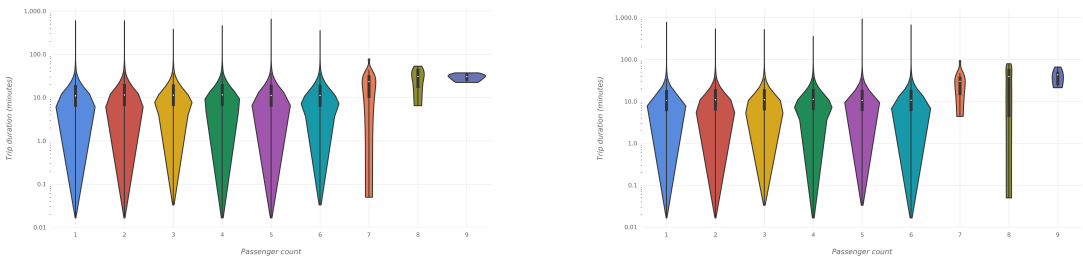

Figure 4.55: Trip duration per the number of passengers in March 2017 (left) and March 2018 (right)

4.6.8 Relationship between Rate-Code ID and Trip Duration

Figure 4.56 shows the distribution of trip duration for each unique rate-code ID in March 2017 and March 2018. The distributions are displayed using strip plots. The figure shows that trips with rate-code IDs of 1, 2, 3 and 5 have similar trip-duration distributions. Rate-code ID of 4 doesn't have trips with as short durations as 1, 2, and 3 rate-code IDs trip-durations. We note also that rate-code ID of 6 has only a few trips with different durations. And finally, we see that rate-code ID of 99—which indicates a missing rate-code ID—has

a few trips as well. The different distributions that are associated with different rate-code IDs indicates a possible value for rate-code ID in building the predictive model.

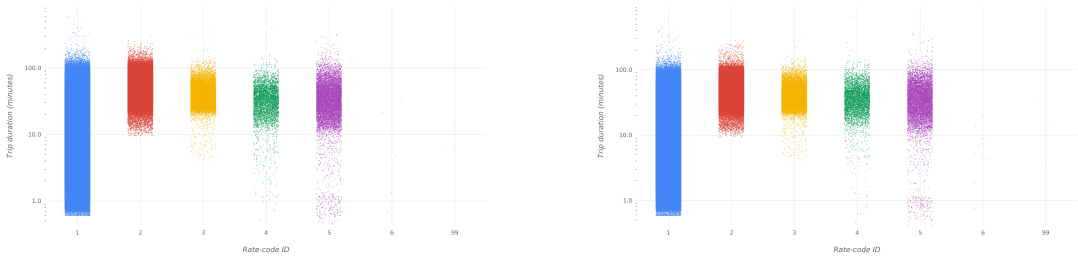

Figure 4.56: Trip duration distribution for each rate-code ID in March 2017 (left) and March 2018 (right)

4.6.9 Relationship between Vendor ID and Trip Duration

Figure 4.57 shows the distribution of trip duration for each vendor ID in the data for March 2017 and March 2018. We can say, depending on the figure, that vendor ID doesn't carry much meaning because trip distribution is very similar for each vendor.

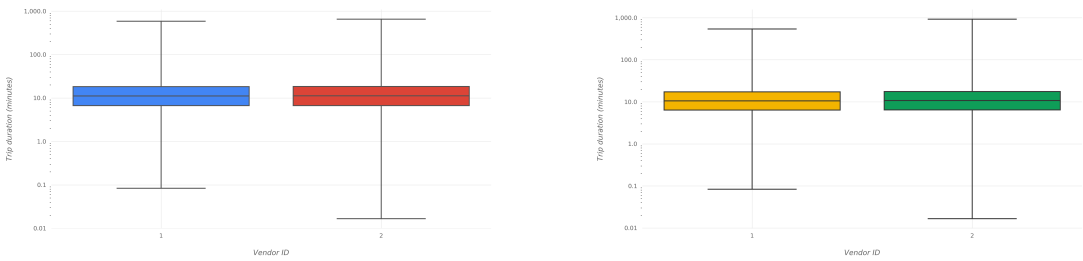

Figure 4.57: Trip duration distribution for each vendor ID in March 2017 (left) and March 2018 (right)

4.6.10 Relationship between Store-and-Forward Flag and Trip Duration

Figure 4.58 shows trip-duration distributino for the two cases of store-and-forward flag which are "Y" for yes and "N" for no. For March 2017, we can see that when the store-and-forward flag is set to "Y", trips tend to have longer durations; we can see also that when the flag is "Y", trips doesn't have trip durations as short or as long as when the flag is "N". For March 2018, we see that this difference doesn't exist although we can note that when the flag value is "Y", trips has more limited trip-duration distribution.

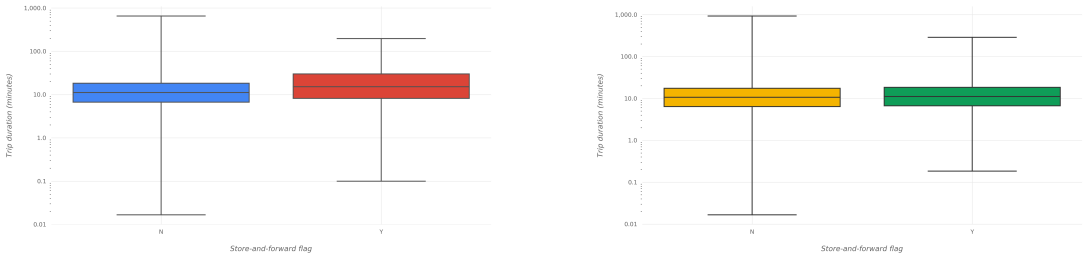

Figure 4.58: Trip duration distribution for the cases of store-and-forward flag in March 2017 (left) and March 2018 (right)

4.6.11 Relationship between Total Charge Amount and Trip Duration

Figure 4.59 shows scatter plots that depict the relationship between total charge amount of trips and their durations in March 2017 and March 2018. From March-2017 plot we can see the expected correlation between trip charge amount and duration. However, we see some trips with considerable charge amounts (like USD 200) but with a duration close to zero. Also, we see trips with varying considerable durations (like 100 minutes) but with a charge amount close to zero. For March 2018 plot, we can notice that there are some data points which are very distant than the others which prevents the plot from showing the distribution of data points in a proper way. Thus, there are trips with erroneous records in taxi-trips datasets that should be deleted.

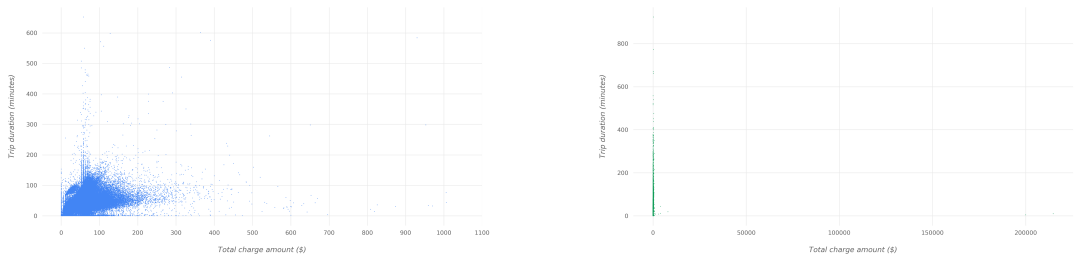

Figure 4.59: Scatter plot between trip total-charge-amount and duration in March 2017 (left) and March 2018 (right)

To determine which trips to delete, we calculate a new variable which is the result of dividing total charge amount by duration for each trip; this variable represents the money paid by minute. After that, and similar to what we did in 4.6.1, we calculate an upper and lower limits using median-absolute-deviation method to filter out outlier records. To determine the lower limit, we apply this formula $M - 4 \times MAD$ and to determine the upper limit, we

use this formula with different threshold value: $M + 20 \times MAD$. Threshold values were chosen to control the strictness used to determine outliers such that we filter out erroneous outliers but at the same time we don't delete a lot of meaningful data. The formulas yield 0.25/0.26 USD per minute as the lower limit and 5.59/5.70 USD per minute as the upper limit for March-2017/March-2018 data. This results in deleting 21,413 (0.2%) records out of 10,126,642 records in March 2017 and 19,994 records (0.2%) out of 9,199,661 records in March 2018. Figure 4.60 shows the scatter plots after removing outliers. We can see how removing outliers solved the problems discussed earlier.

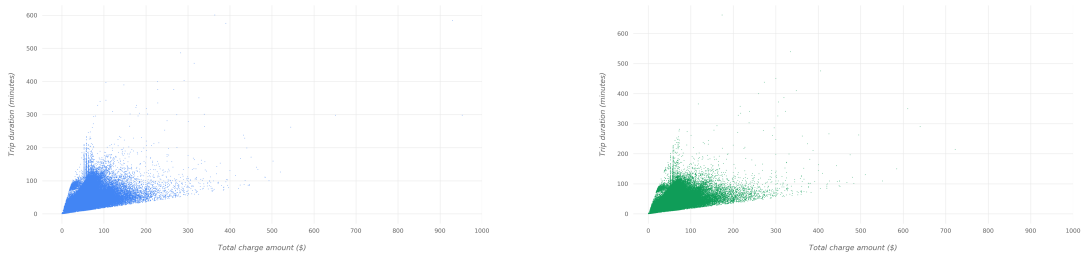

Figure 4.60: Scatter plot between trip total-charge-amount and duration after removing outliers in March 2017 (left) and March 2018 (right)

4.6.12 Correlation between Variables

To see how variables in our datasets are linearly related, we calculate the correlation coefficient between the variables. Specifically, we will calculate Pearson correlation coefficient betweem variables. This correlation coefficient ranges from -1 to +1 where -1 means strongest negative correlation, +1 means strongest positive correlation, and 0 indicates no correlation. Figure 4.61 shows a heatmap that visualizes the correlation between some variable in taxi-trips datasets of March 2017 and March 2018. Knowing the degrees of correlations between variables, especially with the target variable, can help in determining predictor features in modeling process.

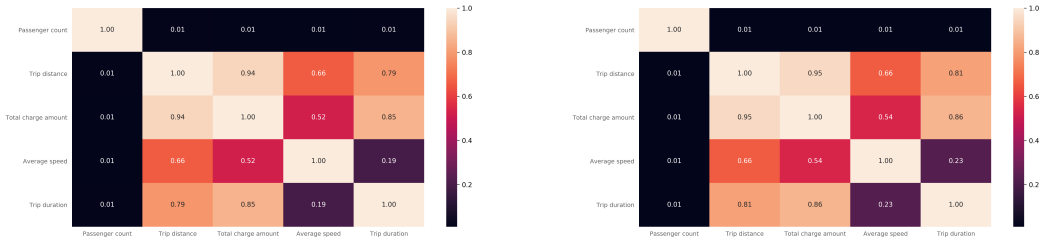

Figure 4.61: Correlation between variables in taxi-trips datasets of March 2017 (left) and March 2018 (right)

From the figure we can see the degree of correlation between each variable and the target variable, trip duration: we can see that there is a strong positive correlation between trip duration and both trip distance and total charge amount which is logically anticipated; we can also see some positive correlation between trip duration and average speed; and finally, we can see that there is no linear correlation between trip duration and passenger count; in fact, passenger count has no correlation with all other variables in the plots. More than that, we can notice strong positive correlation between total charge amount and trip distance, and significant positive correlation between average speed and each of trip distance and total charge amount.

Figure 4.62 shows correlation heatmap for weather datasets for both months of this study. It can be seen that there is a strong positive correlation between precipitation and snowing amount (i.e. “New Snow”); we can see also a significant negative correlation between average temperature and snow depth.

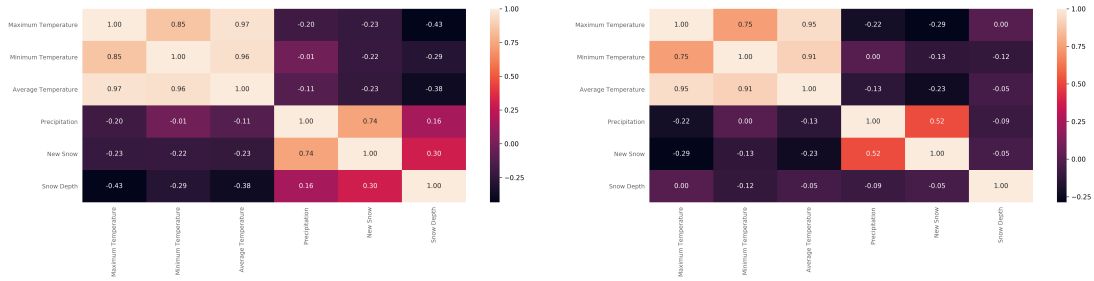

Figure 4.62: Correlation between variables in weather datasets of March 2017 (left) and March 2018 (right)

4.7 Conclusion

In this chapter, datasets of this study were preprocessed, cleaned, and prepared for the next stages of modeling and analysis. An extensive exploratory data analysis (EDA) was applied on the datasets which helped in understanding the data better, discovering the nature of relationships between different variables in the datasets, and spotting many errors and irregularities in the data; spotting those errors was followed by correction processes to alleviate the effects of these errors on the modeling processes later.

CHAPTER 5: MODELING AND RESULT DISCUSSION

5.1 Introduction

In this chapter, we show how the predictive machine-learning model is built: first we illustrate the feature-selection and feature-engineering procedures; then the process of choosing the best-performing models out of candidate models will be explained; moreover, we will show the steps of using our data to train the model and then to evaluate the model. Figure 5.1 summarizes the stages of modeling starting with feature selection and engineering and ending with model evaluation.

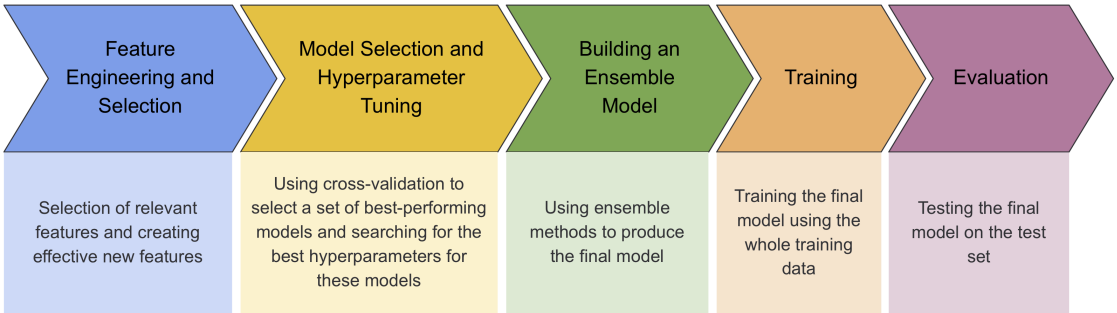

Figure 5.1: The stages of the modeling process

Data Used in Modeling

Due to computation and time limitations, we were not able to perform the modeling operations using the whole data because it is very big (training data alone is more than 10,000,000 rows). Thus, we reduced the data as follows: we kept the trips data of the 1st, 4th, and 5th of March 2017 and the 1st, 4th, and 5th of March 2018 between 1 p.m. and 11 p.m. and removed the rest. This means that our training data consists now of the taxi trips that took place between 1 p.m. and 11 p.m. in the the 1st, 4th, and 5th of March 2017—this is 598,407 rows of data which is still a substantial amount of data. The same applies for test data with 513,089 rows of taxi-trips data.

5.2 Feature Selection

In the previous chapter, we explored the project datasets to understand them and spot errors in them and deal with those errors. In that process, we were able to use all variables (i.e. features or fields) of the data. It is not the case for modeling: for modeling, we have a group of predictor variables that the model uses to predict the target variable. We must exclude any variable from the predictor variables that might cause data leakage. Data leakage occurs when predictor variables include data that will not be available at the time of making predictions. For example in our case, data that can only be available at the end of the taxi trip; we can't include this data in our predictors because our goal is to predict trip duration before the trip starts.

Table 5.1 lists the variables that should be removed from taxi-trips datasets and the removal reason. For weather datasets, the reasoning of using their data is explained in 3.2.3. We will delete three variables—`Departure Temperature`, `HDD`, and `CDD`—from weather datasets because their meaning is not clear to us at the time of performing this study. For the definition of each field in the project datasets, refer to section 3.2.

Table 5.1: Fields to be removed from taxi-trips datasets and reasons of removing them

Field to Be Removed	Reason
tpep_dropoff_datetime	Its value becomes known only at the end of the trip. In fact, if we include this variable in our predictor variables, we will be able to achieve 100% accuracy because the target variable can be calculated using this field
trip_distance	Its value becomes known only at the end of the trip
RateCodeID	According to NYC Taxi and Limousine Commission (n.d.), the value of this field might change during the trip (i.e. it is not determined before the start of the trip)
Store_and_fwd_flag	Its value is determined when the taxi tries to send trip data which is at the end of the trip.
payment_type	From the field description, it seems that the value of this field gets determined at the end of the trip
Fare_amount, Extra, MTA_tax, Improvement_surcharge, Tip_amount, Tolls_amount, Total_amount	These fields are related to the amount of money charged to the passenger at the end of the trip

After removing the aforementioned fields, the taxi-trips datasets look like the table in Figure 5.2 and weather datasets look like the table in Figure 5.3. We can see that taxi-trips datasets contain now five features plus the target variable and that weather datasets contain seven features.

VendorID	tpep_pickup_datetime	passenger_count	PULocationID	DOLocationID	trip_duration
1	2017-03-04 07:00:00	1	48	163	4.150000
2	2017-03-04 07:00:00	1	136	247	8.150000
2	2017-03-04 07:00:00	5	238	132	31.850000
2	2017-03-04 07:00:00	1	163	236	2.200000
1	2017-03-04 07:00:01	1	68	234	5.216667

Figure 5.2: The fields remaining in taxi-trips datasets after removing unwanted fields

Date	Maximum Temperature	Minimum Temperature	Average Temperature	Precipitation	New Snow	Snow Depth
2017-03-04	-1.111111	-8.333333	-4.722222	0.0	0.0	0.0
2017-03-05	2.777778	-10.000000	-3.611111	0.0	0.0	0.0

Figure 5.3: The fields remaining in weather datasets after removing unwanted fields

5.3 Feature Engineering

By doing feature engineering, we use existing features to produce new features that are expected to increase the predictive performance and accuracy of the machine-learning model. The insights obtained from the exploratory data analysis that was performed in chapter 4 will be utilized in this feature-engineering process.

5.3.1 Extraction of Features from Pickup-Datetime

In taxi-trips datasets, we have a feature named `tpep_pickup_datetime` that specifies the date and time of passenger pickup. However, the data type of this feature (i.e. datetime) can't be used directly by most machine-learning models. Thus, we will extract some useful features out of this feature. Figure 5.4 shows the code used to do so.

```
# used to create a feature below
is_weekend_dict = {'Monday': 'No', 'Tuesday': 'No', 'Wednesday': 'No',
                   'Thursday': 'No', 'Friday': 'No', 'Saturday': 'Yes',
                   'Sunday': 'Yes'}

# performing the operations on training and test data
for df in [data, test_data]:
    df['pickup_day_of_month'] = df.tpep_pickup_datetime.dt.day
    df['pickup_hour'] = df.tpep_pickup_datetime.dt.hour
    df['pickup_period_of_day'] = pd.cut(
        df.tpep_pickup_datetime.dt.hour, [0,2,5,8,11,14,17,20,23], include_lowest=True,
        labels=['0_2','3_5','6_8','9_11','12_14','15_17','18_20','21_23'])
    df['pickup_day_name'] = df.tpep_pickup_datetime.dt.day_name()
    df['pickup__is_weekend'] = df.pickup_day_name.map(is_weekend_dict)
    # deleting the original field
    df.drop('tpep_pickup_datetime', axis=1, inplace=True)
```

Figure 5.4: Code used to extract new features from pickup-datetime feature

As can be seen in the figure, we extracted the day-of-the-month (i.e. 1,2,3,...,31) of pickup for taxi trips; we also extracted the period-of-day of pickup by dividing the day into periods of three hours each; many other features were extracted as illustrated in the figure. After the extraction is done, the original field, `tpep_pickup_datetime`, was deleted.

5.3.2 Adding Temperature Features

The temperature features available in the helper weather-datasets can be used to add new features to taxi-trips datasets. Figure 5.5 shows the code used to do so. We can see from the figure that a new feature named `day_avg_temperature` was added to describe the average temperature in the day of pickup of taxi trips. Also, another feature was added to specify the difference between the maximum and minimum temperatures in the day

of pickup of taxi trips. As can be seen in the figure, we add these temperature features by using the day-of-month of the trip as a link between taxi-trips datasets and weather datasets: for each trip, we take the day-of-the-month of the trip and look into weather datasets to get the temperature in that day and add it to the record of the trip in taxi-trips datasets.

```
for (df, w_df) in [(data, weather_data), (test_data, weather_test_data)]:
    day_temp = {}
    for d, t in zip(w_df.Date.dt.day, w_df['Average Temperature']):
        day_temp[d] = t
    df['day_avg_temperature'] = df.pickup_day_of_month.map(day_temp)

    day_temp_range = {}
    for d, mx_t, mn_t in zip(w_df.Date.dt.day, w_df['Maximum Temperature'],
                             w_df['Minimum Temperature']):
        day_temp_range[d] = mx_t - mn_t
    df['day_temperature_range'] = df.pickup_day_of_month.map(day_temp_range)
```

Figure 5.5: Code used to add temperature-features from weather datasets

5.3.3 Adding Precipitation Features

As with temperature features, we will use the precipitation features in weather datasets to add new features to taxi-trips datasets to help in improving predictive accuracy. Figure 5.6 shows the code used in that process.

```
for (df, w_df) in [(data, weather_data), (test_data, weather_test_data)]:
    day_prec = {}
    for d, p in zip(w_df.Date.dt.day, w_df['Precipitation']):
        day_prec[d] = p
    df['day_precipitation'] = df.pickup_day_of_month.map(day_prec)

    day_new_snow = {}
    for d, ns in zip(w_df.Date.dt.day, w_df['New Snow']):
        day_new_snow[d] = ns
    df['day_new_snow'] = df.pickup_day_of_month.map(day_new_snow)

    day_snow_depth = {}
    for d, sd in zip(w_df.Date.dt.day, w_df['Snow Depth']):
        day_snow_depth[d] = sd
    df['day_snow_depth'] = df.pickup_day_of_month.map(day_snow_depth)
```

Figure 5.6: Code used to add precipitation-features from weather datasets

From the figure we can see that we added features to specify the total precipitation, the new snow, and the snow depth in the day of each trip.

5.3.4 Adding Borough and Service-Zone Features

In taxi-trips datasets, there is a location ID specified for each trip as shown in Figure 3.2. As we described in 3.2, there is a helper table (see Figure 3.3) that specifies the borough and service zone of each location ID in addition to the name of the zone associated with the ID. Figure 5.7 shows how this table was used to add borough and service-zone features to the taxi-trips datasets.

```python
locID_borough_map = {}
locID_service_zone_map = {}
for loc_id, b, sz in zip(zones_lookup_table.LocationID.tolist(),
                         zones_lookup_table.Borough.tolist(),
                         zones_lookup_table.service_zone.tolist()):
    locID_borough_map[loc_id] = b
    locID_service_zone_map[loc_id] = sz


for df in [data, test_data]:
    df['pickup_borough'] = df['PULocationID'].map(locID_borough_map)
    df['dropoff_borough'] = df['DOLocationID'].map(locID_borough_map)
    df['pickup_service_zone'] = df['PULocationID'].map(locID_service_zone_map)
    df['dropoff_service_zone'] = df['DOLocationID'].map(locID_service_zone_map)
```

Figure 5.7: Code used to add borough and service-zone features using the helper location table

We can see from the figure that the pickup borough, drop-off borough, pickup service-zone, and drop-off service-zone were added to the taxi trips records.

5.3.5 Feature Encoding

By looking at Figure 5.8, we can see how our taxi-trips datasets look like after applying the previous steps.

VendorID	passenger_count	PULocationID	DOLocationID	trip_duration	pickup_day_of_month	pickup_hour	pickup_period_of_day	pickup_day_name	pickup__is_weekend	day_avg_temperature
1	1	48	163	4.150000	4	7	6_8	Saturday	Yes	-4.722222
2	1	136	247	8.150000	4	7	6_8	Saturday	Yes	-4.722222
2	5	238	132	31.850000	4	7	6_8	Saturday	Yes	-4.722222
2	1	163	236	2.200000	4	7	6_8	Saturday	Yes	-4.722222
1	1	68	234	5.216667	4	7	6_8	Saturday	Yes	-4.722222

day_temperature_range	day_precipitation	day_new_snow	day_snow_depth	is_passenger_count_gt_6	pickup_borough	dropoff_borough	pickup_service_zone	dropoff_service_zone
7.222222	0.0	0.0	0.0	0	Manhattan	Manhattan	Yellow Zone	Yellow Zone
7.222222	0.0	0.0	0.0	0	Bronx	Bronx	Boro Zone	Boro Zone
7.222222	0.0	0.0	0.0	0	Manhattan	Queens	Yellow Zone	Airports
7.222222	0.0	0.0	0.0	0	Manhattan	Manhattan	Yellow Zone	Yellow Zone
7.222222	0.0	0.0	0.0	0	Manhattan	Manhattan	Yellow Zone	Yellow Zone

Figure 5.8: A sample of taxi-trips datasets after applying feature engineering

We notice that many features have non-numerical values like the `pickup_borough` feature. To make our data suitable for all machine-learning models, we need to convert these features to numerical features. There are two cases regarding this conversion: 1) When the feature contains only two unique values. 2) When the feature has more than two unique values. Figure 5.9 explains what to do in the two cases with an example for each case.

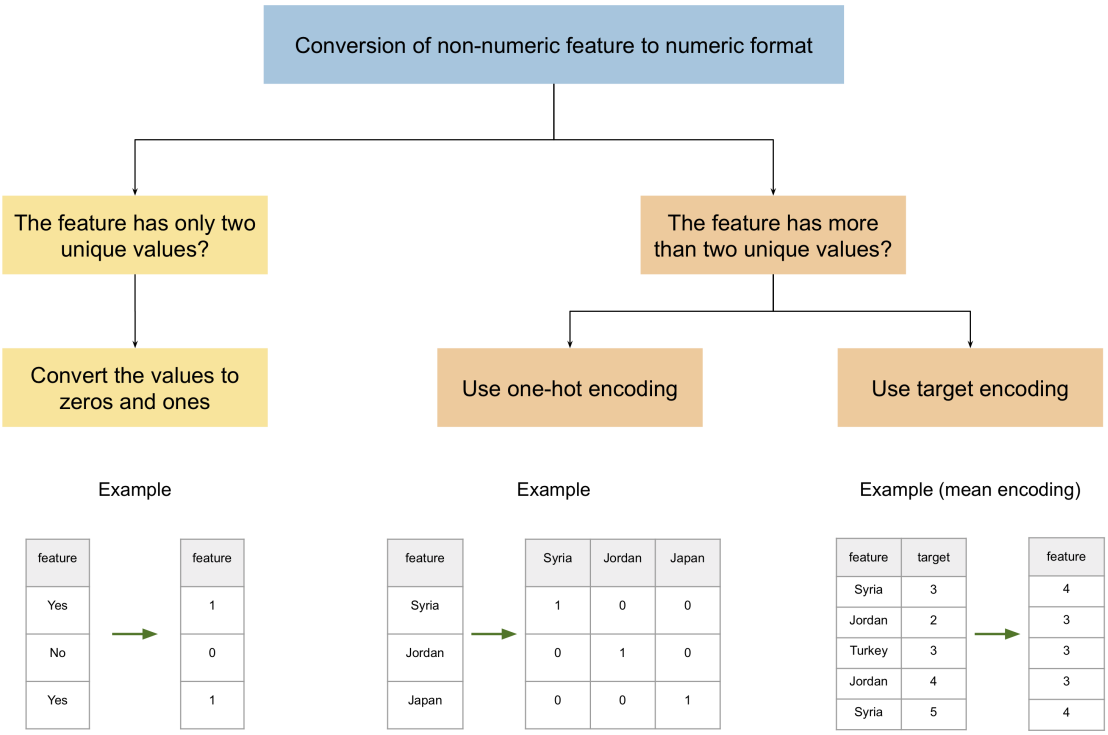

Figure 5.9: Feature conversion to numeric format

For features with more than two unique values, the general approach is to use one-hot encoding. For some features that are thought to be more important than others like features related to pickup and drop-off locations, we combine one-hot encoding and target encoding to distill all useful information from these features. Figure 5.10 shows part of the code used to convert non-numeric features to numeric format.

```
# columns to be used for one-hot encoding
cols_for_onehot_enc = ['VendorID', 'PULocationID', 'DOLocationID', 'pickup_day_name',
                       'pickup_borough', 'dropoff_borough', 'pickup_service_zone',
                       'dropoff_service_zone', 'pickup_period_of_day', 'pickup_day_of_month']

onehot_col_prefixes = {x:x for x in cols_for_onehot_enc}

# columns to be used for label encoding
cols_for_label_enc = ['pickup__is_weekend']

# columns to be used for target encoding later
cols_for_target_enc = ['PULocationID', 'DOLocationID', 'pickup_period_of_day', 'pickup_hour']

# columns for both one-hot encoding and target encoding
cols_targetend_and_onehot = [x for x in cols_for_target_enc if x in cols_for_onehot_enc]
```

```
# performing one-hot encoding and label encoding while keeping the
# columns that will be used for target encoding later

# one-hot encoding: training data
tmp_cols = data[cols_targetend_and_onehot]

data = pd.get_dummies(data, columns=cols_for_onehot_enc,
                      prefix=onehot_col_prefixes, prefix_sep='__')
for c in tmp_cols:
    data[c] = tmp_cols[c].values

# one-hot encoding: test data
tmp_cols = test_data[cols_targetend_and_onehot]

test_data = pd.get_dummies(test_data, columns=cols_for_onehot_enc,
                           prefix=onehot_col_prefixes, prefix_sep='__')

for c in tmp_cols:
    test_data[c] = tmp_cols[c].values

# label encoding
for df in (data, test_data):
    for c in cols_for_label_enc:
        df[c] = pd.factorize(df[c])[0]

display(data.shape, test_data.shape)
```

Figure 5.10: Code used to convert non-numeric features to numeric format

When applying one-hot encoding on the training and test datasets (i.e. `data` and `test_data` variables in code), we faced a complication: the number of features resulting

from applying one-hot enconding on a feature in the training dataset is sometimes different than the number of features resulting from applying one-hot enconding on the same feature in the test dataset. This happens when the unique values of the feature in the training dataset are different than those in the test dataset. Figure 5.10 shows the code used to deal with this problem and align the training and test datasets so they have the same features.

```
# align data and test_data (making sure they have the same
# columns) after one-hot encoding
missing_cols = set(data.columns) - set(test_data.columns)
for c in missing_cols:
    test_data[c] = 0

missing_cols = set(test_data.columns) - set(data.columns)
for c in missing_cols:
    data[c] = 0

# to ensure the same column order also
test_data = test_data[data.columns]
```

Figure 5.11: Code used to align the training and test datasets

There is another complication regarding one-hot encoding when we use it on a feature with a large number of unique values: the number of features resulting from the encoding will be large. For example, for `PULocationID`feature which specifies the ID of the pickup zone, it contains more than 180 unique values which means that one-hot encoding on this feature will produce more than 180 features. Due to this issue, the dimensionality of our data might become too large which makes it harder to process and analyze this data—this problem is known as the curse of dimensionality which is described by Verleysen and François (2005). To avoid this problem, we apply principal component analysis (PCA)—a dimensionality reduction technique—on features resulting from one-hot encoding when the number of unique values in the original feature is large. Figure 5.12 displays the code used to apply PCA on the one-hot-encoded features resulting from `PULocationID` and `DOLocationID` features.

```
if pca_on_zones:
    n_comp = 130
    locid_cols = [c for c in data.columns
                  if (c.startswith('PULocationID__') or c.startswith('DOLocationID__'))]
    tmp_data = data[locid_cols]
    data.drop(locid_cols, axis=1, inplace=True)
    pca = PCA(random_state=7, svd_solver='randomized', n_components=n_comp).fit(tmp_data)
    tmp_cls = ['PUDOLocIdPCA_{}'.format(x) for x in range(1,n_comp+1)]
    tmp_data = pd.DataFrame(pca.transform(tmp_data), columns=tmp_cls)
    data = pd.concat([data, tmp_data], axis=1)

    locid_cols = [c for c in test_data.columns
                  if (c.startswith('PULocationID__') or c.startswith('DOLocationID__'))]
    tmp_data = test_data[locid_cols]
    test_data.drop(locid_cols, axis=1, inplace=True)
    tmp_cls = ['PUDOLocIdPCA_{}'.format(x) for x in range(1,n_comp+1)]
    tmp_data = pd.DataFrame(pca.transform(tmp_data), columns=tmp_cls)
    test_data = pd.concat([test_data, tmp_data], axis=1)
```

Figure 5.12: Code used to perform PCA on one-hot-encoded location-ID features

5.3.6 Adding Origin-and-Destination-Boroughs Features

After performing feature encoding in the previous section, we are now able to add more features. Specifically, applying one-hot encoding on the `pickup_borough` and `dropoff_borough` features resulted in a binary feature (i.e. its value can be 0 or 1) for each borough as a pickup borough and as a destination borough as shown in Figure 5.13. So when the origin borough of a trip is Manhattan for example, a value of 1 will appear in `pickup_borough__Manhattan`.

pickup_borough__Brooklyn	pickup_borough__Manhattan	dropoff_borough__Brooklyn	dropoff_borough__Manhattan
0	1	0	1
0	1	0	1
0	1	1	0
0	1	0	1
0	1	0	1

Figure 5.13: Example of pickup_borough and dropoff_borough features after encoding

So using these encoded features, we will create features that specify both the origin and destination boroughs of trips by multiplying the values of the corresponding features. Figure 5.14 shows the result of applying this on the fields shown in Figure 5.13 for clarification. These featuures are called interaction features because they result from multiplying existing features and are expected to provide more useful information to the machine-learning models.

pickup_borough__Brooklyn_x_dropoff_borough__Brooklyn	pickup_borough__Brooklyn_x_dropoff_borough__Manhattan	pickup_borough__Manhattan_x_dropoff_borough__Brooklyn	pickup_borough__Manhattan_x_dropoff_borough__Manhattan
0	0	0	1
0	0	0	1
0	0	1	0
0	0	0	1
0	0	0	1

Figure 5.14: Example of origin-and-destination-boroughs features

Figure 5.15 shows the code used to create these origin-and-destination-boroughs features.

```
# features that indicate the origin and destination boroughs of the trip
for df in (data, test_data):
    cols_pickup_bor = [c for c in df.columns if c.startswith('pickup_borough__')]
    cols_dropoff_bor = [c for c in df.columns if c.startswith('dropoff_borough__')]
    for cp, cd in itertools.product(cols_pickup_bor, cols_dropoff_bor):
        df[cp + '_x_' + cd] = df[cp] * df[cd]
```

Figure 5.15: Code used to create origin-and-destination-boroughs features

5.3.7 Removing Features that Have Zero Variance

After performing the previous steps, we end up with more than 600 features. Some of them have zero variance which means they have the same value for all trips in the dataset.

This happens after one-hot encoding and feature interaction; for example, for the features created in 5.3.6, there might be no trip with EWR as the origin borough and Queens as the destination borough; thus, the pickup_borough__EWR_x_dropoff_borough__Queens feature value will be 0 for all trips.

Those features with zero variance don't introduce any useful information to the machine-learning model and thus should be deleted. Figure 5.16 displays the code used to delete them.

```
cols_with_one_val = []
for c in data.columns:
    if (data[c].unique().shape[0] == 1) and (test_data[c].unique().shape[0] == 1):
        cols_with_one_val.append(c)
data.drop(cols_with_one_val, axis=1, inplace=True)
test_data.drop(cols_with_one_val, axis=1, inplace=True)
```

Figure 5.16: Code used to delete features with zero variance

5.4 Model Selection and Hyperparameter Optimization

After applying feature selection and feature engineering, the data is ready to be used to build machine-learning models.

The outcome of this stage is a group of models that have better predictive performance on the project data than other models. We start with a set of candidate models; then we examine each model using *validation techniques* (more on that later) to get an idea about the model performance on our data; and finally, we select the best-performing models and further search the parameter space for the best combination of parameters for these models.

The candidate models—that will be examined to select the best of them—are: 1) LightGBM which is a gradient-boosting-decision-tree model (Ke, Meng, Finley, Wang, Chen, Ma, ... & Liu, 2017). 2) XGBoost which is also a gradient-tree-boosting model described by Chen and Guestrin (2016). 3) CatBoost which is also a gradient-tree-boosting model presented by Dorogush, Ershov, and Gulin (2018). 4) Random Forest model (Scikit-learn implementation) which combines multiple decision trees for more robust performance as described by Raschka (2015). 5) Neural Network model (Scikit-learn implementation). 5) Stochastic Gradient Descent linear model (Scikit-learn implementation). 6) Linear Support Vector Regression (Scikit-learn implementation). 7) K-Nearest-Neighbors Regression (Scikit-learn implementation).

To examine the potential of each of these models and to find the best hyperparameters of the chosen models, cross-validation will be used. We will apply cross-validation—specifically k-fold cross-validation—on the training dataset as follows—based on the description provided by Géron (2017): the training set will be randomly split into k folds (i.e. k parts) then for k iterations, one of the k parts will be used for model evaluation and the others for model training where a different part is chosen for evaluation each iteration. The evaluation scores of all iterations are averaged to get a score that represents the predictive performance of the model. Figure 5.17 shows a visualization of the k-fold cross-validation technique.

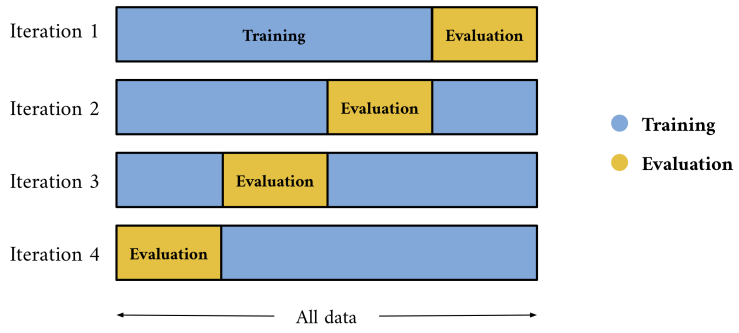

Figure 5.17: A visualization of k-fold cross-validation

The importance of using cross-validation for model selection and for finding the best parameters—and not evaluating the model directly on the test set—is because learning the parameters of a predictive model using a certatin dataset and then evaluating it on the same dataset is a mistake that leads to overfitting where the model would have a perfect score on that dataset but it would fail to generalize and generate accurate predictions on unseen data.

Figure 5.18 shows the code used to perform 2-fold cross-validation to examine the performance of LightGBM model. It can be seen from the figure that we have created a function `CV_helper` to make `KFold` tool, which is provided in Scikit-learn package, was used to generate cross-validation data indices (training and validation indices) in each iteration so we can create a training and validation sets to train and evaluate the model. We can notice also the `shuffle=True` parameter of KFold which shuffles the data before generating the indices. Moreover, it can be seen that there is a part of the code that performs target

encoding in each iteration as discussed in 5.3.5.

```python
def CV_helper(train_index, val_index):
    global data, cols_for_target_enc

    # creating datasets for cross validation
    X_tr = data.iloc[train_index]
    X_val = data.iloc[val_index]
    y_tr = data.trip_duration.iloc[train_index]
    y_val = data.trip_duration.iloc[val_index]

    # target encoding
    for col in cols_for_target_enc:
        means = X_tr.groupby(col).trip_duration.mean()
        X_tr[col + '_mean_encoded'] = X_tr[col].map(means)
        X_val[col + '_mean_encoded'] = X_val[col].map(means)

        medians = X_tr.groupby(col).trip_duration.agg(lambda ser: np.median(ser))
        X_tr[col + '_median_encoded'] = X_tr[col].map(medians)
        X_val[col + '_median_encoded'] = X_val[col].map(medians)

        stds = X_tr.groupby(col).trip_duration.std(ddof=0)
        X_tr[col + '_std_encoded'] = X_tr[col].map(stds)
        X_val[col + '_std_encoded'] = X_val[col].map(stds)

        mins = X_tr.groupby(col).trip_duration.min()
        X_tr[col + '_min_encoded'] = X_tr[col].map(mins)
        X_val[col + '_min_encoded'] = X_val[col].map(mins)

        maxs = X_tr.groupby(col).trip_duration.max()
        X_tr[col + '_max_encoded'] = X_tr[col].map(maxs)
        X_val[col + '_max_encoded'] = X_val[col].map(maxs)

        X_tr.drop(col, axis=1, inplace=True)
        X_val.drop(col, axis=1, inplace=True)

    X_tr = X_tr.drop('trip_duration', axis=1)
    X_val = X_val.drop('trip_duration', axis=1)

    target_endoded_vars = [x for x in X_val.columns if
                           x.endswith(('_mean_encoded', '_median_encoded',
                                       '_std_encoded', '_min_encoded',
                                       '_max_encoded'))]

    X_val[target_endoded_vars] = X_val[target_endoded_vars].fillna(-1)

    return (X_tr, y_tr, X_val, y_val)
```

```python
# LightGBM model

print(time.strftime('%H:%M'))

kf = KFold(n_splits=2, shuffle=True, random_state=7)

mae_cv = []

for train_index, val_index in kf.split(data.drop('trip_duration', axis=1)):

    # creating the datasets for CV with target encoding
    X_tr, y_tr, X_val, y_val = CV_helper(train_index, val_index)

    # creating and running the model
    train_data = lgb.Dataset(X_tr, label=y_tr)

    params={'learning_rate': 0.1, 'objective':'regression_l1',
            'metric':'', 'num_leaves': 31, 'boosting': 'gbdt',
            'verbose': 1, 'random_state':311, 'max_depth': -1,
            'bagging_freq': 0, 'bagging_fraction': 1.0,
            'feature_fraction': 1.0, 'min_data_in_leaf': 20,
            'num_threads': 8, 'verbosity': -1,
            'early_stopping_round': 15}

    num_round = 500

    light = lgb.train(params, train_data, num_round, valid_sets=[lgb.Dataset(X_val, label=y_val)],
                      valid_names=['validation'], early_stopping_rounds=15, verbose_eval=False)

    light_pred = light.predict(X_val)

    print('Iteration score: ', end='')
    print(mean_absolute_error(light_pred, y_val))

    mae_cv.append(mean_absolute_error(light_pred, y_val))

print('Average score across iterations =', np.mean(mae_cv))

gc.collect(); print(time.strftime('%H:%M'))
```

Figure 5.18: Code used to perform k-fold cross-validation with LightGBM model

To fine-tune the set of the best-performing models, we perform hyperparameter optimization. One of the ways to do that is by using `RandomizedSearchCV` method which is a tool provided in Scikit-learn package. It randomly picks a value for each parameter out of a predifined list of parameter values, then it uses cross-validation to evaluate model performance using the randomly-picked parameters. It repeats this process for a specified number of iterations and finally it outputs the combination of parameters that achieved the best performance. However, because we perform target encoding in each iteration of cross-validation as seen in Figure 5.18, we will not be able to use `RandomizedSearchCV`; instead, we will implement a similar method to fine tune our models. Figure 5.19 shows an example of this method for the LightGBM model.

```python
# LightGBM model

print(time.strftime('%H:%M'))

# the possible parameter values that we will use in searching
# for the best combination of parameters
parameter_space = {
    'learning_rate': [0.1, 0.05, 0.01, 0.005],
    'objective': ['regression_l1', 'regression_l2', 'huber', 'fair', 'poisson', 'mape'],
    'num_leaves': [31, 55, 77, 100, 150],
    'max_depth': [6, 7, 8, 9, 11, -1],
    'min_data_in_leaf': [20, 50, 100, 200, 500, 1000, 2000],
    'boosting': ['gbdt', 'dart', 'gbdt'],
    'bagging_freq': [0, 1, 2, 3],
    'bagging_fraction': [0.7, 0.8, 0.9, 1.0, 1.0],
    'feature_fraction': [0.7, 0.8, 0.9, 1.0, 1.0],
    'num_round': [300, 500, 700, 1000, 1200, 1500]
}

for i in range(30):

    print('### {} ###'.format(i+1).center(50))

    kf = KFold(n_splits=2, shuffle=True, random_state=7)

    mae_cv = []

    params={'learning_rate': np.random.choice(parameter_space['learning_rate']),
            'objective': np.random.choice(parameter_space['objective']),
            'num_leaves': np.random.choice(parameter_space['num_leaves']),
            'max_depth': np.random.choice(parameter_space['max_depth']),
            'min_data_in_leaf': np.random.choice(parameter_space['min_data_in_leaf']),
            'boosting': np.random.choice(parameter_space['boosting']),
            'bagging_freq': np.random.choice(parameter_space['bagging_freq']),
            'bagging_fraction': np.random.choice(parameter_space['bagging_fraction']),
            'feature_fraction': np.random.choice(parameter_space['feature_fraction']),
            'verbosity': -1, 'verbose':-1, 'random_state':311, 'early_stopping_round': 15,
            'metric':'', 'num_threads': 8}

    num_round = np.random.choice(parameter_space['num_round'])

    for train_index, val_index in kf.split(data.drop('trip_duration', axis=1)):

        # creating the datasets for CV with target encoding
        X_tr, y_tr, X_val, y_val = CV_helper(train_index, val_index)

        # creating and running the model
        train_data = lgb.Dataset(X_tr, label=y_tr)

        light = lgb.train(params, train_data, num_round, valid_sets=[lgb.Dataset(X_val, label=y_val)],
                          valid_names=['validation'], early_stopping_rounds=15, verbose_eval=False)

        light_pred = light.predict(X_val)

        print('Iteration score: ', end='')
        print(mean_absolute_error(light_pred, y_val))

        mae_cv.append(mean_absolute_error(light_pred, y_val))

    print(params)
    print('num_rounds:', num_round)
    print('Average score across iterations =', np.mean(mae_cv))

    gc.collect()

    print('='*60)
    print(time.strftime('%H:%M'))
    print('='*60, flush=True)

print(time.strftime('%H:%M'))
```

Figure 5.19: Code used for hyperparameter optimization for LightGBM model

After applying the procedures mentioned above to select the best models and optimize their hyperparameters, we end up with the set of models listed in Table 5.2. In the table, the models names and their optimized parameters are listed. Table 5.2 shows only the models selected for next stages. In addition to them, there are models that were excluded because of their inferior error score.

Table 5.2: Selected models, their optimized parameters, and their cross-validation MAE score

Model	Optimized Parameters

LightGBM	`# the optimized parameters`
	`params={'learning_rate': 0.05, 'objective': 'regression_l1', 'num_leaves': 100,`
	`'max_depth': 8, 'min_data_in_leaf': 20, 'bagging_freq': 1, 'bagging_fraction': 0.8,`
	`'feature_fraction': 0.8, 'verbosity': -1, 'verbose': -1, 'random_state': 311,`
	`'metric': '', 'num_threads': 2, 'boosting': 'gbdt'}`
	`num_round = 500`

XGBoost	`# the optimized parameters`
	`params = {'learning_rate': 0.1, 'eval_metric': 'mae', 'max_depth': 8, 'min_child_weight': 2,`
	`'subsample': 0.8, 'colsample_bytree': 0.8, 'colsample_bylevel': 0.9,`
	`'lambda': 1.0, 'alpha': 10.0, 'objective': 'reg:squarederror', 'nthread': 2,`
	`'verbosity': 0, 'seed': 3}`
	`num_round = 500`

5.5 Building an Ensemble Model

Using the output of the previous stage which is a set of models characterized with high performance on our datasets and their corresponding parameters (shown in Table 5.2), in this stage we will use these models to construct an ensemble model. Figure 5.20 shows a visualization of the ensemble model. In this ensemble model, we have three layers: the first layer works on the taxi-trips data; each model in this layer generates predictions that are fed to each model in the second layer; the second layer operates on the predictions generated by first-layer models; each model in the second layer also generates predictions that are fed to the third layer; the third layer consists only with an averaging unit which takes the predictions of second-layer models and average them to generate the final predictions.

This ensemble model is a stacking ensemble model which is used to enhance the accuracy and robustness of predictions. In the first layer, we used the LightGBM and XGBoost models which were listed in Table 5.2 since they perform best on the taxi-trips data. In the second layer, we used the Linear-Support-Vector-Regression model and another LightGBM model; we chose these two models after trying multiple models in this layer.

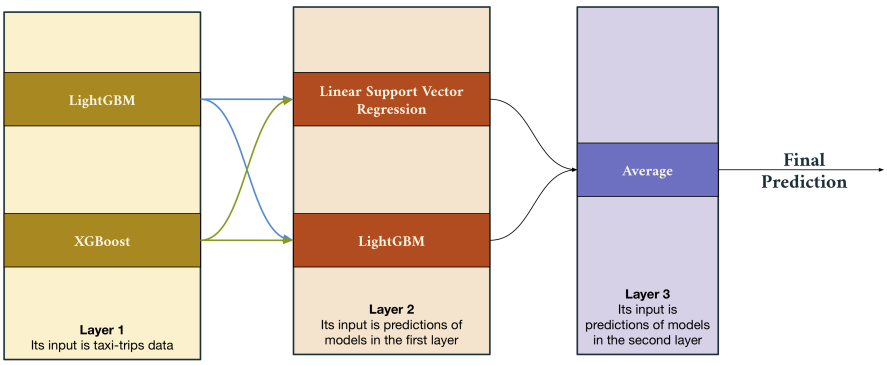

Figure 5.20: The operation of the ensemble model

5.6 Training and Evaluating the Final Model

The previous stage yielded the final model of this project. Now, we can train this model on the whole training data and then evaluate its performance on the test dataset which the model has never seen (since the model was built and tuned using cross-validation on the

training data); thus, this evaluation will also assess the generalization performance of the model. To illustrate the process that will be followed in training and evaluating the model, Figure 5.21 shows a diagram that summarizes the process.

To avoid overfitting or data-leakage problems when training the ensemble model, the following approach was followed: before starting, we divide our test data (March-2018 data) into two sets: a set (let us call it second-layer-set) used to train the second-layer models (as will be explained) and a set (let us call it final-test-set) to test the final model. The first step after that is training the first-layer models on the training taxi-trips data (March-2017 data). Then we generate predictions using these models the second-layer-set. After that, we train the second-layer models on the second-layer-set dataset which contains the predictions of first-layer models plus the target variable.

After applying the previous steps, the model becomes ready to be evaluated on the final-test-set as follows: firstly we generate predictions on the final-test-set using first-layer models. Then we generate predictions using second-layer models on the first-layer predictions. After that, we average the predictions of second-layer models to get the final predictions.

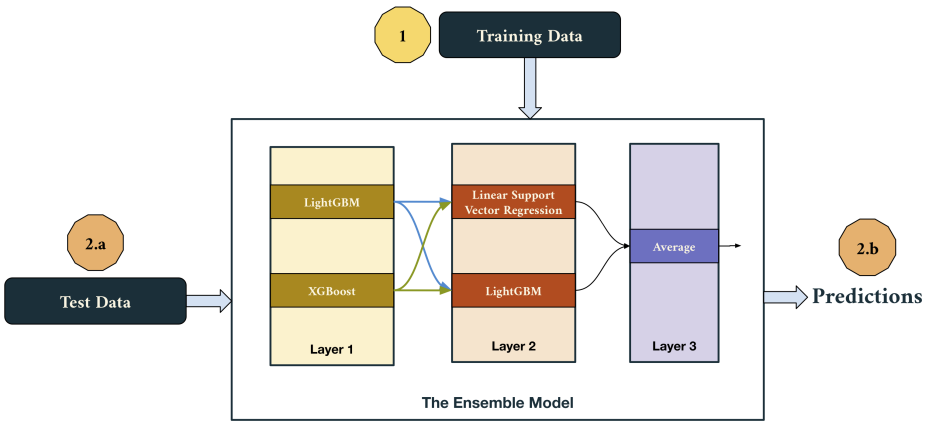

Figure 5.21: The steps of the process of training and evaluating the final model of the project

The evaluation of the ensemble model on the final-test-set gave a MAE score of **3.096** minutes. This means that our model can predict duration of taxi trips in New York City with an average error of 3.096 minutes. To put this MAE value into perspective, the

average trip duration in our taxi-trips data is ~14 minutes and 25% of trips with durations longer than 17.5 minutes. Thus, we can say that the model produced in this study has a good performance and can be used for taxi-trip duration estimation.

5.7 Feature Importance

LightGBM and XGBoost models provide a way to know the importance of features in our data. Figure 5.22 and Figure 5.23 show plots of feature importances for LightGBM and XGBoost respectively. The plots show the most important 15 features. The importance is calculated based on the number of times a feature is used in the model. Of course these feature importances are for the models of the first layer in the ensemble model, the models that work on the taxi-trips data.

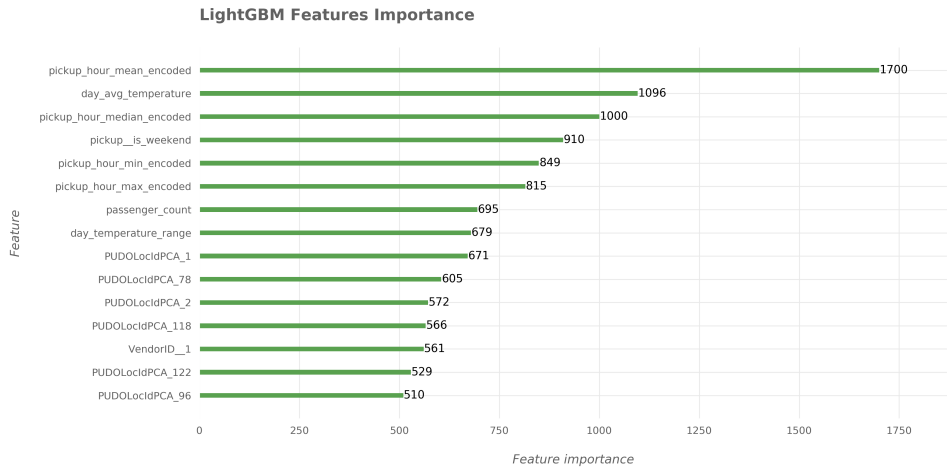

Figure 5.22: Feature importances for LightGBM model

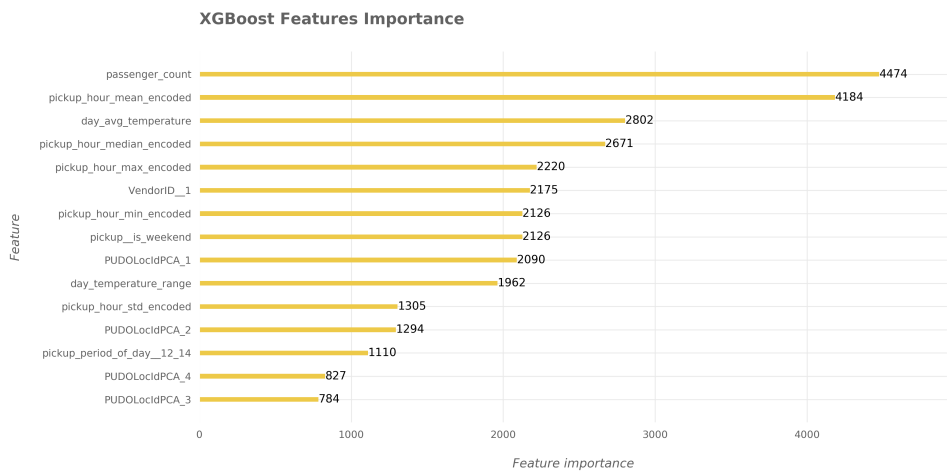

Figure 5.23: Feature importances for XGBoost model

We can notice that many features are considered among the most important features for both LightGBM and XGBoost like `pickup_hour_mean_encoded`, `pickup_hour_median_encoded`, `day_avg`, `temperature`, etc. We see two weather features in both plots; this indicates that using the weather datasets to improve performance was an effective idea.

5.8 Conclusion

In this chapter, we started with selecting the relevant and appropriate features from our data to be used in the modeling process such that no data leakage happen; then we applied feature engineering to extract as much meaning as possible from the available data to be utilized in building the model; after that, we inspected a set of candidate models to choose the best of them regarding performance and then we searched for the best hyperparameters of the selected models; in the next step, the optimized selected models were used to construct the final ensemble model which achieved an MAE of 3.096 minutes. This final model is now ready to be deployed in the next chapter.

CHAPTER 6: DEPLOYMENT

6.1 Introduction

In the previous chapter, we built the machine-learning model through many stages. The resulting model is able to predict the duration of taxi trips in New York City given some data about the taxi trip. In this chapter, we will deploy this model as a web application to demonstrate how the model can be used by anyone who wants trip-duration estimation.

6.2 Deployment Process

The goal of the deployment process is to make the machine-learning model developed in this study publicly available to anyone who wants to get an estimation of a taxi-trip duration. The model developed in this study needs specific input data related to a taxi trip in order to predict the duration of the trip. The user needs to submit this data to get the duration estimation.

To deploy our model, we will build a web application using a Python web framework called Flask. In other words, we will build a website on the Internet using front-end (HTML, CSS, JavaScript, etc.) and back-end (Python, Flask, Google App Engine, etc.) technologies. Any user can access this website through the website URL and interact with the predictive model in a user-friendly way.

6.2.1 Deployment to Google App Engine

The aforementioned web application will be developed on a local machine and then deployed to Google App Engine. Google App Engine is one of Google-Cloud-Platform tools; it provides a fully-managed serverless platform for developing and hosting web applications. To be able to deploy the web application, Google Cloud SDK needs to be installed on the local machine. Along with the Flask-application code and files, we deploy to Google App Engine a YAML file that contains the configurations for Google App Engine and a text file ("requirements.txt") that contains the names and the versions of the

packages and libraries used to develop the Flask application; these packages will then be installed on Google App Engine to run the application.

6.2.2 Usage and Deployment of the Predictive Model

We want to find a way to be able to use the machine-learning model developed in this study without the need to load all the data and train the model each time a user accesses the web application, especially that the data we used to train our models was large which is linked to very long model-training time. It is obvious that the web-applcation user would not wait for so long to be able to get the desired duration estimation. Thus, we need to be able to use our model without re-training it every time. Luckily, all the libraries we used in this study to construct the final ensemble model allow us to save the corresponding *trained* model as a file; this file can then be loaded inside the Flask application using the corresponding library; the machine-learning model can be re-constructed from that file and used for prediction without loading the data or re-doing the training process.

Other than the predictive models, we used principal component analysis and feature scalers during the construction of our model. As with the models, the PCA tool and the feature scaler need to be "trained" before they can be used. Similar to the case of the predictive models, we can save the trained PCA and scaler tools so they can be used inside the Flask application without loading data or re-training the tools.

6.2.3 Input Data of the User

For the machine-learning model to generate predictions, it needs to be provided with values for the same features used to train it in the first place. However, since the number of features used to train the machine-learning models in this study is large (217 features), we can't let the user of the web application enter the values of all these features. Because many of the features used in this study are constructed from other features, we can make the user enter a few values for a few features and then use these values to calculate the values for other features.

For example, we have a binary feature for each borough in New York City which specifies whether the trip originated from that borough or not; for example we

have the features `pickup_borough__Manhattan`, `pickup_borough__Queens`, `pickup_borough__Brooklyn`, etc. The value of each of these features can be 1 (to indicate that the corresponding borough is the pickup borough of the trip) or 0 (to indicate that the corresponding borough is *not* the pickup borough of the trip). So we can let the user choose the pickup zone of the trip and we use that to determine the pickup borough; then we make the value of the feature of that borough 1 and the values for other boroughs 0. The same principle can be applied to many features in this study.

6.2.4 The Application Interface

Figure 6.1 shows the interface of the developed web application. From the figure we can see that there are some fields the user needs to fill with information about the trip that the user wants to predict its duration like pickup zone and passenger count as shown in Figure 6.2. These values entered by the user are then used to calculate the values for all other features as described in the previous section. After that, the machine-learning model is used to generate prediction using features values and the prediction is sent and displayed to the user on the website interface as shown in Figure 6.3.

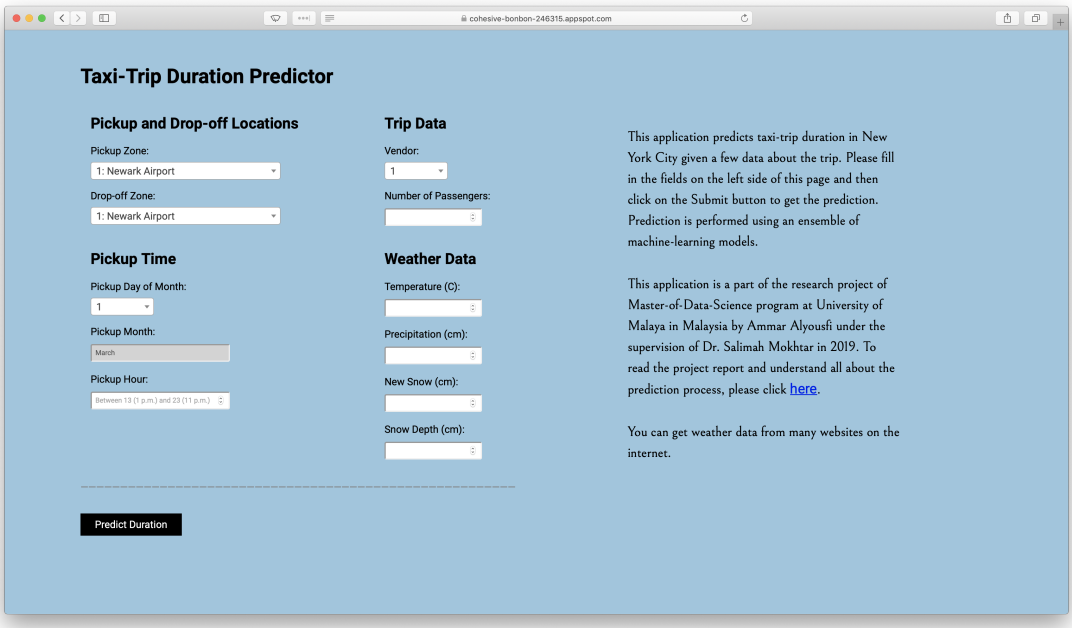

Figure 6.1: The interface of the Flask web application

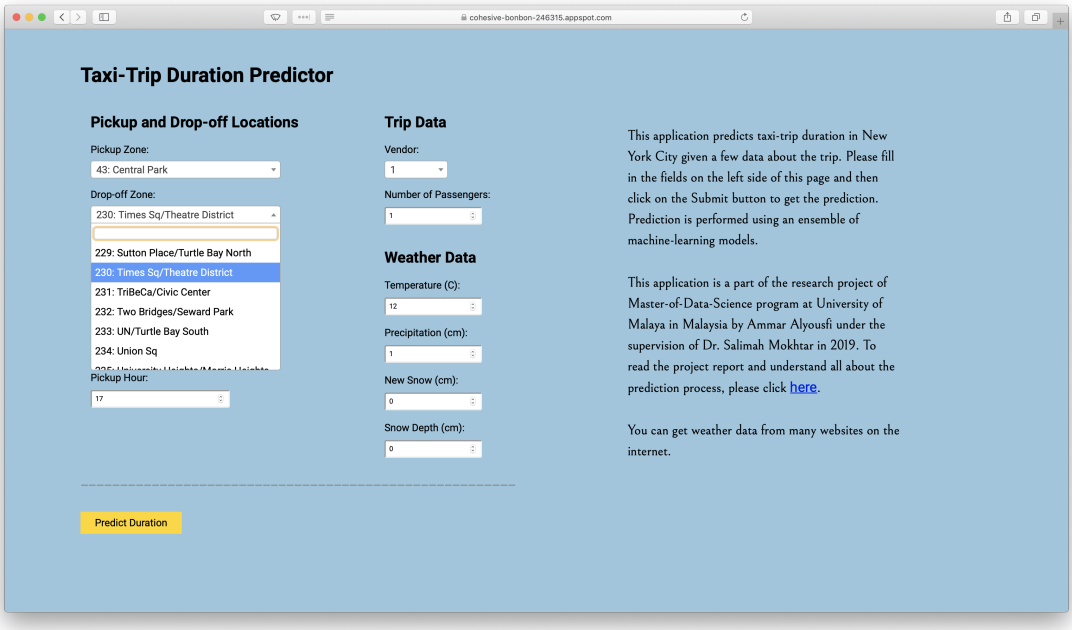

Figure 6.2: The user fills in the values for the trip

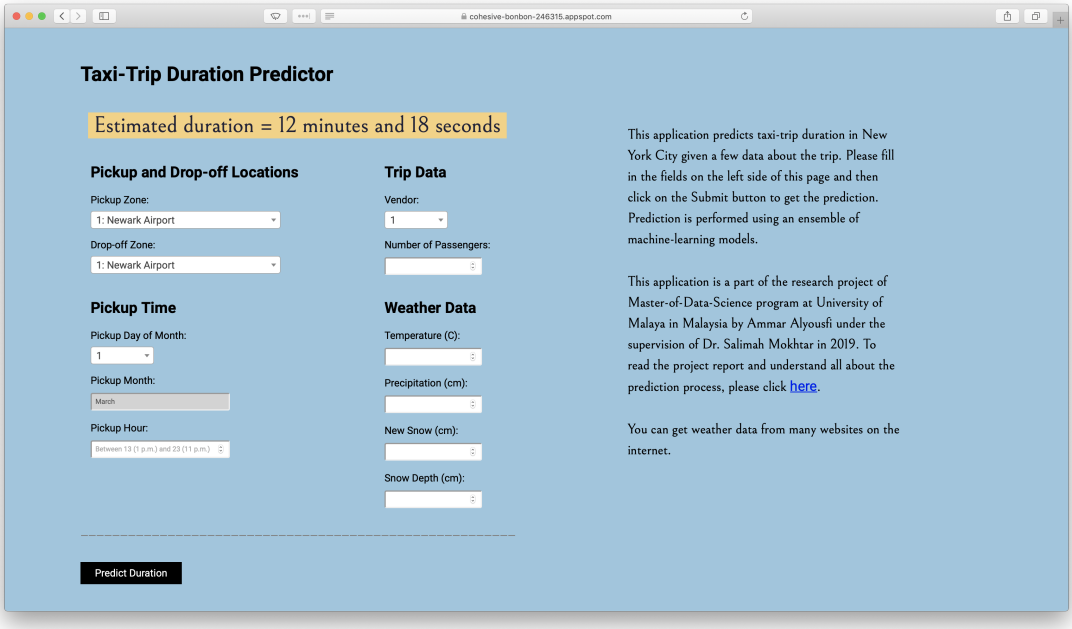

Figure 6.3: The prediction is calculated and displayed to the user

6.3 Limitations

As described in chapter 5, we reduced the size of the data to be able to complete the modeling stage due to the huge initial size of the data; we kept the data of the 1st, 4th, and 5thof March 2017 and March 2018 for the period between 13 (1 p.m.) and 23 (11

p.m.). Thus, because we trained the model on the data of these specific days and hours, we should use it to predict values for trips on the same days and in the same hours. For that reason, we constraint the user to enter values within these ranges. However, we can't say that the model predictions for values outside these ranges would necessarily be bad but we set those limits to be on the safe side.

6.4 Conclusion

In this chapter, we described the approach followed to deploy the machine-learning model of this study on the Internet to be publicly available for users who want to predict taxi-trips durations in New York City. We also specified the tools used in the deployment process and the interaction method between the user and the web application.

The application can be accessed on the following link: https://cohesive-bonbon-246315.appspot.com/. It will stay available at least until the end of October 2019.

CHAPTER 7: CONCLUSION

In this project, we built a machine-learning model to predict taxi-trip duration in New York City before the trip starts. We introduced an open-source and reproducible solution to the problem of trip-duration prediction using the machine-learning and data-science techniques. Moreover, we deployed the resulting predictive model as a universal web application to demonstrate how anyone can generate predictions quickly on the fly.

We started by defining the business problem to be solved; we identified its significance and scope also. After that, we looked in the literature at studies that shared the goal of our study to see how others approached this problem and what they achieved. Then we decided on the data that will be used and the methodology that will be followed in this project. After that, we started cleaning and preparing the data of the study to be ready for next stages; also, we applied an extensive exploratory data analyis on the data which led us to discover multiple error cases in the data and correct them. In the next stage, we began the implementation of the machine-learning models of the study following data-science principles that ensure the robustness and correctness of the developed model. At the end of this stage, we got a predictive model that is able to predict taxi-trip duration with a good accuracy. To make this model publicly available, we deployed it as an easy-to-use web application.

7.1 Limitations

During this study, we faced many limitations that will be described in this section.

7.1.1 Change in the Dataset of the Study

After starting this project, the organization which provides the datasets that we intended to use in this project decided to stop publishing the exact pickup and drop-off locations (i.e. langitudes and latitudes) of taxi trips. Instead, they started to publish the zone of pickup or drop-off. Previously, the exact locations were provided; they were used by authors of studies in the literature who used datasets from the same organization. This reduced the

amount of useful data that can be fed to the machine-learning model in order to generate more accurate predictions.

7.1.2 Computation Limitation

Due to the nature of the data of this study which describes taxi-trips in New York City, the number of data records is huge. We performed exploratory data analysis and data preparation on the data of March 2017 and March 2018 which contains around 20,000,000 rows. However, when we started the modeling process, we discovered the difficulty of dealing with this amount of data for machine-learning and data-transformation purposes. Even after we used a cloud machine with 52 GB of RAM, the difficulties didn't disappear. This led us to reduce the amount of data as described in the modeling chapter.

7.1.3 Time Limitation

The time that was given to implement and deploy this study (chapters 4, 5, and 6) was short (around one month); this prevented us from exploring more options that would take long periods of time. On a related note, running the code of this study took long time because it was dealing with a lot of data and many machine-learning processes. Given more time, we would be able to explore more advanced techniques and to incorporate more data in the process.

7.2 Future Work and Recommendations

The open-source machine-learning model produced in this study can be used by individuals and organizations who wish to use trip-duration estimation to manage their resources better. In this section, we will describe some operations that can improve the solution provided in this study.

In this study, the taxi-trips data of March 2017 was used to train a model that will be evaluated on March-2018 data. Choosing this setting was based on the assumption that the same month (i.e. March) has similar distributions and conditions in 2017 and 2018. However, using data of Febreuary 2018 might prove to be more useful than March 2017

to generate predictions for March 2018; this should be tested to verify. Moreover, in this study we used the data of one month (March 2017) to train the model; we expect that using data for more months (i.e. March 2016, March 2015, etc.) will improve the predictive ability of the model, although this would need high processing and memory capabilities.

Regarding data used in this study also, we used weather data to improve the accuracy of the model; we used daily weather data. However, to get more accurate predictions, hourly weather data can be used. More data can be useful for the model as well like data that specifies the dates of public holidays and special occasion because traffic tends to be affected in these dates.

As mentioned earlier, the programs produced in this study are all open source; thus, anyone can modify these programs to suit their own needs.

7.3 Study Code and Files

To access and download the code files (Jupyter Notebooks) and other files used in this study, one can use the following link to the project Github repository: https://github.com/ammar1y/Data-Science-Research-Project. All graphs and all the steps mentioned in this project including the deployment operations were performed using Python code and thus they can be reproduced easily using the code notebooks and files in the Github repository.

REFERENCES

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). ACM.

Cross-Industry Standard Process for Data Mining (CRISP-DM) [Diagram]. Retrieved May 19, 2019, from https://commons.wikimedia.org/wiki/File:CRISP-DM_Process_Diagram.png.

Dorogush, A. V., Ershov, V., & Gulin, A. (2018). CatBoost: gradient boosting with categorical features support. arXiv preprint arXiv:1810.11363.

Géron, A. (2017). Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

Gui, Z., & Yu, H. (2014). Trip travel time forecasting based on selective forgetting extreme learning machine. Mathematical Problems in Engineering, 2014.

Jindal, I., Chen, X., Nokleby, M., & Ye, J. (2017). A unified neural network approach for estimating travel time and distance for a taxi trip. arXiv preprint arXiv:1710.04350.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems (pp. 3146-3154).

Lam, H. T., Diaz-Aviles, E., Pascale, A., Gkoufas, Y., & Chen, B. (2015). (Blue) taxi destination and trip time prediction from partial trajectories. arXiv preprint arXiv:1509.05257.

Leys, C., Ley, C., Klein, O., Bernard, P., & Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. Journal of Experimental Social Psychology, 49(4), 764-766.

Li, Y., Gunopulos, D., Lu, C., & Guibas, L. (2017, November). Urban travel time prediction using a small number of gps floating cars. In Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (p. 3). ACM.

Nakata, T., & Takeuchi, J. I. (2004, August). Mining traffic data from probe-car system for travel time prediction. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 817-822). ACM.

National Weather Service (n.d.). Explanation of the F6 Preliminary Local Climate Data Report. Retrieved July 14, 2019, from https://www.weather.gov/grr/climateF6explain.

New York State Legislature (2019). NY State Vehicle and Traffic (VAT) Law: 1180-A Maximum speed limits. Retrieved July 20, 2019, from http://public.leginfo.state.ny.us/lawssrch.cgi?NVLWO:.

NYC OpenData. (n.d.). NYC Taxi Zones. [Image]. Retrieved from https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc.

NYC Taxi and Limousine Commission (2018). Data Dictionary – Yellow Taxi Trip Records [PDF file]. Retrieved May 19, 2019, from https://www1.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf.

NYC Taxi and Limousine Commission (n.d.). Taxi Fare. Retrieved July 27, 2019, from https://www1.nyc.gov/site/tlc/passengers/taxi-fare.page

Raschka, S. (2015). Python machine learning. Packt Publishing Ltd.

Sculley, D. (2010, April). Web-scale k-means clustering. In Proceedings of the 19th international conference on World wide web (pp. 1177-1178). ACM.

Singh, A. D., Wu, W., Xiang, S., & Krishnaswamy, S. (2015, October). Taxi trip time prediction using similar trips and road network data. In 2015 IEEE International Conference on Big Data (Big Data) (pp. 2892-2894). IEEE.

The SciPy Developers (n.d.). Scipy library main repository. Retrieved July 21, 2019, from https://github.com/scipy/scipy/blob/463ad5f/scipy/stats/stats.py#L2651-L2763.

Verleysen, M., & François, D. (2005, June). The curse of dimensionality in data mining and time series prediction. In International Work-Conference on Artificial Neural Networks (pp. 758-770). Springer, Berlin, Heidelberg.

Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. management Science, 17(11), 712-716.

Zhan, X., Hasan, S., Ukkusuri, S. V., & Kamga, C. (2013). Urban link travel time estimation using large-scale taxi data with partial information. Transportation Research Part C: Emerging Technologies, 33, 37-49.

Zhang, X., & Wang, H. L. (2011). Selective forgetting extreme learning machine and its application to time series prediction.