Forecasting Ride-Share Driver Demand Using Machine Learning

Dharnidhar Reddy Banala, Kalyan Vikkurthi, Manidedeepya Chennapragada and Nikhil Mylarusetty

Department of Applied Data Science, San Jose State University

Data 270: Data Analyst Process

Professor Dr. Eduardo Chan

May 17, 2023

Abstract

In the taxi industry, it is imperative to minimize wait times for both drivers and passengers through efficient taxi dispatching. This problem can be solved by optimizing the fleet allocation and making drivers aware of passenger locations. Related works have demonstrated different strategies, such as data-driven approaches and optimization algorithms, to improve taxi dispatching. Previous studies in the field of taxi dispatching have encountered limitations such as lack of data, poor data quality, and the absence of real-time data, which have caused the challenges in accurately estimating driver demand and optimizing taxi operations. This research paper collects real-time data on New York City taxi rides from a reliable online source using a robust data collection process, providing detailed and well-organized information. This study aims to develop advanced machine learning models to estimate driver demand in specific areas, considering factors such as time, day of the week and pickup locations. For this, a new target feature was created to analyze the pickup demand patterns. Random Forest Regressor, XGBoost Regressor, Facebook Prophet, and Long Short-Term Memory models are implemented, and compared the predictive effect of these models. The Random Forest Regressor outperformed when compared to the other regressor and time series models with an accuracy 90.4%. This enhanced efficiency enables ride-sharing companies to better allocate their resources, improve customer satisfaction, and create a more seamless and reliable transportation experience for all stakeholders involved.

Introduction

1.1 Project Background and Executive Summary

Efficient taxi dispatch can help drivers and passengers find each other faster. To accomplish this, ride-sharing firms and other transportation providers must precisely estimate the demand for drivers in various places and at various times. The Driver Demand Prediction problem statement is a machine learning task that requires an analysis of driver demand in a particular region or area using a collection of input data including each weekday time, activities, weather, and other variables that may affect demand. The significance of this issue stems from its ability to optimize the operations of ride-sharing firms and other transportation providers, resulting in better customer service and more income.

In this research study, four machine learning models are employed to address the Driver Demand Prediction problem. The first two models are the Random Forest Regressor and the XG-Boost Regressor, which are both popular and powerful machine learning techniques for regression, classification, and time series issues. The third model, Prophet, is a freely available software created by Facebook in 2017 and can be utilized in both R and Python programming languages Taylor and Letham (2018c). This technique is beneficial for the driver demand prediction problem since it can handle different seasonality and many other variables that help increase forecast accuracy. LSTM is the fourth model, a time series forecasting technique comparable to LSTM that supports other time series forecasting algorithms such as ARIMA and SARIMA.

The objective is to analyze and contrast the performance of four machine learning models for the Driver Demand Prediction problem. The motivation behind this research is to provide ride-sharing companies and other transportation providers

with accurate driver demand forecasts, leading to better customer service and increased revenue. The models are trained using historical data comprising details about previous driver demand and the necessary input attributes. The trained models are then used to estimate future driver demand, resulting in better-dispatching decisions.

The intended contributions of this research study are enhanced driver demand forecast accuracy and for the driver demand prediction problem, four machine learning models are compared. The optimization of the operations of ride-sharing firms and other transportation providers is one of the applications of this research that will enhance revenue and improve customer service. Furthermore, the insights of this study can be used to guide future research on machine learning models for the Driver Demand Prediction problem.

The target of this project is to improve vehicle supply prediction. This prediction problem is an important machine-learning task that can help ride-sharing services and other transport operators manage their operations. The Random Forest Regressor, XG-Boost Regressor, Facebook Prophet, and LSTM are four machine learning models investigated and compared in this research article for the Driver Demand Prediction problem. The goal is to provide realistic projections of driver demand, which will lead to better dispatching decisions and better customer service. Improved accuracy and a comparison of four machine learning models for the Transporter Supply Estimation problem are expected contributions of this research study, with implications for optimizing the operations of ride-sharing firms and other transportation providers.

1.2 Project Requirements

In this project, the complete focus is predicting the driver demand at a particular time, especially at peak hours, so that both the drivers and the customers will mutually benefit. The first and foremost requirement for this project is an environment to implement these

algorithms. So, based on the comparisons between the Jupyter notebook and Google Colab, it is evident that the processing speed (RAM) and there is also availability of GPU in the Google Colab. Moreover, this project involves the use of various libraries of Python such as scikit learn, numpy, pandas, scipy, etc.

Data Requirements

As for the dataset, in this project, the data comes from the official New York City website. This website provides information about different types of taxi data that are available in the NYC. It mainly provides information about three different taxi data which are then classified as Yellow, green, and vehicle hire taxis. Out of these, this project utilizes only yellow taxi data information which is very famous and has abundant data compared to other taxi data. The NYC website used different data compression techniques and stored the entire data in the parquet format. These parquet files are stored in a very organized manner and stored the data for each month. The size of each month's dataset contains around 2-3 GB. In this project, the data is collected for the first six months of 2023 and merged all dataset into a new final raw dataset. The sample of this rawd merged dataset in shown in Figure 1. In addition to this, website has the information about the coordinates and lookup files associated to the yellow taxi trips dataset and collected all this information as well. The NYC yellow taxi driver demand dataset contains details about taxi trips in New York City, such as pickup and drop-off locations, fare amounts, timestamps, and trip distances. As the dataset is extensive, it necessitates pre-processing and feature engineering before analysis can be conducted. After the conversion of the data, the main datatypes used inside the dataset are datetime, numerical, and categorical values. This needs to be converted for further exploration of the data. Supervised algorithms will be used involving random forest regressors, XG Boost regressors, Facebook Prophet, and LSTM, which are supposed to be conFigured. Supervised algorithms that involve a Random Forest Regressor, an XGBoost

Regressor, a Facebook Prophet, and LSTM. Since the dataset available from NYC is for each month, the final dataset needs to be merged and make a single standard dataset as a final dataset for the project which is known as data integration.

Figure 1

Sample Merged Raw dataset

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID
0	1	2022-01-01 00:35:40	2022-01-01 00:53:29	2.0	3.80	1.0	N	142	236
1	1	2022-01-01 00:33:43	2022-01-01 00:42:07	1.0	2.10	1.0	N	236	42
2	2	2022-01-01 00:53:21	2022-01-01 01:02:19	1.0	0.97	1.0	N	166	166
3	2	2022-01-01 00:25:21	2022-01-01 00:35:23	1.0	1.09	1.0	N	114	68
4	2	2022-01-01 00:36:48	2022-01-01 01:14:20	1.0	4.30	1.0	N	68	163
...	...	...	...	...	...	...	...	...	...
3558119	1	2022-06-30 23:45:51	2022-06-30 23:51:48	NaN	0.00	NaN	None	148	256
3558120	2	2022-06-30 23:25:00	2022-06-30 23:40:00	NaN	5.01	NaN	None	79	262
3558121	2	2022-06-30 23:29:00	2022-06-30 23:37:00	NaN	1.55	NaN	None	164	79
3558122	2	2022-06-30 23:24:15	2022-06-30 23:50:19	NaN	5.30	NaN	None	211	239
3558123	2	2022-06-30 23:33:53	2022-06-30 23:54:58	NaN	4.41	NaN	None	255	158

19816565 rows × 22 columns

Functional Requirements

Data Management. In terms of data management, the collected data was stored in the Google bucket for each month wise with the usage of the google cloud platform(GCP). The GCP is very popular and offers a wide variety of services that are deployed in the cloud. These services can be accessed anywhere, anytime from all over the world for more reliability and availability.

Data Security. To maintain security and authorization for the data, Different IAM roles were created for each individual in the project and granted access to these buckets to pull the data. These IAM roles are created and customized to set up the access for which user has which access and how much control they have. These IAM configurations are completely built on the GCP.

Error Handling. In order to organize the data Different error handling techniques were introduced at the script level and show the error message if unauthorized people access the data

AI Requirements

In this project, several libraries are used to develop and train the models. The two most important libraries used for the model deployment are Sklearn and Keras. The Sklearn module has different libraries and can be imported directly for model development. Four different models such as Random forest, LSTM, XG boost, and Facebook Prophet were imported and performed analysis on this.

Random forest regression(RF) is a very popular algorithm and is widely used by the industry. This is because RF follows ensemble learning techniques and is very good at handling the missing data and outliers which makes it more robust. XG Boost is also a popular one in the industry as this model also follows the ensemble techniques approach. The major difference between these models is it follows boosting technique while RF follows the Bagging approach. Facebook Prophet is another model which was imported from the SKlearn Library. This model is very good at predicting the target variables for time series data compared to other models. This is because the model architecture is divided in to different components and handles variety of data at each level such as Trend, Holiday, and seasonality. Keras was used to import the LSTM model which is also called Long Term Short Memory. LSTM follows the deep learning techniques while the other three models follow machine learning approaches. This model is very good at predicting the data because of its deep layers inside the architecture and the information is processed to these layers by removing the errors from the previous layer. In this project, collab was used as the final deployment as it has built-in CPUs . GPU's and RAM. It is true that training different models it requires a high amount of RAM, CPU's and GPU's for accurate predictions. In the future , if the projects require to be scalable at real-time data this can be easily developed by using the google collab pro which has more CPU's, GPU's, and RAM for an efficient processing system upon google servers.

1.3 Project Deliverable

For this project, the project proposal will be the starting phase, which will involve conducting a literature survey, defining the problem statement, and outlining the research objectives. This proposal will play a crucial role in providing guidance and direction for the project. The next step will be to develop a work breakdown structure (WBS) that breaks down the project tasks and assigns specific start and end dates to ensure effective project management and scheduling. Subsequently, an introduction report will be prepared, offering a comprehensive overview of the entire project. This report will cover important aspects such as the project background, requirements, deliverables, and a thorough review of relevant literature and technologies. Additionally, a data management plan and a project management plan will be created, focusing on data management procedures, project timelines, resource allocation, risk management strategies, and communication protocols. This plan will be ensuring efficient handling of data and optimal utilization of project resources to maintain a consistent time.

After the planning phase, to prepare the dataset for modeling the exploratory data analysis will be done in the data engineering process. Once the data is prepared, model development will commence, including the proposal, development, and evaluation of various machine learning models. This phase will include justifying the use of selected models, defining evaluation methods, and conducting validation experiments. The findings and insights obtained from the research will be shared through a final research presentation, providing a concise summary of the study's outcomes and implications. Furthermore, individual and group research reports will be prepared to thoroughly analyze the strengths, limitations, and overall effectiveness of the employed machine learning models. These reports will offer a comprehensive evaluation, shedding light on the research methodology and presenting a clear understanding of the research outcomes described in table 1.

Table 1

A Tabular View of Project Deliverables With Their Description and Due Dates

Deliverable	Description	Due Date
Project Proposal	Document with the research objectives, which involve conducting a literature survey, presenting background information, and defining the problem statement.	24-Feb-23
Work Breakdown Structure	Developing a hierarchical breakdown that outlines all the project tasks that need to be completed, along with the start and end dates for each task.	04-March-23
Introduction	A report comprising the project background, requirements, deliverables, technology and solution survey, and literature review of existing research.	10-Mar-23
Data and Project Management Plan	A comprehensive document that outlines the various aspects of data and project management, including data management procedures, storage and backup methods, project timelines, resource allocation, team roles and responsibilities, risk management strategies, and communication and reporting protocols.	31-Mar-23
Data Engineering	Perform exploratory data analysis on the dataset to ensure it is properly analyzed and processed, resulting in a high-quality dataset that can be effectively utilized in modeling. Through these data preparation steps, the aim is to enhance the accuracy and effectiveness of the models.	21-Apr-23
Model Development	Develop the models to utilized in the research project, including their proposals, supports, comparison and justification, evaluation methods, and validation and evaluation results.	5-May-23
Final Research Presentation	A PowerPoint presentation that effectively communicates the knowledge and insights obtained from the final research report.	12-May-23

Note. The table 1 is continued in next page

Deliverable	Description	Due Date
Individual Research Report	Create a report that analyzes each of the machine learning models utilized in the research project, outlining its strengths, limitations, and overall effectiveness in achieving the research objectives.	15-May-23
Group Research Report	Create a comprehensive and informative final report that summarizes the study and includes all necessary and relevant information. The report should be detailed and valuable, providing readers with a clear understanding of the research methodology and results.	17 May 2023
Prototype	A prototype of the ride-share engine is deployed which has maximum accuracy.	12-May-23

Note. Continuation for table 1

1.4 Technical survey and solutions

In the present research, algorithms will be used where there can be implemented ensembling techniques on the algorithms that have been selected. According to Vanichrujee et al. (2018), in this paper they used ensemble modeling using Recurrent neural network and XG-Boost. Further, the recurrent neural network involves methods like LSTM and Gated Recurrent Unit. The Datasets have been taken from a Thailand repository called Toyota Tsusho Electronic (Thailand) Co, Ltd. with 5000 taxis over a 4 month period.

The three methods are implemented individually across different areas including hospitals, subways, airports, and many more. However, the author has compromised on the number of features where there are only six features that are included in the dataset. The most interesting features in the above features are National Holidays but it is not clear how they have tackled the national holiday demand using RNN and XG Boost technique where that type of holiday is. Moreover, the weather conditions are also limited to just 2 cases i.e clear

and rainy. Although there is a lot of potential to anticipate the cab at the right time, this study project incorporates more features and weather conditions.

Following separate implementation of the algorithms, they compared using sMAPE (Symmetric Mean Absolute Error). In the case of High demand Areas, XGBoost outperformed both LSTM and RNN. However, combining them using ensemble techniques provided a better result. Similarly, the LSTM, Facebook Prophet and XgBoost would provide excellent results for taxi demand prediction.

The main challenge in predicting taxi demand is the dynamic nature of the data, influenced by factors such as weather, traffic, and events. Capturing temporal dependencies is one of the main tasks in this project where Xu et al. (2017) has proposed a model using long short-term memory (LSTM), which captures the dependencies and makes accurate predictions. So, they used LSTM as their algorithm because it can handle different data types. The authors collected taxi demand data from NYC over a period of 3.5 years and used the window approach to train their model. The model predicted taxi demand for the next 60 minutes based on the past hours of taxi demand data. The performance of their LSTM model was compared to baseline models using linear regression, and the outcomes revealed that the LSTM model performed better than the baselines.

The authors did not consider external features like weather, geographical locations, and special events which are the dynamic nature of taxi demand. And also, in this paper, they only compared with the baseline models and failed to compare their model with state-of-the-art models like ARIMA, SARIMA, Prophet, LSTM, and many other time series algorithms in terms of efficiency and accuracy. And their paper also fails to explain the hyperparameter tuning process, which affects their results.

In this project, finding seasonal patterns, temporal patterns, and trends is one of the major tasks as it is a time-series data. A paper by Boumeddane et al. (2021) was found for

this task, where they used ARIMA to capture the patterns and trends and give better accuracy for the model. They used information from a ride-hailing service company with operations in Algiers for this study. In their paper, they used the most efficient ARIMA model parameters using a method known as grid search, which involved figuring out the right sequence for the auto-regressive, differencing, and moving average components. Once they found their optimal parameters, they trained their model to predict the future. They evaluated the performances of their models using error metrics, MAPE and RMSE.

In addition to using a grid search approach to optimize the ARIMA model's parameters, the authors proposed a model stacking technique that involved combining the predictions of multiple models, including ARIMA, SVM, and LSTM, using a weighted average.

According to the results, the stacked approach was more accurate than the individual predictions, and ARIMA was the best algorithm out of the individual algorithms. But there are algorithms that can outperform ARIMA in terms of its effective seasonality pattern and trend analysis.

On the basis of this, it can be inferred how crucial it is to select an algorithm based on the seasonality and trends in the data. At that point, the Prophet algorithm was considered which may perform better than the ARIMA model. Prophet's ability to manage various seasonality patterns and its automatic detection and incorporation of holiday effects make this possible.

This research focuses on considering multiple factors that are responsible for the machine learning model such as temperature, weather conditions, peak hours e.t.c and also covers the numerous areas that are listed on the NYC official website. However, most of the research problems are based on measurements of high activity or considering very few environmental variables or climatic features. In Z. Liu, Chen, Sun, et al. (2020) study, using a

forecasting technique, several models such as combination forecasting model (CFM) ,ridge regression model (RRM), random forest model (RFM). They have downloaded the dataset from the Xi'an taxi management office where the data is generated every 5 seconds based on the number of trips that are generated in the area. The data has around 40 million track points, and this data has to undergo an extensive data cleaning process, and then they considered only error-free data. Out of all the factors they have considered, they realized that vital factors are only time and temperature. The entire research was based on the three models and compared the metric results of each model to identify the best-fit model for this approach. Initially, a random set of k feature characteristics is chosen from the available collection of attributes at each node of the decision tree. Subsequently, the most effective attribute from the subset is chosen to create a split. Secondly, the RRM model is used to analyze collinear data and it is the most advanced version of a technique that involves minimizing statistical inference of predicted data. Their problem in this type of scenario is that the model easily overfits when the sample has more features, and less sample quantity is used. Because of this, they also added the identity matrix for the existing equation to calculate the collinear data, which in turn gives us a high regression coefficient. According to the prediction outcomes, CFM demonstrates superior performance in both accuracy and robustness when compared to FRM and RRM, particularly in the Bell Tower areas.

Jamil and Akbar (2017) presented a unique approach for predicting the taxi demand in transportation systems. This approach involves using the automatic ARIMA model which will accurately forecast the areas where driver demand is likely to be high. The researchers gathered information from a well-known taxi enterprise website in China and conducted preprocessing by removing anomalies and consolidating the data into 30-minute intervals.Post which they proceed with the model selection as ARIMA and evaluate the model with the standard metrics such as MAE and RMSE. In the standard ARIMA

model, one needs to choose the p,d, and q parameters which represent the degree of differencing, the number of auto-regressive terms, and the number of moving averages respectively to determine which values are best fit for the algorithm. However, the automatic ARIMA model is capable enough to choose the best values for each parameter. The results of this study shows that it is very effective in predicting the customer hotspots with an accuracy over the 90%. The author also suggested the future work as to incorporate data from multiple sources and to explore more predicting models.

The study proposed several short-term prediction models and an ensemble data stream architecture to apply these models. The first models presented in the study were the Time-Varying and Weight-Varying Poisson Models, which adjust the Poisson distribution to account for time and seasonal variation in the rate of demand for taxi services. The ARIMA model was adopted as the third method because of its ability to detect autocorrelation in time series data, making it appropriate for short-term prediction tasks. The authors decided to use the ARIMA model because it is a well-known and frequently applied time-series forecasting model that has been successfully used in numerous real-world applications.

The authors wanted to evaluate the performance of their suggested models against the well-known ARIMA Model. Though they used streaming data for their study, the dataset used was only available for a month, which limited the ability to make longer-term predictions. Overall, the study provided insights into the use of time-series forecasting models for predicting taxi demand and demonstrated the potential benefits of ensemble data stream architectures in the field.

In the paper by Wang and Mi (2020), a comparison study of two well-liked time series forecasting techniques, ARIMA and LSTM, for anticipating the demand of vehicle sharing customers, demonstrating the significance of using hybrid algorithms. The authors collected data from a car sharing service provider in China and then preprocessed the data by

scaling and normalizing it, and split it into training and testing sets. They trained both ARIMA and LSTM models on the training set, using different combinations of input features and hyperparameters, and evaluated their performance on the testing set using error metrics.

The study's findings demonstrated that in terms of performance, the LSTM model outperformed the ARIMA model. In comparison to the ARIMA model, the LSTM model demonstrated reduced MAE, MSE, and RMSE values. The amount of previous time steps used as input had the biggest impact on the accuracy of both models, according to the authors' sensitivity analysis, which they performed to examine the impact of different input variables on forecasting performance.

Chen et al. (2020) proposed a novel approach for predicting taxi demand in the road areas by using the multi task learning (MTL) and a graph convolution network (GCN). The MTL approach is utilized to collectively train several related prediction tasks, such as peak demand detection and short- and long-term demand prediction. With this strategy, the model may use shared representations across several tasks, improving prediction accuracy. The geographical dependencies and connections between various road sections in the transportation system are recorded using the GCN layer. In order to incorporate network topology and spatial dependencies, authors used the graph convolution operation on the GCN layer to the road segment adjacency matrix. The authors collected data over a period of about 6 months and trained the model using a dataset from New York City which includes pickup, trip details, trip fare, drop off locations and timestamps. The model was evaluated with the multiple metrics that includes the standard MAE value, MSE value and also the peak hour detection accuracy. The results shows that the MTL_GCN model achieved better prediction performance when compared to the several baseline models of regression and deep learning models, especially in terms of the accuracy of predicting all the tasks and showed greater performance. Below table 2 provides comparison for different technology surveys.

Table 2

Comparison of Technical Survey

Authors	Technical Approaches	Models Used	Evaluation Metrics	Results	Conclusion
Vanichrujee et al. (2020)	Ensemble model combining RNNs and XGBoost	RNNs, XGBoost	Mean Absolute Percentage Error (MAPE)	MAPE: 12.53% for ensemble model	The ensemble model combining RNNs and XGBoost outperforms individual models for taxi demand prediction.
Xu (2022)	XGBoost algorithm	XGBoost	accuracy, Precision, Recall	accuracy : 89.7%	The XGBoost algorithm is effective in analyzing and identifying hot spot areas with high taxi passenger demand.
Chen et al. (2020)	Multitask learning and Graph Convolutional Networks (GCN)	Multitask learning, GCN	Mean Squared Error (MSE), R-squared (R^2)	MSE: 0.0175, R^2: 0.856	Multitask learning and GCN-based approach improves taxi demand prediction by considering the interdependencies among different
Boumeddane et al. (2021)	Model stacking approach combining multiple ML algorithms	Linear Regression, Random Forest, Gradient Boosting	Mean Absolute Error (MAE), RMSE	MAE: 22.17, RMSE: 28.53	The model stacking approach improves the accuracy of ride-hailing demand forecasting by combining multiple ML algorithms.
Z. Liu, Chen, Sun, et al. (2020)	Combination forecasting model using time series analysis and random forest	Time series analysis, Random Forest	MAE, Root Mean Squared Error (RMSE)	MAE: 18.32, RMSE: 24.15	The combination forecasting model incorporating time series analysis and random forest improves the accuracy.

Note. The table is continued in next page

Authors	Technical Approaches	Models Used	Evaluation Metrics	Results	Conclusion

Jamil & Akbar (2017)	Automatic ARIMA model	ARIMA	Mean Absolute Error (MAE), Root Mean Squared Error (RMSE)	MAE: 17.93, RMSE: 23.19	The automatic ARIMA model accurately predicts taxi passenger hotspots, facilitating efficient allocation of taxis.
Xu et al. (2018)	Streaming data analysis	Not specified	Root Mean Squared Error (RMSE)	RMSE: Varies based on the dataset	Streaming data analysis shows promising results for predicting taxi-passenger demand.
Wang & Mi (2018)	ARIMA and LSTM	ARIMA, LSTM	Mean Absolute Percentage Error (MAPE)	MAPE: 7.84% for LSTM, 8.62% for ARIMA	LSTM outperforms ARIMA in demand forecasting for car sharing users.

Note. Continuation for table 2

1.5 Literature Survey

According to Makridakis et al. (2022b) this paper presents a comprehensive comparison and analysis of forecasting methods in the fields of statistics, machine learning, and deep learning. They conducted an in-depth analysis of the relevant literature and evaluated the pros and cons of different approaches based on factors such as data size, complexity, and accuracy. They contrasted machine learning methods like random forests, gradient boosting, and support vector machines with more conventional statistical techniques like ARIMA, exponential smoothing, and state space models. They also talked about deep learning techniques like convolutional and recurrent neural networks. The authors highlighted the trade-offs between interpretability and accuracy among these methodologies and proposed ways of integrating multiple methods to enhance forecasting performance. This study offers insights into the advantages and disadvantages of various methodologies, as well

as future research directions, making it a significant resource for both academics and practitioners in the field of time series forecasting.

Rayle et al. (2016) did a study on the forecasting of driver demand in which they evaluated the effectiveness of ride-hailing services in San Francisco in comparison to taxis and public transportation. The data is collected through a survey by questioning different types of questions such as frequency of travel, the reason for travel, and mode of travel for around 380 participants in the San Francisco area. They have gone through the reports and evaluated by summarizing the information provided by the users and understand which travel mode is highly used and different perceptions of the transportation system. The authors found that ride-hailing services were more efficient in meeting customer demand, thanks to their real-time response capabilities. But for this research paper, they did not use any kind of models to evaluate the report instead the assessment was done by going through the reports. However, the authors also identified that ride-hailing services were susceptible to surges in demand, which could lead to increased waiting times for customers.

The study conducted by Silveira-Santos et al. (2022) on the Facebook Prophet model for time series forecasting to examine how the COVID-19 epidemic affected ride-hailing rates in Atlanta and Boston. The authors used the model to forecast ride-hailing demand and fares for the cities, taking into account the impact of the pandemic on travel patterns and behaviours. They found that ride-hailing fares decreased significantly during the pandemic period, particularly during the lockdown periods when travel restrictions were in place. The authors noted that the use of the Facebook Prophet model allowed them to capture nonlinear data relationships and produce accurate predictions, providing insights into the pandemic's effects on the ride-hailing sector.

In this research study by Stefanon et al. (2021), the authors present a novel approach to enhance the accuracy of hourly electricity spot price forecasts in the Italian market. Their

methodology combines two techniques, namely Prophet and Seasonal Trend Decomposition, to achieve this goal. The proposed approach employs a time series decomposable model, which consists of three key components, namely trend, seasonality, and holidays. Additionally, Seasonal Trend Decomposition, a conventional time series decomposition method, is used to partition the time series into seasonal, trend, and residual elements. Initially, the authors remove the trend and seasonality components from the original time series using Seasonal Trend Decomposition, thereby retaining only the residuals. The Facebook Prophet technique was then employed to forecast the residuals, and the predicted values were merged with the trend and seasonal components to generate the ultimate forecast. The research findings demonstrate that the proposed approach outperformed other modern forecasting systems in terms of precision. Moreover, the authors highlight the importance of utilizing multiple techniques to enhance forecasting accuracy, as various methodologies may capture distinct facets of the data.

Kankanamge et al. (2019) proposes the use of Isolated XGBoost Regression for predicting taxi trip travel time. The authors collected a dataset containing various input parameters such as pick-up and departure points, trip distances, weather data, and traffic data. They employed XGBoost, a powerful machine learning technique, to build a regression model that forecasts journey durations based on these input features. To enhance the model's accuracy, the authors devised an Isolated XGBoost approach that trains separate XGBoost models for each input feature and combines them into a single model. The study's results demonstrate that the suggested technique outperforms existing XGBoost models and provides highly accurate predictions of taxi trip travel times.

Carson-Bell et al. (2021) proposes the use of ensemble learning methods, specifically Random Forest, to predict ride-hailing demand at pick-up locations. The authors acknowledge the challenge in forecasting demand patterns, which is critical in ensuring

customer satisfaction, increasing revenue, and optimizing fleet utilization. The researchers pre-processed a dataset of pick-up locations and demand data from a ride-hailing company for training and testing purposes. They assessed the effectiveness of different machine learning methods like Random Forest, AdaBoost, and Gradient Boosting. The outcomes showed that Random Forest, with a root mean square error (RMSE) of 0.073, had the best prediction accuracy. The study also demonstrated that ensemble learning approaches further improved the model's performance, resulting in an RMSE of 0.063. The findings suggest that using Random Forest and ensemble learning techniques can be a viable solution to accurately forecast ride-hailing demand, enabling businesses to enhance their operations and customer satisfaction.

The research by Gupta et al. (2018) recommends using Gradient Boosting and Random Forest machine learning models to forecast the length of a taxi ride. The authors collected data from the NYC TLC, which included pickup and drop-off locations, trip distances, and meteorological data. They pre-processed the data and performed feature engineering to create input features for the models. The Random Forest and Gradient Boosting models were then trained on the dataset to forecast cab trip time. An ensemble-based approach was employed to combine the results of both models to improve the forecast accuracy. The study found that the ensemble-based method outperformed both individual models, with a MAPE of 7.18%. Furthermore, the study compared the performance of the ensemble-based approach with other machine learning models, including Multi-Layer Perceptron and Support Vector Regression, and demonstrated that the most accurate ensemble models for predicting cab trip time were Random Forest and Gradient Boosting. Overall, the study highlighted the effectiveness of using an ensemble-based method with Random Forest and Gradient Boosting models for accurate cab trip time prediction.

Xu et al. (2018b) proposed a method to improve the performance of XGBoost by collecting data from peak hours in Chengdu for 20 days and using DBSCAN Clustering to identify hotspot areas based on actual distance instead of Euclidean distance. The author tested XGBoost with and without Fourier eigenvalues on the identified hotspot areas, and found that the model performed better with Fourier eigenvalues. These findings suggest that incorporating Fourier eigenvalues can enhance the performance

The paper by Zhao et al. (2020) presents a novel approach to predicting taxi passenger demand by integrating data from both Uber and taxi services. Two deep learning models, USTN and STIMN, were developed to leverage both datasets and improve prediction accuracy and robustness. USTN pretrains a deep neural network on Uber data, which is then fine-tuned on taxi data, while STIMN predicts passenger demand for both taxi and Uber by utilizing a shared feature extraction layer. The authors compared their deep learning models to the traditional ARIMA model and demonstrated their superiority in prediction accuracy. In summary, the paper showcases the potential of deep learning models and multi-source data integration for enhancing taxi passenger demand prediction.

Poongodi et al. (2021) in their paper performs a method for predicting the duration of taxi trips in New York City using two machine learning methods (MLP and XGBoost). The authors aim to assist taxi drivers in optimizing their routes and reducing customer wait times by providing accurate trip duration predictions. They collected data on various factors that can affect travel times, such as pick-up and drop-off points, the weather and traffic congestion. The researchers evaluated the model's effectiveness using several metrics, such as MAE and RMSE, and found that XGBoost outperformed MLP in accurately predicting taxi trip durations. This study suggests a potential strategy for using machine learning algorithms to estimate taxi travel times in New York City, which could increase the efficiency of taxi services and improve customer experience.

Askari et al. (2020) presents a novel approach to predict taxi demand by integrating Points of Interest (POI) data into a LSTM model. This model can learn the temporal relationships in sequential data. By analyzing historical taxi demand data and POI attributes, such as the presence of restaurants or retail complexes, the model can predict taxi demand for the next time period. The authors evaluated their proposed model using data from New York City and compared it to other existing machine learning methods. Regarding accuracy of forecasts, the model which utilized LSTM and incorporated POI information demonstrated superior performance in comparison to other models. Additionally, the authors examined the impact of various POI categories on the model's efficacy, and identified specific categories, including airports and tourist attractions, which had a notable impact on the prediction of taxi demand. In conclusion, this study suggests a promising approach to enhance taxi demand prediction accuracy by incorporating deep learning and POI data.

Naji et al. (2021) presents a new method for forecasting taxi demand by utilizing GANs (Generative Adversarial Networks) which use data from multiple sources.. The authors discuss how GANs can produce realistic taxi demand data samples by combining various sources of information, such as weather and events data. They also mention the potential use of GANs in producing synthetic data for other transportation-related applications. The paper suggests that using GANs with data from multiple sources is a promising approach for forecasting taxi demand in the industry. Overall, the paper introduces an innovative method for taxi demand forecasting, which has the potential to enhance accuracy and optimize fleet management in the transportation industry.

The paper by Z. Liu, Chen, Sun, et al. (2020) discusses the challenge of supply-demand imbalance in public transportation services and how forecasting online taxi-hailing demand can help alleviate this issue. To estimate demand for online taxi-hailing, the authors offer six models based on backpropagation neural networks with extreme gradient boosting.

The models perform better at making predictions when additional data is added, according to the authors. The study provides a methodology for estimating demand for online real-time taxi hailing that takes into account information about expected taxi demand. The results indicate that using XGB models is more effective than BPNN models and that incorporating additional data into the models improves their accuracy. Below Table 3 provide information about comparision of different literature surveys.

Table 3

Comparison of Literature Survey

Paper	Year	Venue	Methodology
Vanichrujee et al. (2020)	2020	IEEE Conference Publication	Ensemble Model based on RNNs and XGBOOST
Xu (2022)	2022	IEEE Conference Publication	XGBoost Algorithm
Chen et al. (2020)	2020	IEEE Conference Publication	Multitask Learning and GCN-Based Taxi Demand Prediction
Boumeddane et al. (2021)	2021	IEEE Conference Publication	Model Stacking approach using Random Forest, Linear Regression and Gradient Boosting
Liu et al. (2020)	2020	Journal of Advanced Transportation	Facebook Prophet and Random Forest
Jamil & Akbar (2017)	2017	International Conference on Science in Information Technology	Automatic ARIMA Model
Xu et al. (2018)	2018	IEEE Journals & Magazine	IEEE Transactions on Intelligent Transportation Systems
Wang & Mi (2018)	2018	IEEE Conference Publication	Comparative Study of ARIMA and LSTM
Chen et al. (2020)	2020	IEEE Conference Publication	Multitask Learning and GCN-Based Taxi Demand Prediction

Data and Project Management Plan

2.1 Data Management Plan

Data Collection Approaches

New York City is renowned for its energy, diversity, and bustling populace, making it a perfect spot for data collection and analysis. The data for this project is collected from NYC

TLC (Taxi and Limousine Commission) record Data. The NYC website Figured out the various types of taxis based on its economy and social factors and the three categories include Green Taxi, Yellow Taxi and For-Hire Vehicles.

Each taxi category is unique in its own way. Firstly, the process of For-Hire Vehicles (FHV) transportation is run by a limo company. This process happens through prior booking such that it is easy to pick the passengers as the driver can reach out to the destination without any hassle in the scheduled time. Next, the green taxis are in the domain of hybrid mode with the help of Street Hail Livery (SHL) where it can prearrange the trip and also pick up passengers through street hails but with limited scope. Finally, coming to Yellow taxi which is more unorganized than the other two where large chaos has been happening with their businesses due to unavailability of drivers at peak time as yellow taxis only take trips through street hailing.

For the present Taxi Based demand Prediction the data is confined to yellow taxi dataset because of its unorganized nature. In this research, SOAP API'S are used to gather and download the data from the official NYC website. Initially the raw data from this website is in the form of parquet files. The reason NYC official website is storing the data in the form of parquet files is because of its convenience in handling the large amount of data and ready to use in today's industry. Moreover the data in the parquet files is suitable and more embedded to work with cloud Technologies due to its columnar data storage. For the project to have deeper or extensive analysis data will be converted from parquet files to CSV Format with the help of APACHE PYSPARK. This will subsequently help us to draw meaningful insights to predict the taxi demand at required areas.

This CSV (Comma Separated values) text file that is generated from parquet files contains column data like drop-off and pick-up locations, fare amounts, trip distances, and trip distances etc with a total of 19 features. Periodically the data is changed on the website,

So according to that change a new column airport_fee is being added to the dataset for the year 2022 and 2023. Even Though there are various types of dataset related to taxi business, we are confined to NYC taxi data as the data is enormous and not reusing any existing data for our project.

Every month nearly 2.5 GB of data is generated in the yellow taxi domain. So for a year it is about 30 GB of Data. To preserve the data from the disasters a backup plan has been engaged where the same data is being replicated across multiple servers specifically in Los Angeles and Texas. All this data is stored in Google cloud platform which is secure and flexible to create permissions for accessibility by using IAM role. This can help us to ensure long term access to data which can be achieved through cost effective process and efficient analysis of large datasets.

Data Storage and Management Methods

In this project, the data for the year 2022 has been taken and the for recent analysis of taxis. It is very predictable that data might shoot up at a rapid phase during various festivals and holiday seasons in that particular month. To maintain Scalability the storage on the Google Cloud Platform must be three times the present usage of data storage. So initially the Google cloud storage is decided to be 25GB.

Moreover, to maintain the data during the disasters it should be stored in various locations to avoid data missing due to natural or man-made calamities. The data is stored at two different locations in the United States. The first copy of the dataset is located in Los Angeles, California. The second copy is stored in Texas, Dallas which will be used for both data availability and disaster recovery. We have prepared the python script in such a way that data is gathered from the website and downloaded as the parquet file for each month as shown in the below Figure 2. The file name is self explanatory and can be considered as a meta data which shows information like year, month and type of taxi data we collected. The

below structure explains the information about the parquet files that we collected form the NYC website.

Figure 2

Overview of the Data

	file_name	file_size_bytes	num_rows	num_cols
0	yellow_tripdata_2022-12.parquet	53640739	3399549	19
1	yellow_tripdata_2022-10.parquet	57061938	3675411	19
2	yellow_tripdata_2022-11.parquet	50106631	3252717	19
3	yellow_tripdata_2022-09.parquet	49619957	3183767	19
4	yellow_tripdata_2022-08.parquet	49717159	3152677	19
5	yellow_tripdata_2022-07.parquet	49367712	3174394	19
6	yellow_tripdata_2022-06.parquet	55365184	3558124	19
7	yellow_tripdata_2022-04.parquet	55222692	3599920	19
8	yellow_tripdata_2022-05.parquet	55558821	3588295	19
9	yellow_tripdata_2022-03.parquet	55682369	3627882	19
10	yellow_tripdata_2022-02.parquet	45616512	2979431	19
11	yellow_tripdata_2022-01.parquet	38139949	2463931	19

Note: Parquet files data from NYC are collected with their corresponding metadata.

Similarly, the data is stored in the Google cloud platform in a very organized and structured way which is convenient for people who have access to this data and understand the data. Since the data is already divided in terms of months for every year and is stored in the similar fashion as 12 CSV files segregated into a single folder with the year. In this project IAM roles are used for the security and appropriate permission is given to each team member according to their roles and responsibility. If any new changes or permission are added, users should request access for the admin and grant the access up on the approval. The below Table 4 structure defines the storage mechanism developed for the taxi data in google cloud platform.

Table 4

Structure of the folder and Name Conventions

Purpose	Folder Name	File Name
Main Folder	NYC TAXI DATASETS	
Year wise Data	YEAR 2022	yellow_tripdata_<YEAR>-<MONTH>.csv

		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
		yellow_tripdata_<YEAR>-<MONTH>.csv
Test and train data	Train_test_data	Train_data.csv Test_data.csv

Note: <YEAR>-<MONTH> are the specific year and month of the yellow_tripdata

Data Usage and Mechanisms

The Data that has been used in the project (NYC yellow taxi Dataset) is a public dataset which is accessible to everyone. However, the dataset is being updated every month on the website with addition of new columns to the dataset. The project is completely pushed into the GitHub Repository with proper documentation of dataset and explanation of code. In terms of data security, that has been refined through the EDA process is provided with a link in the github with an SSH linked to it.

Moreover to cut the cost of the Google Cloud Storage the data that is stored in the GCP will be erased after all the submissions are completed. All the datasets that are stored for the whole year will be removed from the Google cloud storage. This will reduce the cost which is allocated for the resource and can be easily achieved by deleting the buckets in the Google Cloud Platform.

2.2 Project Development Methodology

For this project CRISP–DM methodology is being implemented for the project development. The below Table 5 gives an overview of the each members allocated work.

Table 5

Responsibility and Resource Allocation for DMP

DMP Phase	Resource Allocation	Work Assigned
Data Collection	Kalyan	Python script is developed to gather the convert parquet files to CSV files using Apache spark
	Dharnidhar	Mapping between the location lookup table and the converted CSV
Documentation and Metadata	Manidedeepya	Documneted the details of methodolgy used and assumptions considerd on this data
Ethics and legal compliance	Nikhil	All softwares used in this project are been taken from the offical websites with registration of proper licensed keys in order by agreeing to their terms and conditions.
Storage and Backup	Dharnidhar	Data is stored in the Google Cloud Platform and for recovery data is replicated across multiple locations
Selection and preservation	Manidedeepya	Data which is frequently used and with the highest importance should be preserved in multiple locations
Data Sharing	Kalyan	All the data and code will be uploaded on to github repository with appropriate permissions

Business Understanding

The business understanding phase is critical in defining the problem statement and project objectives. Initially, for the project idea we have gone through various research papers for the project objective. After extensive research , the goal is to identify the most appropriate models to analyze existing taxi data and provide insights in such way to predict the taxi demand area that can be used to improve the performance of the taxi business. This can be achieved by exploring various technical papers that include ieee papers, springer etc which is based on taxi demand using machine learning and deep learning.moreover studying their literature surveys also gives us a clear insight how to proceed with the objectives , algorithms and improvements. The objectives include understanding the taxi industry, analyzing the data, and using appropriate models to optimize the business. By conducting research on existing taxi data, including factors such as pick-up and drop-off locations,

passenger demographics, travel time, distance, and fare, machine learning algorithms can be used to construct predictive models that can forecast demand, optimize routes, and reduce wait times. In addition to this, the aim is to determine the demand areas for taxis. Next, the planning of project starts which includes wbs chart, pert chart and gantt chart. All these are helpful to determine and direct the project in right direction.

In the case of the taxi sector, this entails learning about the industry, assessing existing taxi data, and determining the best models to enhance business performance. Regression, clustering, and classification models can be explored to identify patterns, make predictions, and generate insights. The ultimate goal is to provide actionable insights that can help taxi companies make data-driven decisions to increase profits, improve customer satisfaction, and reduce costs. The information obtained during the business understanding phase serves as the foundation for the subsequent phases of the project, which include data preparation, modeling, evaluation, and deployment.

Data Understanding

The data understanding phase of this project involves collecting data, verifying its accuracy, and conducting exploratory data analysis (EDA) to uncover patterns and insights. In this case, the data is taken from the website of New York city (NYC) and is in parquet format. At this stage the data is in unreadable format and unable to see the contents inside the files.to convert the parquet files to the csv files, python script is used on the apache pyspark scala shell.this script easily convert the data into csv in less time as apache pyspark is a distributed computing system and it is very good at handling the unformatted large data.

Once the data is converted to csv, the next step is to detect and verify anomalies in the dataset. Anomalies in the data are the elements that deviate substantially from the majority of data and may be a result of various factors, such as measurement errors, inaccuracies during data entry, or other factors. Anomalies can be detected using statistical methods such as box

plots, scatter plots, and histograms.in addition to this, one of the biggest anomaly especially for this dataset is timezone because all our systems in california follow pacific standard time(PST). In order to predict exact timing to pick up customers, the operating system time of PST has to be set to east standard time (New York timing). Once anomalies are identified, they can be verified by comparing them to other sources of data or by examining the data collection process.

Data cleaning is the next step in the data understanding phase. Interpolation techniques can be used to fill in missing values in the dataset. Interpolation is a mathematical technique for estimating missing data points based on the values of neighboring data points. Here in this case as it is time series project the interpolation technique has to be done meticulously. The missing values are filled not just by filling the average of all the values. Infact, it is done by filling an approximate average value by its surrounding values.

Finally, EDA is performed to identify patterns and insights in the NYC taxi dataset. EDA involves visualizing the data using charts and graphs to identify trends, relationships, and anomalies in the data. Moreover the data that is used for taxi demand prediction is from the year 2022. We are not considering 2021 and 2020 data because the whole transportation industry got disrupted due to covid pandemic. Any further analysis on this 2020 and 2021 data does not give accurate results for normal scenario. EDA can also be used to identify outliers, clusters, and other patterns that may be relevant to the analysis.

In conclusion, the data understanding phase is critical in preparing the data for analysis. This involves collecting the data, verifying its accuracy, converting it to the appropriate format, detecting and verifying anomalies, cleaning the data using interpolation techniques, and performing EDA to identify patterns and insights. These steps ensure that the data is ready for the subsequent phases of the data science project, including data preparation, modeling, evaluation, and deployment.

Data Preparation

During this project, the stage of data preparation is of great importance as it involves transforming unprocessed data into a structure that is suitable for analysis. In the present case, data preparation includes extracting taxi demand feature variables, translating the data pre-processing for the NYC dataset, doing feature engineering with PCA analysis, and partitioning the data into training and testing subsets.

The first step is to extract the feature variables for taxi demand from the dataset. These variables can include the time of day, day of the week, location, pulocationid, dulocationid, ratecodeid. Feature variables are important because they help to identify relationships in the data and patterns which can be useful to predict taxi demand. Moreover, there is also an embedding of taxi look up table (csv file) to the original NYC dataset because the mapping of pickup location happens with this table. In this lookup table, there are 266 unique locations. After performing EDA in the above phase , Manhattan proves to be the busiest taxi demand of all 266 regions and more deeper analysis has to be performed on this area.

The data is then transformed and pre-processed to ensure that it is in a format suitable for analysis that involves the removal of missing data, converting categorical variables into numerical variables, and scaling the data. In this data transformation phase, the features that are selected in the above step of feature variables are reduced so that only the features that are useful to predict the taxi demand are selected such features include timestamp, pulocationid, dolocationid,airport_fee,payment_type,fare_amount,trip_amount. Feature engineering is the next step in the data preparation phase. The process of producing new features from current data that are more informative and predictive of the target variable is known as feature engineering. PCA analysis can be used in this scenario to discover the most essential characteristics in the dataset. So for this project ,at each time stamp there are multiple pickups

happening in the New York region. To get the count of pickups for each time stamp in each location will be using timestamp and pickup location as crucial parameters . After many trial and errors, the average time period of pickups count for each location is about 15 min. So the count is based on the above parameters. Finally for the 12 month period, pickup count of each location is calculated.smoothing techniques needs to be applied of the calculated counts and as the data of the counts might be of different values or in the precision.

Further, the data is subdivided into training and testing. The training subset is the portion of the data set used to train the ML model, whereas the testing subset is used to evaluate the model's correctness. It is essential to divide the data into training and testing subsets to ensure that the model is well-balanced and can generalize to new data effectively since this is a time series model data cannot be divided into two parts based on the assumptions. A cross validation data needs to be created and needs to splitted according to it.

Therefore, the data preparation phase is critical in preparing the data for analysis. This involves extracting feature variables for taxi demand, transforming the data pre-processing for the NYC dataset, performing feature engineering with PCA analysis, and splitting the data into training and testing subsets. These steps ensure that the data is in a format which is suitable for analysis and that the machine learning model can accurately predict taxi demand.

Modeling

The modeling phase of a taxi demand prediction project involves constructing and training machine learning models to make accurate predictions. Choosing appropriate models based on the data and its target features is a very important task in this phase. Models chosen must handle the data and provide better accuracy. In this project, several ml models such as xg boost, random forest, prophet and dart can be used to build predictive models. These models require parameter tuning to optimize their performance and accuracy.

All the above mentioned algorithms supports time series predictions. The first algorithm is xg boost which performs better when it is combined with arima algorithm. This xg boost takes advantage of this time series algorithms and gives best forecasting results. Whereas the other algorithms are completely time series specific like prophet, dart (lstm, arima). The modeling phase's initial step is to optimize the selected parameters for accuracy. These parameters are critical in ensuring that the models are accurate in predicting taxi demand. The optimization process involves selecting the best values for each parameter that will result in the best model performance.The models are constructed using the optimized parameters, and they are trained using the training subset of the data. The models are trained to learn the patterns and relationships that can predict taxi demand accurately. This involves separating data into training and validation sets as well as adapting the model to the training data.

Once the models are trained, they are tested using the testing subset of the data. The performance of the models is evaluated using various metrics such as precision, recall, F1 score and, accuracy,. The models with the best performance are selected for further optimization.

Using hyper-parameter tuning, models are optimized at the concluding stage of the modeling phase. This procedure involves modifying the models' parameters to enhance their efficacy. Hyper-parameter tuning can be done using various techniques such as grid search and random search.

The modeling phase of this project is critical in building accurate predictive models which involves optimizing the selected parameters, constructing and training ml models, testing the models, and optimizing the models using hyper-parameter tuning. The end result of this phase is a set of predictive models that can accurately predict taxi demand.

Evaluation

In the evaluation phase of a taxi demand prediction project, metrics such as F1 score and RMSE are used to assess the performance of the ML models. F1 score is used to measure the accuracy of the models in making precise predictions, while RMSE is used to measure the error rate of the models. High accuracy metrics are desired, indicating that the models are capable of making accurate predictions.

Performance analysis is conducted using the metrics mentioned above to determine which models are performing well and which ones are not. Models with high F1 scores and low RMSE values are considered to be performing well, while those with low scores and high values are not. The goal is to explore and enhance the models that are not performing well, with the target of improving forecasting accuracy.

Once the models have been explored and enhanced, the best performing model is chosen based on its accuracy metrics. The model with the highest F1 score and the lowest RMSE value is considered to be the best performing model. This model can then be used to make accurate predictions of taxi demand, which can help taxi businesses improve their performance and optimize their resources.

Deployment

The deployment phase of a taxi demand prediction project involves testing the ML models in a staging environment before introducing them in a real-world setting. This is done to ensure that the models are working as expected and are not causing any performance issues. During this phase, performance monitoring is conducted using alerts and feedback to detect any issues that may arise.

Once the models have been tested and deployed, documentation of the models and findings is done to ensure that the information is available for future reference. Moreover, all the data and the implementation is uploaded to github repository with restricted permissions for future enhancements and references. A unique SSH is created for the team members to

access the data and keep the information in a secure manner. This documentation can be used to track the progress of the project and to provide insights into the models' performance.
If any modifications are required, they are implemented during this phase to improve the performance of the models. The modifications may involve tweaking the models' parameters or changing the algorithms used in the models. The goal is to improve the accuracy of the models and make them more efficient.

In conclusion, the deployment phase of a taxi demand prediction project is critical in ensuring that the models are working as expected and are not causing any performance issues. Performance monitoring is conducted using alerts and feedback, and documentation of the models and findings is done to provide insights into the models' performance. Modifications may be required to improve the models' accuracy and efficiency.

2.3 Project Organization Plan

The goal of the taxi demand prediction project is to use data analytics to accurately forecast future demand for taxi services. To achieve this, the project will be organized using the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology, which consists of six stages as shown in below Figure 3.

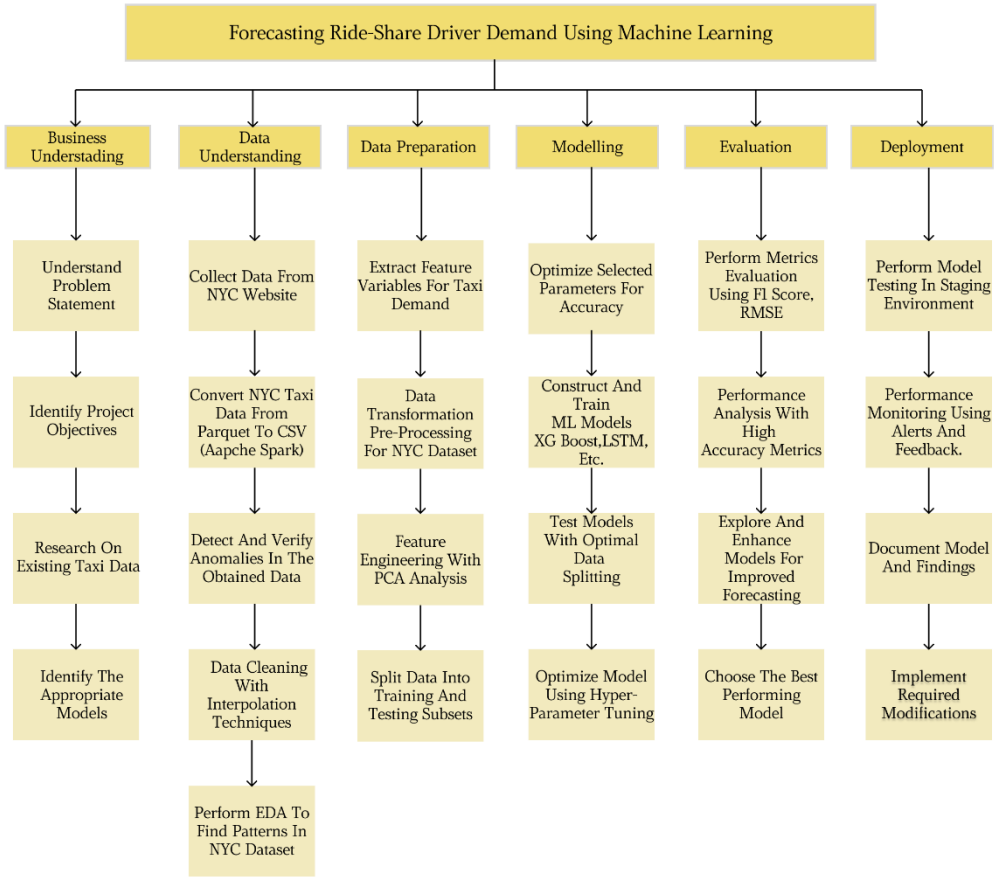
In the business understanding stage, the project team will work with stakeholders to clearly define the problem and identify business objectives. This will involve understanding the current taxi service market, identifying key performance indicators (KPIs), and determining the target audience. In the data understanding stage, the project team will gather and analyze data from various sources, such as historical taxi ride data, weather data, and events data, to identify patterns and trends in taxi demand.

In the data preparation stage, the team will clean, transform, and preprocess the data to prepare it for modeling. This will involve data cleaning, feature engineering, and data normalization. In the modeling stage, the team will use various statistical and machine

learning techniques to build predictive models that accurately forecast taxi demand. The team will also evaluate the performance of the models and choose the best-performing one.

Figure 3

Work Breakdown Structure for Ride-Share Driver Demand Prediction



Note: WBS showing 6 phases of CRISP-DM for project planning

In the evaluation stage, the team will thoroughly test the model to ensure it meets the business objectives and KPIs. This will involve measuring the accuracy of the predictions, comparing the model against benchmarks, and identifying any limitations or issues with the model. Lastly, in the deployment stage, the team will implement the model into the taxi service operations and continuously monitor its performance to ensure it remains accurate and effective.

Overall, by using the CRISP-DM methodology, the taxi demand prediction project will be well-organized and structured, ensuring that the predictive model is accurate, effective, and meets the needs of the business.

2.4 Project Resource Requirements and Plan

The project requires a number of tools and resources, including data storage (Google Cloud Storage), cloud computing (Google Cloud Platform), machine learning libraries (Scikit Learn, Numpy, Pandas, Dask), and version control (GitHub).

To start, the historical taxi ride data will be stored in Google Cloud Storage. This data will then be preprocessed using tools such as Scikit Learn, Numpy, and Pandas to clean and transform the data into a format suitable for training the machine learning model. Google Colab or Jupyter Notebook can be used to perform this preprocessing and exploratory data analysis. Once the data is cleaned and formatted, which will be further used to train a ML model, which will be executed on a 64-bit or higher local machine. Dask can be used to speed up the computation process and make it more efficient.

After the model has been trained and tested, can be used to create data visualizations and analyze the results. The final version of the model can be stored in GitHub for version control and future reference. Overall, the project requires a range of tools and resources to be successful, including cloud storage and computing, data preprocessing and visualization, machine learning libraries, and version control. By utilizing these tools effectively, it is possible to build a predictive model that accurately forecasts taxi demand in a given area at a specific time.

Software Requirements

The Python Programming Language, specifically Version 3.7, is utilized as the foundation for developing the model. Google Colab is employed alongside the Pyarrow library for converting data from Parquet format to a dataframe. Scikit-Learn (Version 0.24)

and Keras (Version 2.12.0) play a vital role in model development. NumPy (Version 1.22.3) is utilized for numerical computations within the model. Pandas (Version 1.4.1) is essential for data collection, pre-processing, and model development tasks. Visualizing data and results is made possible with Seaborn (Version 12.0). The DateTime library (Version 5.1) is used to set the current environment to the New York Timezone. Lastly, Geopandas (Version 12.2) is employed for handling geospatial data within the model. The Table 6 gives an overview of the project software requirements.

Table 6

Software Requirements

Software	Libraries	Configuration	Purpose
Python Programing Language		Version 3.7	Developing a model
Google Colab	Pyarrow	Version 3.0.0	Data Conversion from parquet to dataframe
	Scikit Learn	Version 0.24	Developing a model
	Keras	Version 2.12.0	Developing a model
	NumPy	Version 1.22.3	Developing a model
	Pandas	Version 1.4.1	Data collection, Data Pre-processing and Model development
	Seaborn	Version 12.0	Visualization
	DateTime	Version 5.1	Setting the Current environment to NY Timezone
	Geopandas	Version 12.2	Handling the geospatial data

Hardware Requirements

The local machine, equipped with 12 CPUs, 16 GPUs, and 16 GB of RAM, is dedicated to the development and construction of models. Its robust hardware configuration enables efficient processing and analysis. Additionally, in the Google Cloud Platform (GCP), a primary region in US East with a storage capacity of 10 GB is employed as the main repository for storing the processed CSV files. As a precautionary measure, a backup region in US West with the same storage capacity is utilized to ensure data integrity and availability

in the event of any unforeseen circumstances or disaster recovery scenarios. This dual-region setup guarantees reliable data storage and safeguards against potential data loss. In Table 7 the hardware requirements for this specific project are given.

Table 7

Hardware Requirements

Hardware	Coonfiguration	Purpose
Local Machine	12 CPUs, 16 GPUs, 16 GB RAM	To develop and build the models
GCP – Standard (Region: US East)	10 GB	Primary region to store the processed CSV Files
GCP – Standard (Region: US West)	10 GB	Backup region to store CSV files incase of disaster recovery

Note. Hardware components used in the project

Tools Licenses

JIRA, a freely available tool, serves the purpose of creating a comprehensive project management plan. GitHub, which is also offered for free, acts as the standard repository for storing and monitoring the project's progress. Figma, another free tool, is utilized for creating visual representations such as WBS and PERT charts. Canvas, provided at no cost, is used for developing engaging and impactful presentations. Students leverage MS Office 365 to create and edit documents effectively. Zoom is used as a student tool for group meetings as shown in Table 8

Table 8

Tools and Licenses

Tool	License	Purpose
JIRA	Free	To build a project management plan
Git Hub	Free	Standard repository to store and monitor the project progress
Figma	Free	To draw WBS and PERT Chart
Canvas	Free	To develop presentations

MS Office 365	Student	To develop or edit documents
Zoom	Student	Group Meetings

Note. Tools and Licenses used in the project

Project Cost Estimation and Justification

In this project, a local machine is utilized as the hardware resource for a duration of 3 months, with an estimated cost of $2,000. This serves as a main component for this project as most of the development is being done using this machine. Data storage is managed through NYC TLC data and a GCP bucket, both available at no cost for the same 3-month period. The project benefits from the cloud service management tool provided by GCP Terminal, while Google Colab serves as the software resource for model deployment, including the training, validation, and test datasets. Additional data storage is facilitated through Google Cloud Storage at a cost of $18 for the 3-month duration.

Data preprocessing and model development tasks are accomplished using Google Colab as the software resource, without incurring any extra charges. Jira is employed as the project management tool, Zoom facilitates project meetings, and Git Hub acts as the repository for version control. Lastly, Figma is employed as a designing tool, supporting the project's visualization needs as described in Table 9.

Table 9

Cost Estimation and Justification

Function	Resource Type	Resource	Duration	Cost Estimation
Local Machine	Hardware	64- bit machine or higher	3 Months	$2,000
Data Storage	Hardware	NYC TLC data , GCP bucket	3 Months	Free
Cloud Service management Tool	Software	GCP Terminal	3 Months	Free
Model Deployment : Train, validation, Test Datasets	Software	Google Colab	3 Months	Free
Data Storage	Hardware	Google Cloud Storage	3 Months	18$

Data Preprocessing	Software	Google Colab	3 Months	Free
Model Development	Software	Google Colab	3 Months	Free
Project Management	Tool	Jira	3 Months	Free
Project Meetings	Tool	Zoom	3 Months	Free
Git Repository	Tool	Git Hub	3 Months	Free
Designing	Tool	Figma	3 Months	Free
			Total	$2018

Note. Cost estimations and resources of this project as per the requirements.

2.5 Project Schedule

Gantt Chart

A Gantt chart is a well-known tool which is used for visualization in project management to represent the timeline of a project, the various tasks involved, and their dependencies. This chart usually consists of a horizontal timeline demonstrating the project duration and a vertical list of individual tasks. In this representation, each task is depicted as a horizontal bar that extends over the duration needed to complete it. Arrows connecting the bars indicate task dependencies and illustrate which tasks must be accomplished before others can commence. Gantt charts find widespread applications across various sectors, including construction, software development, event planning, among others. They serve as a valuable instrument for project managers to oversee project progress and ensure timely completion. In summary, a Gantt chart is an effective visual aid that displays a project's timeline, task dependencies, and durations. It is widely used in project management to monitor progress and manage resources efficiently.

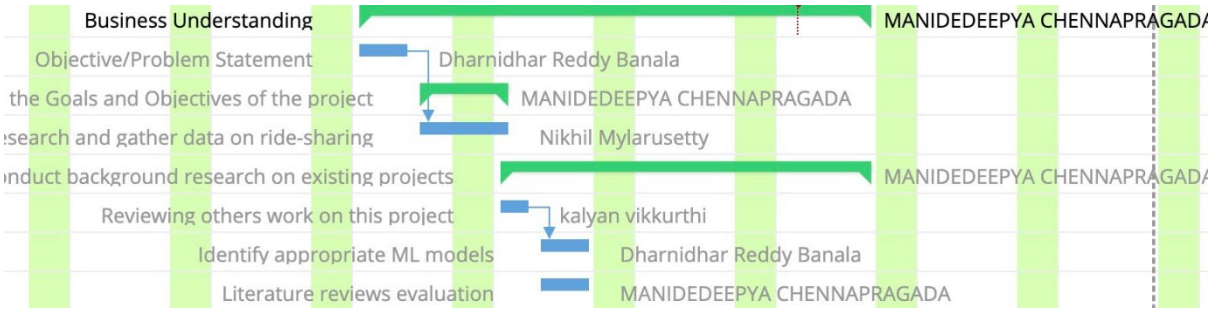
The current research project comprises sprints of two weeks each, from the Business Understanding stage to Deployment. The project team has scheduled weekends as holidays, aligning with standard company working hours. However, the team has opted not to take any spring breaks during the project, and tasks have been allocated throughout the entire project duration, including during the spring break. Each team member has been assigned specific

tasks, ensuring that every member is working on a task at any given point, except in cases where dependencies arise. The subtasks have been assigned for 2-3 days based on the complexity of the task. Overall, the project team has adopted a professional approach to task allocation and management, utilizing effective tools and strategies to ensure project success. Each phase of the project is accompanied by a breakdown of the effort involved, as explained in the following subsections.

Business Understanding. This phase starts on Feb 6, 2023. In this phase, there are two user stories and four tasks that need to be completed by Feb 17, 2023.

Figure 4

Gantt Chart for Business Understanding



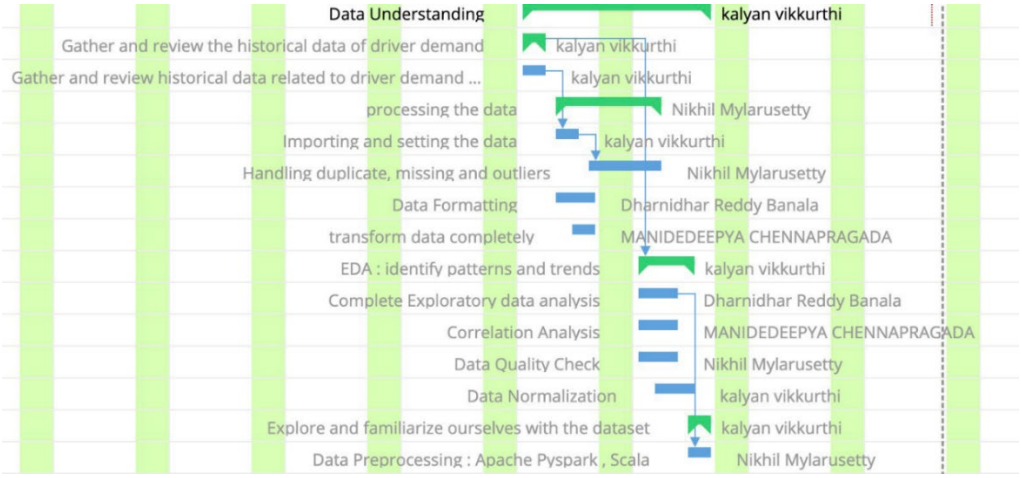
Note. Tasks, timelines, and progress in Business Understanding are depicted in a Gantt chart.

These tasks include. Research and gather data on ride-sharing, reviewing others work on this project, Identify appropriate ML models, Literature reviews evaluation. All these tasks of business understanding are allotted for two to three days and assigned to each member according to their availability. The Figure 4 shows the dependencies of the tasks on each other and assignment of work to each team member.

Data Understanding. This phase starts on Feb 20, 2023. In this phase there are four user stories and ten tasks that need to be completed by March 03, 2023.

Figure 5

Gantt Chart for Data Understanding



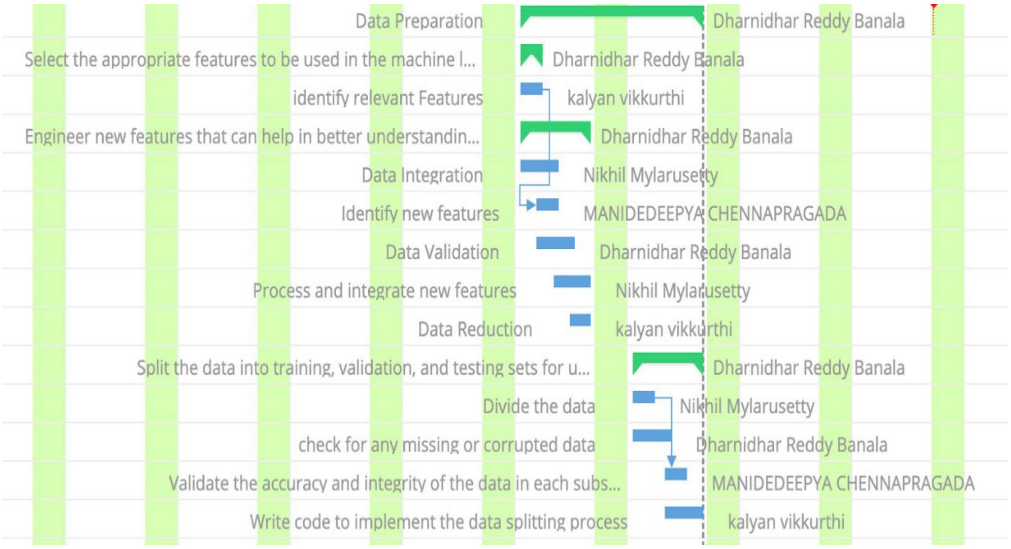
Note. Tasks, timelines, and progress in Data Understanding are depicted in a Gantt chart.

These tasks include Research and gather data on ride-sharing, Gather and review historical data related to driver demand from various datasets, Importing and setting the data, Data Formatting, transform data completely, Handling duplicate, missing and outliers, Complete Exploratory data analysis, Correlation Analysis, Data Quality Check, Data Preprocessing : Apache Pyspark , Scala. Eventhough some of the tasks are assigned to one member in the jira, those tasks are performed in a group as they are group activities that include gathering data, Data Quality checks etc. Moreover, planning activities are also performed during this phase which include WBS, Gantt charts. All these tasks of Data Understanding are allotted for two to three days and assigned to each member according to their availability. The Figure 5 shows the dependencies of the tasks on each other and assignment of work to each team member.

Data Preparation. This phase starts on March 6, 2023. In this phase there are three user stories and ten tasks that need to be completed by March 17, 2023.

Figure 6

Gantt Chart for Data Preparation



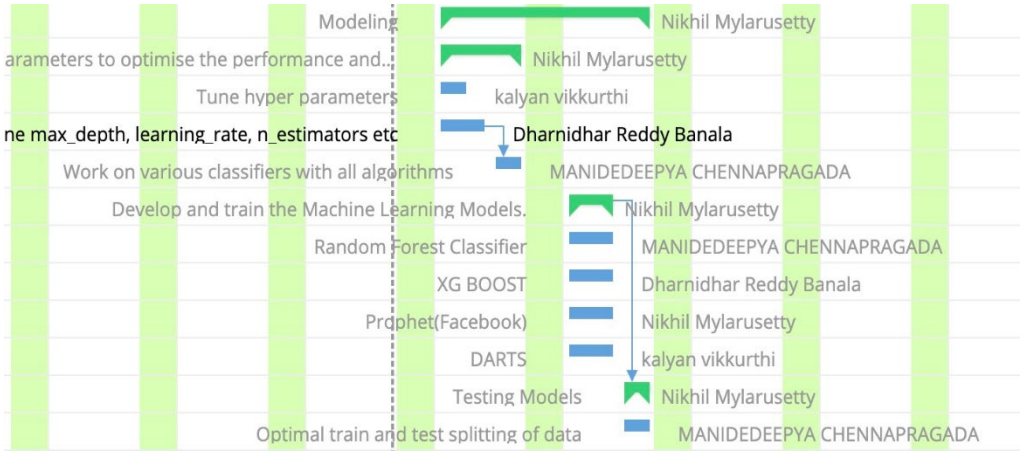
Note. Tasks, timelines, and progress in Data Preparation are depicted in a Gantt chart.

These tasks include identify relevant Features, Data Integration, Identify new features, Data Validation, Process and integrate new features, Data Reduction, Divide the data, check for any missing or corrupted data, Validate the accuracy and integrity of the data in each subset by performing EDA, Write code to implement the data splitting process. Here in this phase Exploratory data analysis is performed and the data that in in the form of parquet files are converted to CSV files to deeply go through the algorithms that is suitable to this data for prediction. Moreover, features are also manipulated in this phase based on the correlational analysis. Each team member is assigned specific Data Preparation tasks based on the project requirements, with a two to three day deadline for completion, as shown in the Figure 6 illustrating dependencies and assignments.

Modeling. This phase starts on March, 2023. In this phase there are three user stories and eight tasks that need to be completed by March 31, 2023.

Figure 7

Gantt Chart for Modeling



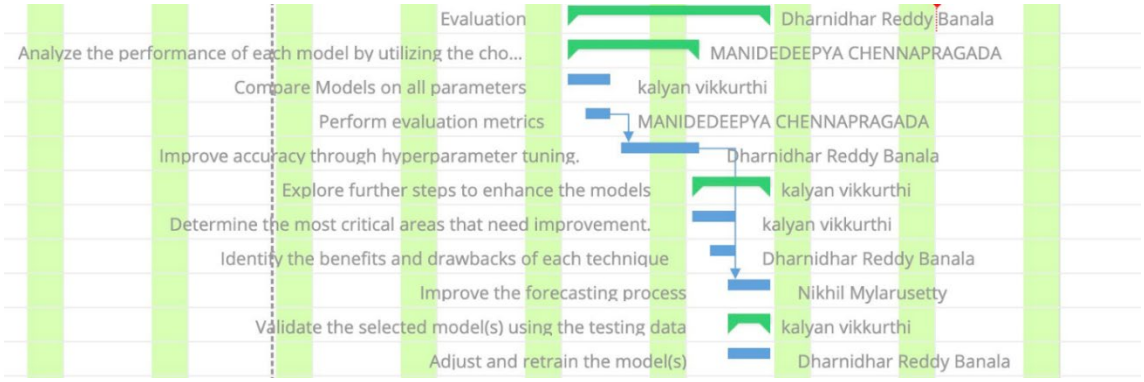
Note. Tasks, timelines, and progress in Modeling are depicted in a Gantt chart.

The first part of the story include fixing and tuning the parameters for the algorithms where each team member is assigned to their particular algorithm and every one does their respective classifiers , parameter determining etc. The final part of this sprint is to test the models based on the new data. The Figure 7 shows the dependencies of the tasks on each other and assignment of work to each team member.

Evaluation. This phase starts on April 03, 2023. In this phase there are three user stories and seven tasks that need to be completed by April 14, 2023.

Figure 8

Gantt Chart for Evaluation



Note. Tasks, timelines, and progress in Evaluation are depicted in a Gantt chart.

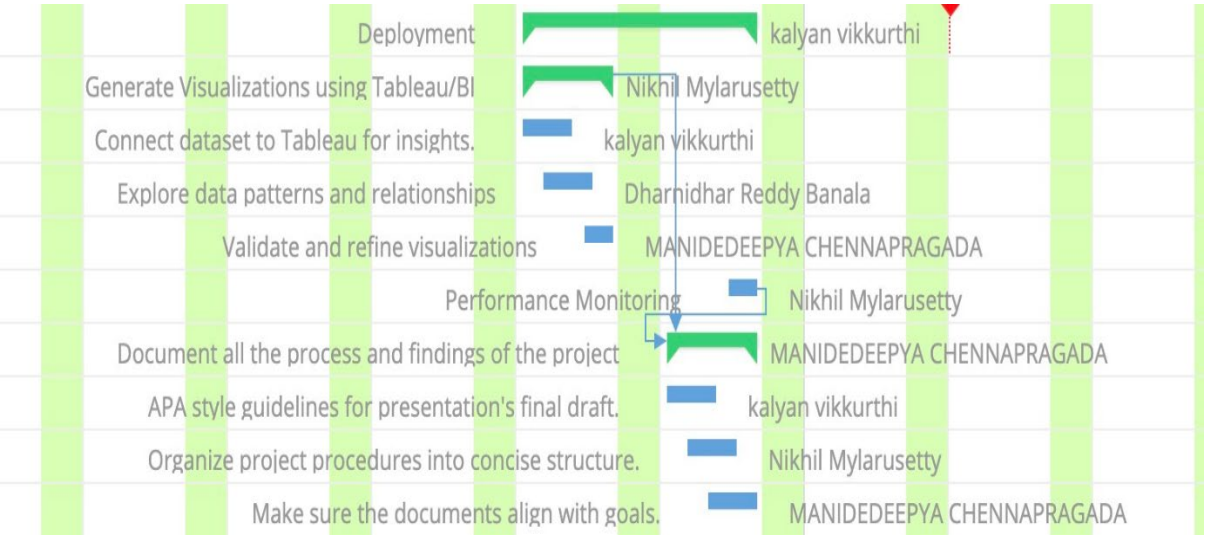
These tasks include Compare Models on all parameters, Perform evaluation metrics, Improve accuracy through hyperparameter tuning, identify the most crucial areas that need improvement, improve the forecasting process, Adjust and retrain the model(s). In this phase

the algorithms performance and accuracy will be improved by using various methods like hyper parameter tuning and ensemble methods. The tasks related to the evaluation process are assigned to team members based on their availability and are given a timeline of two to three days. The Figure 8 displays how the tasks are interdependent and the workload is allocated among team members.

Deployment. This is the last phase of the sprint where it starts on April 17 2023 and ends on May 5 2023. This phase includes connecting the project data with the tableau to provide clear insights and also it involves the documentation process for the project . So if there is any need to extend the documentation process it can be extended to complete the document. Moreover, even in this phase whole team collaboratively works on document process. The below Figure 9 depicts the process of the deployment phase.

Figure 9

Gantt Chart for Deployment



Note. Tasks, timelines, and progress in Deployment are depicted in a Gantt chart.

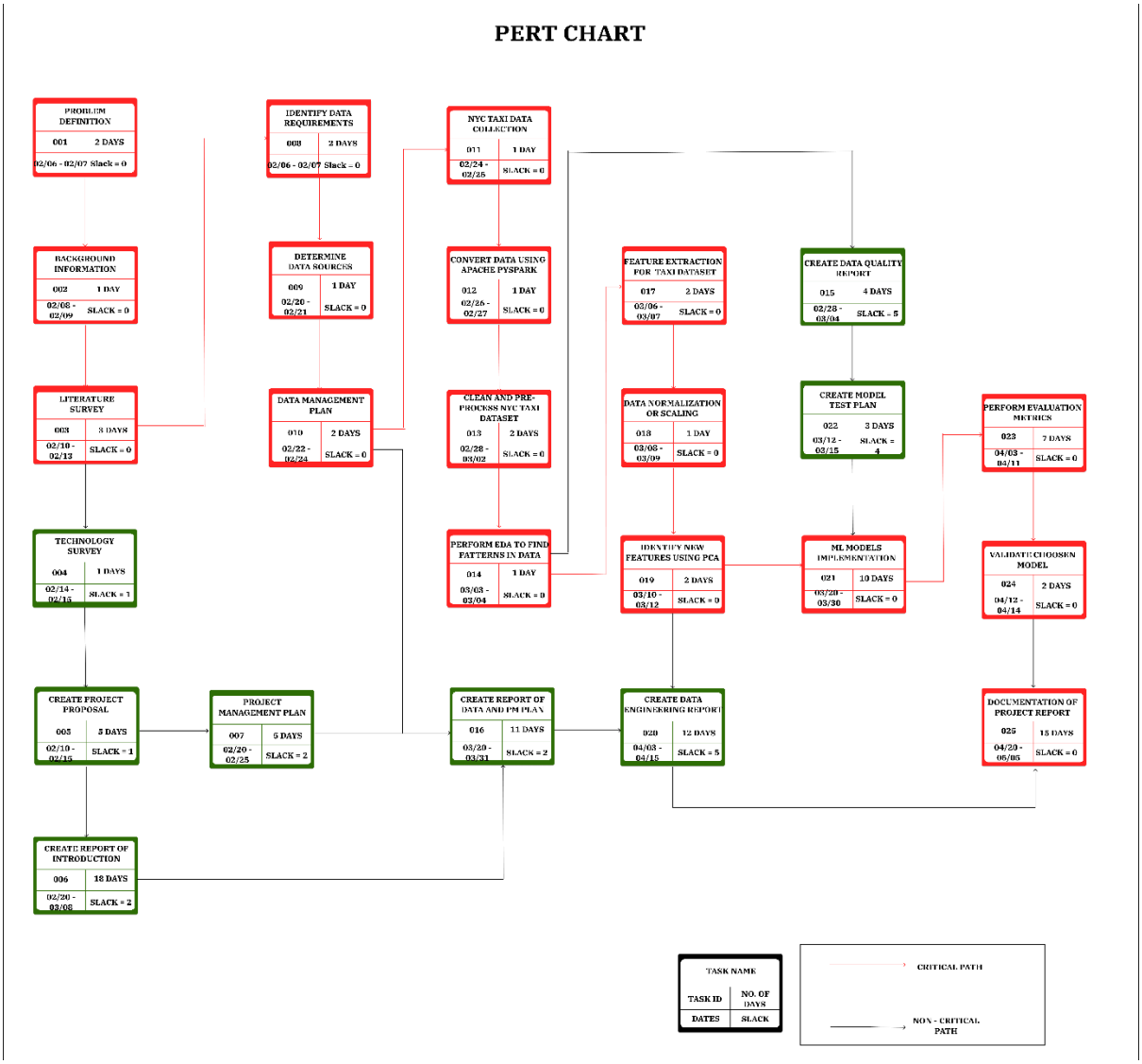
PERT (Program Evaluation Review Technique) Chart

A PERT chart is a tool used for planning and scheduling project responsibilities and activities of a project. PERT charts, as opposed to Gantt charts, which are used to visualize a project's timeline, focus on task dependencies and the critical path, which establishes the

minimum time required to complete a project. PERT charts use nodes to represent tasks and arrows to represent dependencies, with each node having information about the duration and dependencies of the task as shown in Figure 10

Figure 10

PERT Chart for Ride-Share Driver Demand Prediction



Note: Illustration of Tasks, dependencies, slack, critical and non-critical paths in PERT chart.

The given PERT chart outlines the activities involved in a project aimed at building a machine learning model for predicting taxi demand in New York City. The project timeline spans over a period of three months, from February 6th to May 5th. Each task is represented by a node in the chart and is linked to other tasks with directed arrows, indicating the flow

and dependency of the tasks. The project's flow is from left to right, with each task's completion required for the subsequent task to begin. The chart helps visualize the interdependencies of tasks and the critical path refers to the series of tasks that play a decisive role in determining the project's finishing date.

The project begins with a problem definition activity, which is expected to take two days and has no slack time. This is followed by a literature survey activity, which spans over three days with no slack time. A technology survey activity is then conducted over one day, followed by the creation of a project proposal and introduction report, which is expected to take five days.

The project management plan activity is then carried out over a period of 18 days, with two units of slack time. This activity involves the identification of data requirements, determination of data sources, and the creation of a data management plan. The NYC taxi data collection activity is then performed over one day, followed by data conversion and cleaning over a period of three days with no slack time. Exploratory data analysis is carried out over two days with no slack time, and a data quality report is created over four days with five units of slack time.

The next set of activities involves the creation of a report of data and project management plan over a period of 11 days with two units of slack time. This is followed by feature extraction for the taxi dataset over a period of 11 days with no slack time. The data normalization or scaling activity is then carried out over one day, followed by the identification of new features using PCA over a period of two days with no slack time. The data engineering report activity is then performed over a period of 12 days with five units of slack time.

The project then enters the machine learning model implementation phase, starting with the creation of a model test plan over 12 days with five units of slack time. The ML

models implementation activity is then carried out over three days with four units of slack time. The evaluation metrics activity spans over ten days with no slack time, followed by the validation of the chosen model over seven days with no slack time. Finally, the project is concluded with a documentation of the project report over 15 days with no slack time.

Data Engineering

3.1 Data Process

The initial stage of the data process is done by collecting transportation-related data for New York City, covering the period from January to June 2022. In this project, the data is collected from the official NYC TLC website, which contains various aspects such as taxi rides, traffic patterns, and relevant metrics. In the data collection phase, the procedure entails gaining entry to the data, extracting the pertinent details, and subsequently archiving these files in the Parquet file format.

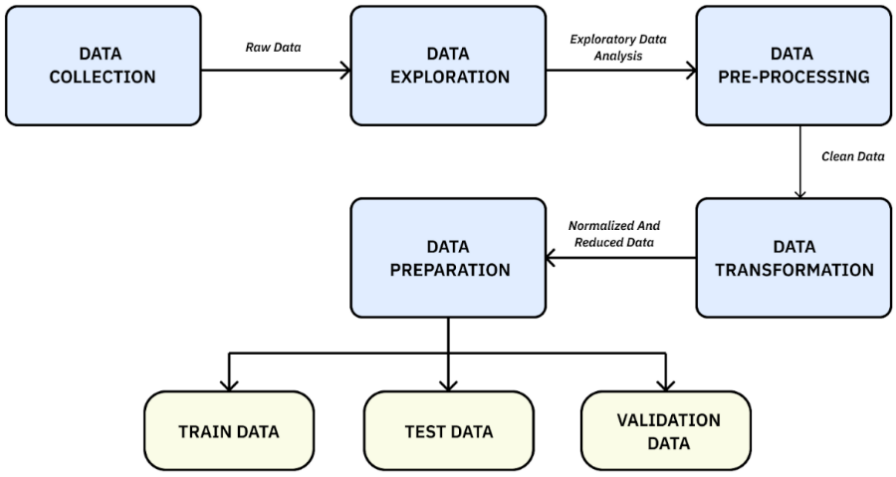
After the data collection phase, the next stage in the data processing procedure entails the analysis of the gathered data through the utilization of Python libraries by the team. Since the data is in parquet format it will be converted to a data frame with the help of inbuilt Pyarrow library in Python. This phase aims to comprehensively understand the data's structure, quality, and potential insights. Techniques like descriptive statistics, data visualization, and correlation analysis will be applied to uncover patterns, trends, and any anomalies within the data. Through this exploration, any data issues or missing values will be identified and addressed before proceeding to the next stages. The aim of data pre-processing will be to clean the data to find meaningful insights by specifically focusing on pickups, fares, tips, and regions.

After the completion of data pre-processing phase, the processed data will undergo a transformation phase that encompasses various operations, including feature extraction, smoothing, normalization, standardization, principal component analysis (PCA) analysis, and regularization. These transformations will be applied to enhance the data and make it more suitable for subsequent analysis and modeling. This transformed data is stored in the Google Cloud Platform (GCP).

Once the data has undergone a transformation, the final step will be data preparation, where the data will be split into three sets in a ratio of 64:18:18, respectively as test, train and validation. This split is crucial to create separate datasets that will be used for model training and evaluation. 18% of the data will be designated as the testing set, which will be used to evaluate the trained model's performance. The remaining data will be used for model training to ensure the model's ability to generalize well to data that is completely new and prevent overfitting. This data preparation procedure will guarantee that the model was trained and evaluated on distinct datasets, allowing for a thorough evaluation of its performance as shown in Figure 11

Figure 11

Flowchart For Data Process



3.2 Data Collection

The NYCTLC website keeps data collected from the early 2000's to the present in a well-organized way. Initially, the data was gathered on paper by hand, and subsequently, taxi drivers were required to enter the data in Excel sheets, which were then loaded into the system. The growing demand for taxis has led to a direct correlation between data storage and the amount of data being stored in the CSV format. . The TLC eventually altered the format

to parquet files, which allow for the compression of large amounts of data as provided in Table 10

Table 10

Data Collection Plan For NYC TLC Dataset

Description of the data collection	
Where are we collecting the data from?	The New York City Taxi and Limousine Commission (TLC) collects and maintains trip data for yellow taxis operating in New York City. The data is collected automatically through the taxi's electronic meter system, which records the pickup and dropoff times and locations, passenger count, trip distance, fare amount, and payment type.
What will be done with the data once it has been collected?	The collected data will be used to predict driver demand and optimize operations for ride-sharing companies and transportation providers. With the obtained data we will be generating a target feature for daily pickup count with respective to each locationID and each time stamp. This will be our output feature (Y) for our models. Four models, including Random Forest Regressor, XGBoost Regressor, Facebook Prophet, and Darts, will be trained on historical data to forecast future driver demand based on factors such as time, day of the week, weather, events, and other variables. This will enable the efficient distribution of the taxi fleet to meet demand in different areas of the city.

Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)										
	1	2	3	4	5	6	7	8	9	10
Variable title	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	PULocationID	DOLocationID	RatecodeID	fare_amount	payment_type
Input (X) or output (Y) variable?	X	X	X	X	X	X	X	X	X	X
Unit of measurement	N/A	Date and time	Date and time	Count	Miles	N/A	N/A	N/A	USD	N/A
Data type	Integer	DateTime	DateTime	Integer	Float	Integer	Integer	Integer	Float	Integer
Collection method	Automated									
If manual	N/A									
Historical data exist?	Yes									
Source of historical data	NYC Taxi and Limousine Commission									
Sampling frequency	Daily									
Sub-grouping needed?	Yes									
Sub-group size	Hourly									
Data collector	Kalyan Vikkurthi									
Start date	01-01-2022									
Due date	31-06-2022									

Key Variables - A summary of the chosen input variables (Y's) and/or output variables (X's)									
	11	12	13	14	15	16	17	18	19
Variable title	congestion_surcharge	store_and_fwd_flag	airport_fee	total_amount	extra	mta_tax	tip_amount	improvement_surcharge	tolls_amount
Input (X) or output (Y) variable?	X	X	X	X	X	X	X	X	X
Unit of measurement	Currency (USD)	Numeric	Categorical (Yes/No)	Currency (USD)	Currency (USD)	Currency (USD)	Currency (USD)	Currency (USD)	Currency (USD)
Data type	Float/Decimal	Character/String	Float/Decimal	Float/Decimal	Float/Decimal	Float/Decimal	Float/Decimal	Float/Decimal	Float/Decimal
Collection method	Automated								
If manual	N/A								
Historical data exist?	Yes								
Source of historical data	NYC Taxi and Limousine Commission								
Sampling frequency	Daily								
Sub-grouping needed?	Yes								
Sub-group size	Hourly								
Data collector	Kalyan Vikkurthi								
Start date	01-01-2022								
Due date	31-06-2022								

The TLC maintains the official website of New York City with the assistance of technology suppliers in the New York region, with the consent of Taxi Cab & Livery Passenger Enhancement Programs (TPEP/LPEP). The data which is recorded by the TLC is classified into three categories. However, one of the business models is selected for the

present research which is yellow taxi datasets. Because of the market's disorganized nature, the yellow taxi data is the best fit for our project. Yellow cabs are only permitted to pick up passengers via street hailing, which requires a significant amount of effort and time on the part of both the consumer and the driver. The website's data is saved in parquet files, which are downloaded and converted to Comma Separated Files (CSV) format for analysis and prediction. These parquet file format offers various advantages, including economical storage and faster querying, making it an excellent choice for our project. The data collected and evaluated from the TLC website is real and credible, having been gathered over several decades through a methodical and structured approach.

The dataset utilized for this research takes up roughly 9-12 GB of space when considering data for 6 months, particularly for the first 6 months of 2022 which is after COVID-19 era. However, the data in the form of parquet files is not available for the entire year, but rather for each month. To undertake exploratory data analysis and prediction, the data from all months must be combined into a single dataset in the next process.

There are two other datasets which have been collected from the same data source for analytical purpose which does not effect the original trips raw dataset for each month. The shape files in this dataset contain geometric and attribute data representing various geographic features, allowing for spatial analysis and visualization of transportation-related information. They provide a passive representation of geographic boundaries, streets, and regions, facilitating a deeper understanding of the distribution and relationship between different transportation metrics across specific areas.

The lookup files included in this raw data collection has reference guides, providing associations between numeric or coded values and their corresponding meanings or descriptions. These files offer additional information that aids in the interpretation of specific data fields, such as unique identifiers for regions or zones.

Dataset Sample for the Raw Data Resource

The Data that is being collected from the New York website is completely raw and has columns that are not required for the current analysis and prediction. However, having a clear understanding of the data with all the columns helps to analyze the data deeply and get useful insights which will be helpful in predicting the output with better accuracy. The below dataset of raw sample contains 19 columns.

- The first column includes 'VendorID' which consists of the data of the vendors Creative Mobile Technologies and VerifoneInc where they are represented in the dataset as the number 1 and 2 respectively.
- Coming to the second column 'tpep_pickup_datetime' which has the data of date and time when the meter in the cab is engaged or when the cab or taxi starts its trip. The data in this column is in the form of date time format.
- Similarly, the next column 'tpep_dropoff_datetime' gives the data when the trip of the cab has been completed or the meter in the cab is disengaged. Even this data is also in the date time format
- 'Passenger_count' is a column that defines the number of passengers in that trip which is mentioned by the driver. The data type of this column is float.
- 'trip_distance' includes the total distance covered by the trip from the pickup to the drop off of a single customer where the datatype of this column is float.
- 'RateCodeID' is the column which determines the rate that is to be calculated for a trip based on its nature. There are different categories for this column.
- 'Store_and_fwd_flag' is the column in which the taxi meter stores the trip record and sends the data that is stored to the Vendors.
- 'Payment_type' is the type of payment that is accepted by the drivers.
- 'Fare_amount' is the amount that is deducted for the trip

- ‘Extra’ column is the amount that is overcharged for waiting, midnight expenses etc.
- ‘MTA_tax’ is the column which describes the amount that is collected on the meter.
- ‘ Improvement_surcharge’ is the charge that is collected based on the long distances that a customer has travelled. Generally, it is $0.30 after a certain long distance.
- ‘Tip_amount’ is the amount of tip that is paid only through the electronic transaction. The tips paid through cash are not included in the dataset.
- ‘Tolls_amount’ column includes the amount that is paid at the toll plaza.
- ‘PULocationID’ and ‘DOLocationID’ columns include all the locations where the pickup and dropoff has happened.
- ‘Total_amount’ is the column that describes the total amount paid by the customer that includes mta tax, tip, toll amount.
- ‘Congestion_Surcharge’ is the charge that is paid to the government as tax.
- ‘Airport_fee’ is the extra fee charged if the pickup is at the airport.

Table 11

Data Exploration Plan

EDA Task	Members responsible
Analysis of the dataset to remove duplicates during training, testing, and validation phases.	Nikhil and Dharnidhar
Conduct column-wise analysis to investigate the data	Nikhil and Dharnidhar
Find any inconsistent data from dataset and discard them.	Kalyan and Manidedeepya
Data Regularization	Kalyan and Manidedeepya
Data Normalization	Nikhil and Dharnidhar
Data Reduction	Kalyan and Manidedeepya

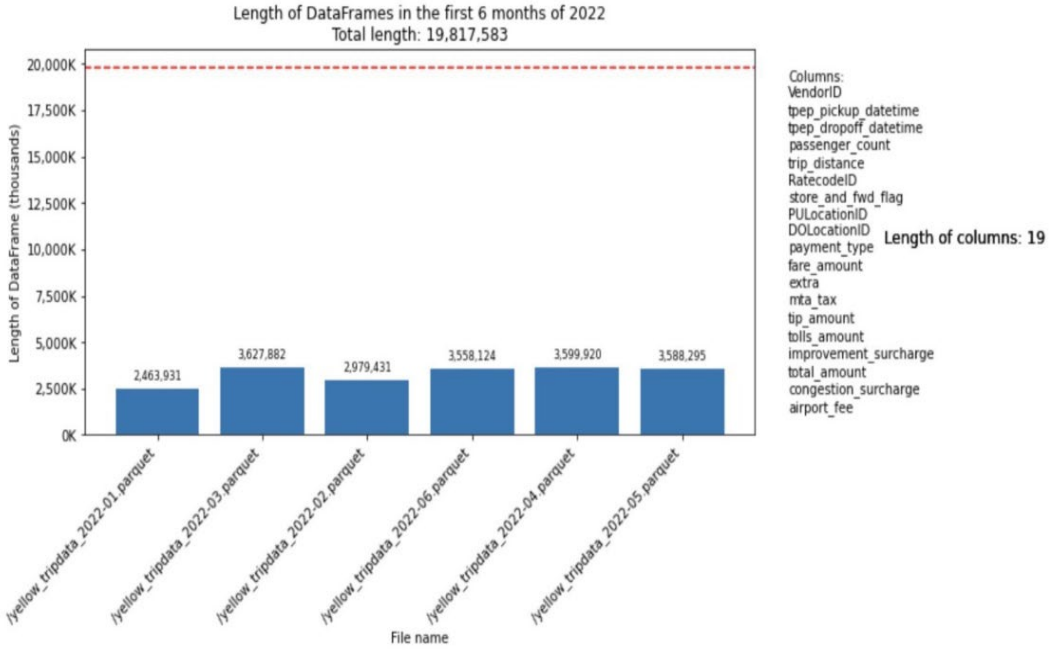
Note: Task division for each member in EDA phase

The above Table 11 gives information about the members responsible for each EDA task which is allocated to each of the members.

The Figure 12 gives information about the raw data set of each month from the official NYC TLC data that is collected with corresponding columns collected.

Figure 12

Raw Dataset Details For Each Month




```
Total length of entire DataFrames: 19817583
```

Figure 13

Sample Raw Dataset For The Month January

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID
0	1	2022-01-01 00:35:40	2022-01-01 00:53:29	2.0	3.80	1.0	N	142	236
1	1	2022-01-01 00:33:43	2022-01-01 00:42:07	1.0	2.10	1.0	N	236	42
2	2	2022-01-01 00:53:21	2022-01-01 01:02:19	1.0	0.97	1.0	N	166	166
3	2	2022-01-01 00:25:21	2022-01-01 00:35:23	1.0	1.09	1.0	N	114	68
4	2	2022-01-01 00:36:48	2022-01-01 01:14:20	1.0	4.30	1.0	N	68	163
...	...	...	...	...	...	...	...	...	...
2463926	2	2022-01-31 23:36:53	2022-01-31 23:42:51	NaN	1.32	NaN	None	90	170
2463927	2	2022-01-31 23:44:22	2022-01-31 23:55:01	NaN	4.19	NaN	None	107	75
2463928	2	2022-01-31 23:39:00	2022-01-31 23:50:00	NaN	2.10	NaN	None	113	246
2463929	2	2022-01-31 23:36:42	2022-01-31 23:48:45	NaN	2.92	NaN	None	148	164
2463930	2	2022-01-31 23:46:00	2022-02-01 00:13:00	NaN	8.94	NaN	None	186	181

2463931 rows × 19 columns

Figure 13 and Figure 14 give details about the January month's raw dataset and the number of rows and columns in them.

Figure 14

Summary Of Raw Dataset For The Month January

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2463931 entries, 0 to 2463930
Data columns (total 19 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   VendorID               2463931 non-null  int64
 1   tpep_pickup_datetime   2463931 non-null  datetime64[ns]
 2   tpep_dropoff_datetime  2463931 non-null  datetime64[ns]
 3   passenger_count        2392428 non-null  float64
 4   trip_distance          2463931 non-null  float64
 5   RatecodeID             2392428 non-null  float64
 6   store_and_fwd_flag     2392428 non-null  object
 7   PULocationID           2463931 non-null  int64
 8   DOLocationID           2463931 non-null  int64
 9   payment_type           2463931 non-null  int64
 10  fare_amount            2463931 non-null  float64
 11  extra                  2463931 non-null  float64
 12  mta_tax                2463931 non-null  float64
 13  tip_amount             2463931 non-null  float64
 14  tolls_amount           2463931 non-null  float64
 15  improvement_surcharge  2463931 non-null  float64
 16  total_amount           2463931 non-null  float64
 17  congestion_surcharge   2392428 non-null  float64
 18  airport_fee            2392428 non-null  float64
dtypes: datetime64[ns](2), float64(12), int64(4), object(1)
memory usage: 357.2+ MB
```

Figure 15 and Figure 16 represents the sample dataset after adding 2 columns to find if there are any data entry issues. These columns are derived by splitting the pickup date time column using pandas. These columns will be month and year. This process is done for every month's dataset.

Figure 15

Sample January Dataset Showing The Two Added Columns

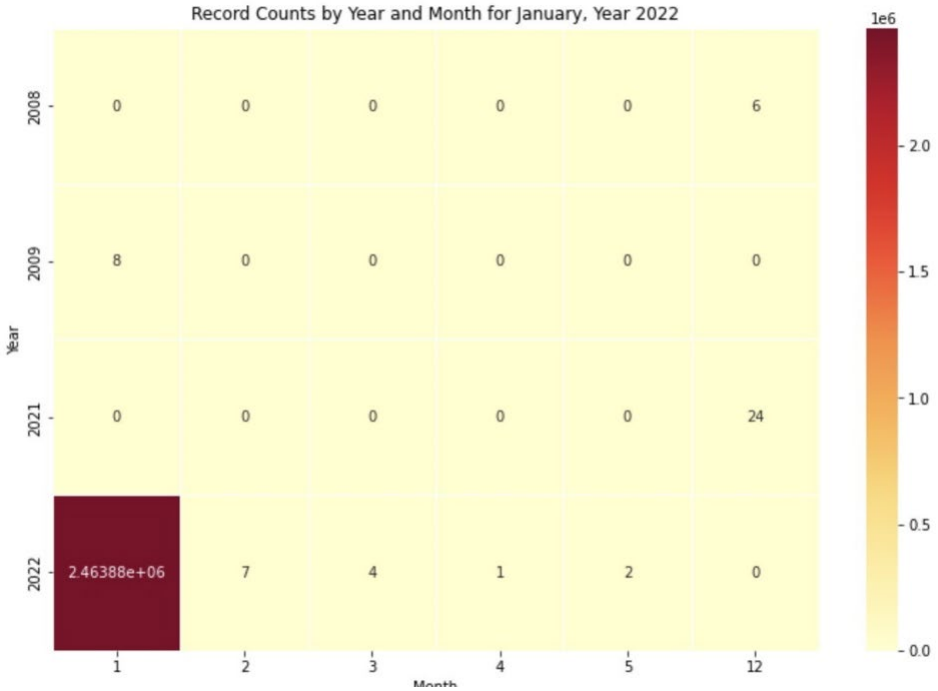
	month	year	airport_fee	congestion_surcharge	total_amount	improvement_surcharge	tolls_amount	tip_amount	mta_tax	extra	...	payment_type	D
2463930	1	2022	NaN	NaN	35.06	0.3	0.0	6.28	0.5	0.0	...	0	
2463929	1	2022	NaN	NaN	15.70	0.3	0.0	0.00	0.5	0.0	...	0	
2463928	1	2022	NaN	NaN	16.52	0.3	0.0	2.00	0.5	0.0	...	0	
2463927	1	2022	NaN	NaN	24.45	0.3	0.0	4.35	0.5	0.0	...	0	
2463926	1	2022	NaN	NaN	13.69	0.3	0.0	2.39	0.5	0.0	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4	1	2022	0.0	2.5	30.30	0.3	0.0	3.00	0.5	0.5	...	1	
3	1	2022	0.0	2.5	11.80	0.3	0.0	0.00	0.5	0.5	...	2	
2	1	2022	0.0	0.0	10.56	0.3	0.0	1.76	0.5	0.5	...	1	
1	1	2022	0.0	0.0	13.30	0.3	0.0	4.00	0.5	0.5	...	1	
0	1	2022	0.0	2.5	21.95	0.3	0.0	3.65	0.5	3.0	...	1	

2463931 rows × 21 columns

Note. The columns month and year added to detect the initial outliers for the January raw dataset

Figure 16

Outliers For January Dataset



Note. Data entry issues visualized using heatmap for the January month

Figure 17

After Removing Outliers Of The January Dataset

	month	year	airport_fee	congestion_surcharge	total_amount	improvement_surcharge	tolls_amount	tip_amount	mta_tax	extra	...	payment_type	D(
0	1	2022	0.0	2.5	21.95	0.3	0.0	3.65	0.5	3.0	...	1	
1	1	2022	0.0	0.0	13.30	0.3	0.0	4.00	0.5	0.5	...	1	
2	1	2022	0.0	0.0	10.56	0.3	0.0	1.76	0.5	0.5	...	1	
3	1	2022	0.0	2.5	11.80	0.3	0.0	0.00	0.5	0.5	...	2	
4	1	2022	0.0	2.5	30.30	0.3	0.0	3.00	0.5	0.5	...	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2463926	1	2022	NaN	NaN	13.69	0.3	0.0	2.39	0.5	0.0	...	0	
2463927	1	2022	NaN	NaN	24.45	0.3	0.0	4.35	0.5	0.0	...	0	
2463928	1	2022	NaN	NaN	16.52	0.3	0.0	2.00	0.5	0.0	...	0	
2463929	1	2022	NaN	NaN	15.70	0.3	0.0	0.00	0.5	0.0	...	0	
2463930	1	2022	NaN	NaN	35.06	0.3	0.0	6.28	0.5	0.0	...	0	

2463879 rows × 21 columns

Now the January data frame don't have quality issues as seen from the Figure 17 to merge into our final dataset. This process is repeated for the next five months of the raw dataset that was collected from the NYC TLC website and merged into a final data frame for further analysis. This process requires to add the month and year columns to identify the

initial raw data outliers for each month. The merged dataset of the 6 months is shown in Figure 18.

Figure 18

Sample Merged Dataset For 6 Months

	day	year	airport_fee	congestion_surcharge	total_amount	improvement_surcharge	tolls_amount	tip_amount	mta_tax	extra	...	payment_type	DOLo
0	1	2022	0.0	2.5	21.95	0.3	0.0	3.65	0.5	3.0	...	1	
1	1	2022	0.0	0.0	13.30	0.3	0.0	4.00	0.5	0.5	...	1	
2	1	2022	0.0	0.0	10.56	0.3	0.0	1.76	0.5	0.5	...	1	
3	1	2022	0.0	2.5	11.80	0.3	0.0	0.00	0.5	0.5	...	2	
4	1	2022	0.0	2.5	30.30	0.3	0.0	3.00	0.5	0.5	...	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
3558119	30	2022	NaN	NaN	15.00	0.3	0.0	0.00	0.5	0.5	...	0	
3558120	30	2022	NaN	NaN	27.35	0.3	0.0	5.19	0.5	0.0	...	0	
3558121	30	2022	NaN	NaN	16.43	0.3	0.0	3.10	0.5	0.0	...	0	
3558122	30	2022	NaN	NaN	27.64	0.3	0.0	0.00	0.5	0.0	...	0	
3558123	30	2022	NaN	NaN	24.46	0.3	0.0	0.00	0.5	0.0	...	0	

19816565 rows × 21 columns

Note. The dataset contains data after removing the data entry issues for first 6 months of 2022

In addition to the above taxi data there is other data that has been considered in order to clearly visualize the patterns on a map. On the NYC TLC website, the shape file folder contains various files that include the main shape file, database files, projection file, spatial index file and metadata file. Out of all these files, only the main shape file is considered as it has the zone, LocationID, borough and geometry. The raw dataset sample can be seen in Figure 19.

Figure 19

Sample Raw Data Of Shape File

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091.011 19...
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008, 103343...
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.770 256767.698, 1026495.593 ...
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068.667 20...
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.310 144283.336, 936046.565 14...
...	...	...	...	...	...	...	...
258	259	0.126750	0.000395	Woodlawn/Wakefield	259	Bronx	POLYGON ((1025414.782 270986.139, 1025138.624 ...
259	260	0.133514	0.000422	Woodside	260	Queens	POLYGON ((1011466.966 216463.005, 1011545.889 ...
260	261	0.027120	0.000034	World Trade Center	261	Manhattan	POLYGON ((980555.204 196138.486, 980570.792 19...
261	262	0.049064	0.000122	Yorkville East	262	Manhattan	MULTIPOLYGON (((999804.795 224498.527, 999824....
262	263	0.037017	0.000066	Yorkville West	263	Manhattan	POLYGON ((997493.323 220912.386, 997355.264 22...

263 rows × 7 columns

Another dataset that is downloaded from NYC TLC website is lookup table. This is the dataset that provides the service_zone in addition to columns like borough and LocationID. This service_zone column is useful to segregate the regions based on different types of business areas such as airports, train terminals etc. The raw dataset is shown in Figure 20.

Figure 20

Sample Raw Data For Lookup Table

	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
...	...	...	...	...
260	261	Manhattan	World Trade Center	Yellow Zone
261	262	Manhattan	Yorkville East	Yellow Zone
262	263	Manhattan	Yorkville West	Yellow Zone
263	264	Unknown	NV	NaN
264	265	Unknown	NaN	NaN

265 rows × 4 columns

Analysis on merged dataset and other raw dataset

EDA, which stands for exploratory data analysis, is a technique which helps to analyze the data with deeper insights. This process helps to explore the dataset in every possible way. With the present data from the NYCTLC website, this EDA process involves in investigating the relationship among various variables and also in finding the problems with the quality of the data.

The dataset is carefully examined to identify any occurrences of null values, which refer to data points that are either absent or undefined. Similarly, duplicate values, which refer to repeated entries of the same data, are also checked to ensure data integrity and avoid

redundancy. The purpose of this is to uphold the quality and precision of data for subsequent analysis and processing. This can been seen in Figure 21, Figure 22 and Figure 23.

Figure 21

Duplicate Values

```
Number of duplicate rows: 0
Duplicate rows:
Empty DataFrame
Columns: [VendorID, tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count, trip_distance, RatecodeID, store_and_fwd_fla
g, PULocationID, DOLocationID, payment_type, fare_amount, extra, mta_tax, tip_amount, tolls_amount, improvement_surcharge, tota
l_amount, congestion_surcharge, airport_fee, year, month]
Index: []

[0 rows x 21 columns]
```

Figure 22

Null Values For Merged Data

```
VendorID                     0
tpep_pickup_datetime         0
tpep_dropoff_datetime        0
passenger_count         671901
trip_distance                0
RatecodeID              671901
store_and_fwd_flag      671901
PULocationID                 0
DOLocationID                 0
payment_type                 0
fare_amount                  0
extra                        0
mta_tax                      0
tip_amount                   0
tolls_amount                 0
improvement_surcharge        0
total_amount                 0
congestion_surcharge    671901
airport_fee             671901
year                         0
day                          0
pickup_date                  0
dtype: int64
```

Figure 23

Null values of lookup table

```
LocationID      0
Borough         0
Zone            1
service_zone    2
dtype: int64
```

3.3 Data Pre-processing

The pickup time and drop-off time in the dataset are not aligned with the local time zone of the NYC region. To ensure consistency and accurate analysis, a conversion of the

timestamps to the NYC region's time zone needs to be performed. This can be accomplished by applying a time zone conversion function or adjusting the timestamps based on the time zone offset of the NYC region. By undertaking this conversion, the data will be standardized to the appropriate time zone, allowing for proper analysis and interpretation of the pickup and drop-off times within the context of the NYC region.

Null values are eliminated from the dataset through column deletion rather than row deletion to avoid any adverse impact on the forecasting project. Deleting rows would have a significant effect on the project as it would potentially disrupt the sequence and continuity of the data. By opting for column deletion, only the specific columns containing null values are removed, ensuring that the overall structure and integrity of the dataset remain intact, which is crucial for accurate forecasting. It is shown in Figure 24

Figure 24

Sample Dataset After Null Values Removed

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	trip_distance	PULocationID	DOLocationID	payment_type	fare_amount	extra	mta_tax	tip_am
0	1	2022-01-01 00:35:40	2022-01-01 00:53:29	3.80	142	236	1	14.50	3.0	0.5	
1	1	2022-01-01 00:33:43	2022-01-01 00:42:07	2.10	236	42	1	8.00	0.5	0.5	
2	2	2022-01-01 00:53:21	2022-01-01 01:02:19	0.97	166	166	1	7.50	0.5	0.5	
3	2	2022-01-01 00:25:21	2022-01-01 00:35:23	1.09	114	68	2	8.00	0.5	0.5	
4	2	2022-01-01 00:36:48	2022-01-01 01:14:20	4.30	68	163	1	23.50	0.5	0.5	
...	...	...	...	...	...	...	...	...	...	...	
3558119	1	2022-06-30 23:45:51	2022-06-30 23:51:48	0.00	148	256	0	9.20	0.5	0.5	
3558120	2	2022-06-30 23:25:00	2022-06-30 23:40:00	5.01	79	262	0	18.86	0.0	0.5	
3558121	2	2022-06-30 23:29:00	2022-06-30 23:37:00	1.55	164	79	0	10.03	0.0	0.5	
3558122	2	2022-06-30 23:24:15	2022-06-30 23:50:19	5.30	211	239	0	24.34	0.0	0.5	
3558123	2	2022-06-30 23:33:53	2022-06-30 23:54:58	4.41	255	158	0	21.16	0.0	0.5	

19816565 rows × 16 columns

Note. Merged data after null value columns removed

Upon on previous analysis, it was discovered that the dataset should only contain 263 locations. However, two additional locations were found within the dataset that are unnamed which has null values indicating anomalies.

To ensure data accuracy and consistency, these unnamed locations were identified as anomalies and subsequently removed from the dataset as shown in Figure 25. This removal

process aims to eliminate any potential inconsistencies or misinterpretations that may arise from including these unnamed locations in the analysis.

Figure 25

Lookup Dataset After Null-Values Are Removed

	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
...	...	...	...	...
258	259	Bronx	Woodlawn/Wakefield	Boro Zone
259	260	Queens	Woodside	Boro Zone
260	261	Manhattan	World Trade Center	Yellow Zone
261	262	Manhattan	Yorkville East	Yellow Zone
262	263	Manhattan	Yorkville West	Yellow Zone

263 rows × 4 columns

To extract longitude and latitude information from the shape file, the geometry column is divided, allowing for the separate retrieval of these geographic coordinates. This process enables the acquisition of longitude and latitude data, which can be utilized for various spatial analyses and mapping purposes. By dividing the geometry column and extracting the longitude and latitude values, the dataset gains valuable geographical context, enhancing the understanding and utilization of the shape file data. The sample data is shown in Figure 26.

Figure 26

Processed Shape File

	LocationID	latitude	longitude
0	1	191376.749531	9.359968e+05
1	2	164018.754403	1.031086e+06
2	3	254265.478659	1.026453e+06
3	4	202959.782391	9.906340e+05
4	5	140681.351376	9.318714e+05
...	...	...	...
258	259	266453.414552	1.025106e+06
259	260	210434.891894	1.010212e+06
260	261	197635.173702	9.806395e+05
261	262	221974.400788	9.990647e+05
262	263	223005.988409	9.978179e+05

263 rows × 3 columns

3.4 Data Transformation

After performing data cleaning and data exploration, the cleaned data has to be converted to be suitable for modeling. For this project, the main aim is to find the number of pickups at particular time with respect to each locationID. To process this, the data set has to be modified to get the desired results. From the previous data preprocessing step, the dataset has 19816565 rows and 16 columns. Out of all these only the necessary columns are pickup_date,PULocationID and the third column is the target feature that has to be generated. This target feature is called daily_pickups and is calculated by grouping all the locationIDs along with pickup date and making a sum of all pickups on a single day with respect to locationID. With this transformation the dataset has been transformed to the desired data frame as shown in Figure 27

Figure 27

Transformed Dataset

	pickup_date	PULocationID	daily_pickups
0	2022-01-01	1	43
1	2022-01-01	3	2
2	2022-01-01	4	122
3	2022-01-01	5	1
4	2022-01-01	7	83
...	...	...	...
39302	2022-06-30	261	543
39303	2022-06-30	262	1608
39304	2022-06-30	263	2284
39305	2022-06-30	264	1257
39306	2022-06-30	265	360

39307 rows × 3 columns

Feature Extraction

As this project involves time series data, the Feature extraction for this data is done based on the date and time column. The format of the datetime column is DD:MM:YYYY HH:MM:SS where pickup_year, pickup_dayofweek, day, and month are extracted from that single column which resulted in the increase in the number of columns for the dataset which can be seen in Figure 28.

Figure 28

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month
0	2022-01-01	1	43	2022	5	1	1
1	2022-01-01	3	2	2022	5	1	1
2	2022-01-01	4	122	2022	5	1	1
3	2022-01-01	5	1	2022	5	1	1
4	2022-01-01	7	83	2022	5	1	1
...	...	...	...	...	...	...	...
39302	2022-06-30	261	543	2022	3	30	6
39303	2022-06-30	262	1608	2022	3	30	6
39304	2022-06-30	263	2284	2022	3	30	6
39305	2022-06-30	264	1257	2022	3	30	6
39306	2022-06-30	265	360	2022	3	30	6

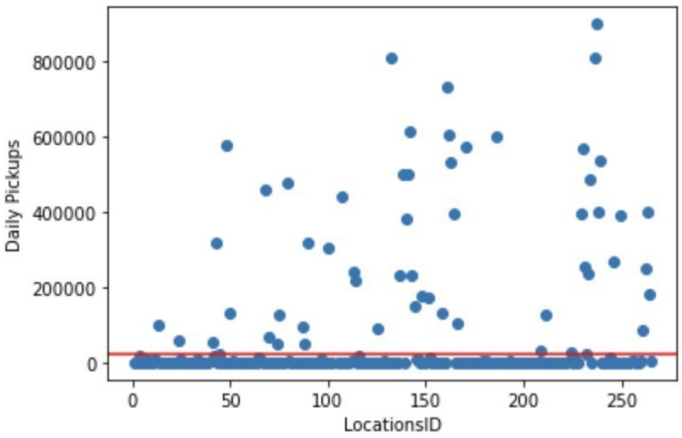
39307 rows × 7 columns

After the data got transformed as shown in Figure 28 based on daily pickups, there are anomalies in the number of pickups per day where the values are very abnormal when compared to normal mean values with respective to daily pickup counts. Here, there is a chance to replace all the anomaly data by interpolating techniques but for this time series data replacing values cannot give accurate prediction when the data is fed to machine learning

models. So, the only way to get rid of these anomalies is to remove them by keeping a threshold value of 10000 pickups such that all the number of pickups above 10000 will be removed as seen from the Figure 29.

Figure 29

Daily Pickups Based On Location ID



Note. Visualization on outliers for daily pickups with respective to locationID

The Figure 30 represent the sample dataset after removal of outliers from the daily pickup column

Figure 30

Sample Dataset After Outliers Removal

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month
0	2022-01-01	1	43	2022	5	1	1
1	2022-01-01	3	2	2022	5	1	1
3	2022-01-01	5	1	2022	5	1	1
5	2022-01-01	8	1	2022	5	1	1
6	2022-01-01	10	14	2022	5	1	1
...	...	...	...	...	...	...	...
39296	2022-06-30	254	3	2022	3	30	6
39298	2022-06-30	256	30	2022	3	30	6
39299	2022-06-30	257	1	2022	3	30	6
39300	2022-06-30	258	5	2022	3	30	6
39301	2022-06-30	260	23	2022	3	30	6

26456 rows × 7 columns

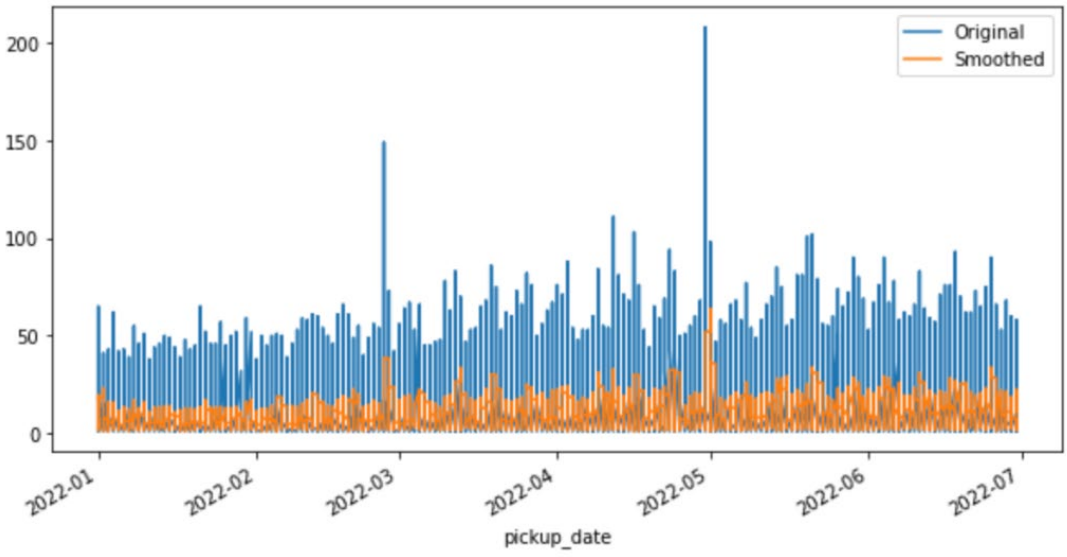
Smoothing

In this Project, a smoothing technique is used to remove the noise and irregularities in the data. Smoothing removes the noise from the original data and makes the pattern finding and analysis much easier. Smoothing will also help to identify key trends by reducing the impact of random fluctuation. This process is very important in time series data before advanced analysis is being applied. There are several ways in smoothing, like moving average, exponential smoothing, and polynomial regression.

Out of which, this project uses moving average where the rolling mean is determined by taking a 7-day moving window and applying it to the 'daily pickups' column using the pandas rolling and mean methods. In Moving Average technique, the level of smoothing will be controlled by varying the size of the moving window of nearby data points that are evaluated for averaging. This value can be adjusted to get the appropriate amount of smoothing. Smoothing will also help the model to train more efficiently and give improved accuracies. The below graph from Figure 31 depicts how data is smoothed after feature extraction and the sample data can be seen in Figure 32.

Figure 31

Graph on Original and Smoothed data



Note. There is no impact on rows and columns but has impact on values inside the column

Figure 32

Sample dataset after Smoothing

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month
0	2022-01-01	1	43	2022	5	1	1
1	2022-01-01	3	2	2022	5	1	1
3	2022-01-01	5	1	2022	5	1	1
5	2022-01-01	8	1	2022	5	1	1
6	2022-01-01	10	14	2022	5	1	1
...	...	...	...	...	...	...	...
39296	2022-06-30	254	3	2022	3	30	6
39298	2022-06-30	256	30	2022	3	30	6
39299	2022-06-30	257	1	2022	3	30	6
39300	2022-06-30	258	5	2022	3	30	6
39301	2022-06-30	260	23	2022	3	30	6

26456 rows × 7 columns

Data Normalization

Data Normalization is the process of normalizing numerical values and rescaling them to a standard distribution. All of these values in the common Distribution enhance the model's precision and ability to predict. There are a number of methods for normalizing data, including Log transformation, Z-score , Decimal Scaling , and min-max Normalization. For this dataset, Min-max normalization is used which is imported from sklearn library and is applied to the feature daily_pickup in this Dataset.

This feature was chosen because it is essential for predicting the outcome, and a single outlier could have a significant impact on the accuracy and the predicted output. This Normalization assisted in the detection of outliers by transforming extreme values into normalized values. Difference between the original feature values and normalize feature values are depicted in the Figure 33 and sample data clearly explains as seen from the Figure 34.

Figure 33

Normalized Features

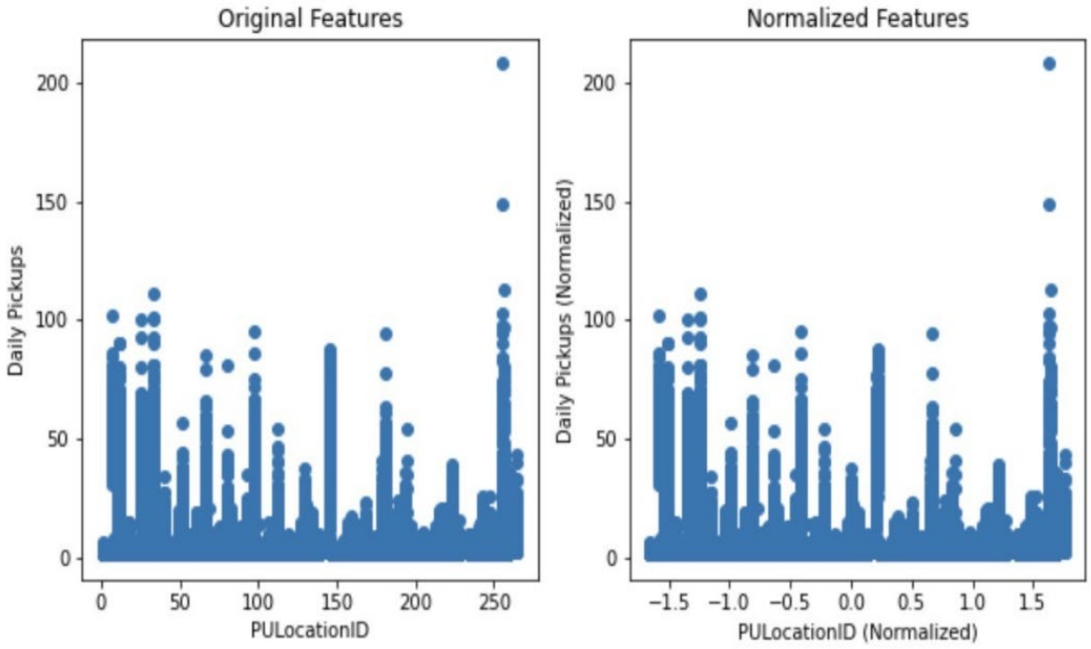


Figure 34

Sample Dataset After Normalization

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month	normalized_pickups
0	2022-01-01	1	43	2022	5	1	1	0.177966
1	2022-01-01	3	2	2022	5	1	1	0.004237
3	2022-01-01	5	1	2022	5	1	1	0.000000
5	2022-01-01	8	1	2022	5	1	1	0.000000
6	2022-01-01	10	14	2022	5	1	1	0.055085
...	...	...	...	...	...	...	...	...
39296	2022-06-30	254	3	2022	3	30	6	0.008475
39298	2022-06-30	256	30	2022	3	30	6	0.122881
39299	2022-06-30	257	1	2022	3	30	6	0.000000
39300	2022-06-30	258	5	2022	3	30	6	0.016949
39301	2022-06-30	260	23	2022	3	30	6	0.093220

26456 rows × 8 columns

Principal Component Analysis (PCA)

PCA is a method for converting the large datasets into minimized datasets by reducing its dimensions, is applied to this dataset by decomposing the original obtained

features into a more manageable collection of uncorrelated variables called principal components.

Initially, a subset of features will be selected for use in the PCA analysis, and then the data will be normalized using the Standard Scaler method. PCA is then applied to the standardized dataset, with the number of principal components set to 2 for the original dataset as seen from Figure 35. In order to extract as much information as possible from the original data, the PCA algorithm generates two linear combinations of the initial features as new components. The ultimate goal is to minimize the dimensionality of the provided dataset.

This new dataset is used for further research, simulation, and modeling. Each principal component's variance ratio, a measure of how much information each principal component contains, is described. A high explained variance ratio indicates that the principal component accounts for a substantial quantity of data variation. The purpose of this is to reduce the dimensionality of the dataset and to identify the most significant features that contribute to the target variable.

Figure 35

Sample Of PCA Components

	principal component 1	principal component 2
0	2.087731	-2.041722
1	2.091158	-2.021197
2	2.094584	-2.000672
3	2.099725	-1.969884
4	2.103151	-1.949358
...	...	...
26451	-1.397609	2.247829
26452	-1.394182	2.268354
26453	-1.392469	2.278617
26454	-1.390756	2.288880
26455	-1.387329	2.309405

26456 rows × 2 columns

Figure 36

PCA for daily_pickups

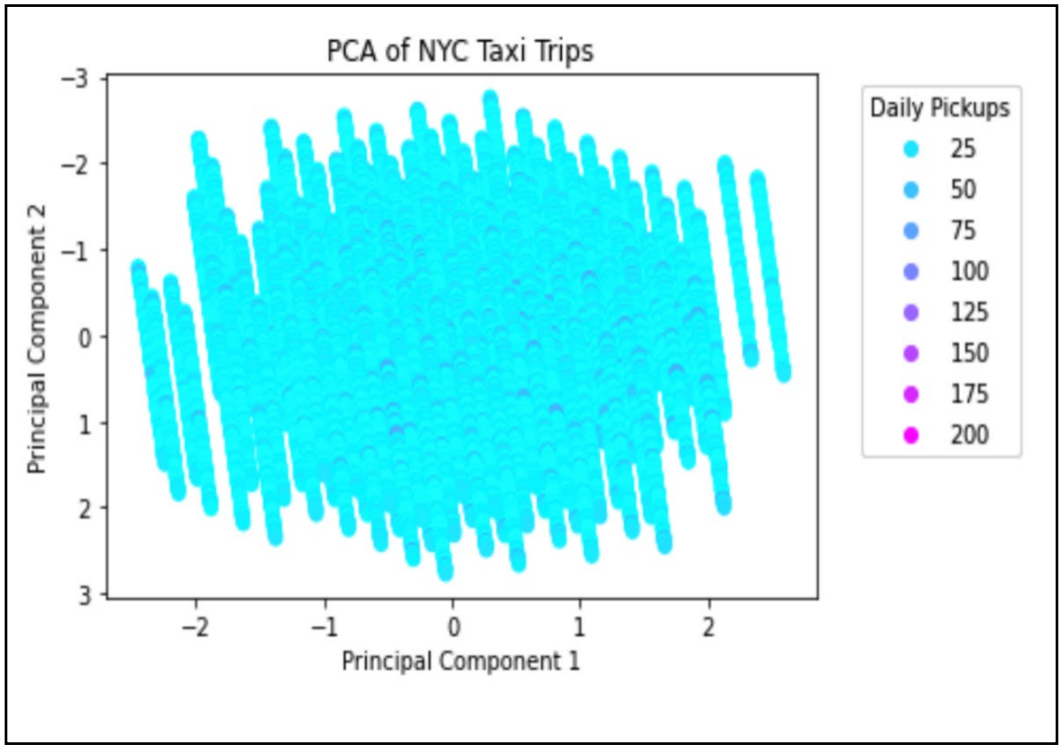


Figure 37

Sample Dataset After PCA

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month	normalized_pickups
0	2022-01-01	1	43	2022	5	1	1	0.177966
1	2022-01-01	3	2	2022	5	1	1	0.004237
3	2022-01-01	5	1	2022	5	1	1	0.000000
5	2022-01-01	8	1	2022	5	1	1	0.000000
6	2022-01-01	10	14	2022	5	1	1	0.055085
...	...	...	...	...	...	...	...	...
39296	2022-06-30	254	3	2022	3	30	6	0.008475
39298	2022-06-30	256	30	2022	3	30	6	0.122881
39299	2022-06-30	257	1	2022	3	30	6	0.000000
39300	2022-06-30	258	5	2022	3	30	6	0.016949
39301	2022-06-30	260	23	2022	3	30	6	0.093220

26456 rows × 8 columns

The Figure 37 for the principal component analysis, where the x-axis is the pc 2 and the y-axis is pc 1. It illustrates the relationship between the two fundamental components and the target_variable. In addition, this procedure helps identify clusters and potential outliers.

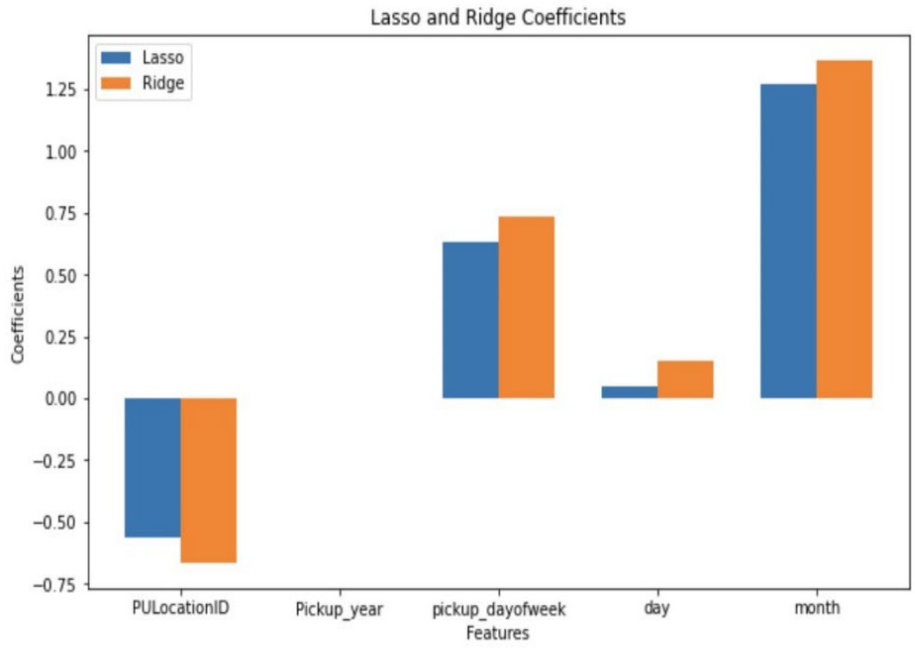
Data Regularization

Due to the complexity and size of our dataset, there exists a possibility of encountering the overfitting phenomenon during the training phase. This can ultimately lead to suboptimal performance of the model. Regularization can be utilized to mitigate overfitting by incorporating a penalty term into the loss function. The regularization process yielded coefficients for each feature, indicating their respective effects on the target variable. The inclusion of a penalty term in the model serves to mitigate the risk of overemphasizing any particular feature within the dataset, thereby bolstering the model's capacity for generalization.

Regularization enables the current model to discern the influence of individual features on the target variable and assign appropriate weights to them. This approach can facilitate the identification of the most crucial features for predicting the target variable and the ones that can be safely disregarded.

Figure 38

Bar Graph for L1 and L2




```
Lasso coefficients: [-0.56591245  0.          0.63051176  0.05124303  1.2700267 ]
Ridge coefficients: [-0.6660894   0.          0.73300903  0.1521206   1.36915643]
```

The graph presented below illustrates the coefficients of Lasso and Ridge for the dataset's features. Additionally, it signifies the comparison of the two aforementioned coefficients. Moreover, based on the aforementioned graph, it can be inferred that certain features exert a greater influence on the targeted outcome, namely the accurate prediction of the number of pickup locations at a given time.

Through the construction of this graph, we can deduce the variables that are influencing our result. The selected data regularization technique for this project is L2 Regularization, as it aims to enhance the precision of the model, which is a vital aspect of time series analysis and prediction.

The utilization of L1 Regularization is likely to yield a subset of the most significant features, which may not be particularly relevant to the current dataset due to the absence of inter-feature correlation.

Figure 39

Sample dataset after L1 and L2 regularization

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month	normalized_pickups
0	2022-01-01	1	43	2022	5	1	1	0.177966
1	2022-01-01	3	2	2022	5	1	1	0.004237
3	2022-01-01	5	1	2022	5	1	1	0.000000
5	2022-01-01	8	1	2022	5	1	1	0.000000
6	2022-01-01	10	14	2022	5	1	1	0.055085
...	...	...	...	...	...	...	...	...
39296	2022-06-30	254	3	2022	3	30	6	0.008475
39298	2022-06-30	256	30	2022	3	30	6	0.122881
39299	2022-06-30	257	1	2022	3	30	6	0.000000
39300	2022-06-30	258	5	2022	3	30	6	0.016949
39301	2022-06-30	260	23	2022	3	30	6	0.093220

26456 rows × 8 columns

Though Smoothing, Normalization, PCA, and regularization are used in the data transformation phase, this project is benefitted by only smoothing and normalization. The reason is that, for time series data there are outliers, fluctuations, noise and variability which

are efficiently handled by smoothing. According to Zhu et al. (2022) smoothing can effectively remove the short term fluctuations and outliers in the data and helps to improve the accuracy during the modeling phase. PCA is not working for this project as there is not much of linear relationship between the data where PCA generally assumes data to be linear as mentioned in the paper by Vitanov et al. (2014) this being the reason there is possibility of information loss if PCA is used.And regularization is also not efficient in case of this project as that can handle overfitting but do not directly address the noise and variability. Normalization is something which this project is benefitted by making the data better with the essence of originality not being lost.

3.5 Data Preparation

In this whole process, the most crucial part is to split the dataset into test, train, and validation sets for both machine learning and deep learning algorithms to predict that accuracy. This splitting of data for the NYC TLC yellow taxi Dataset is done differently as it is time series data.

Here, the training, testing, and validation data are divided in the ratio 64:18:18 based on months. For this project, it is not possible to directly divide the data in the form of a ratio. Instead, all three phases of data are divided in a sequential time manner. The first four months, January, February, March, and April months are considered as training data, and May and June are considered for testing and validation, respectively. To train the models in an effective manner, 64 percent of data is considered for the training phase, and the rest is divided into equal lengths.

Considering the testing phase, the data is considered for one month, and tested the accuracy of the model. The models' accuracy can be improved by tuning the hyperparameters based on testing data. Finally, the saved model is tested on Validation data which in this case is the last month of the data.

The samples of train, test and validation data are in the Figures 40, 41 and 42. Here, in the Figure 40, there 64% of data which is the training data with 16228 of data points or rows. Figure 41 shows the validation data which is 18% with 4269 rows. And Figure 42 shows the testing data which is of 18% ratio with 4291 rows.

Figure 40

Sample Training Dataset

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month	normalized_pickups
0	2022-01-01	1	2	2022	5	1	1	0.004831
1	2022-01-01	5	1	2022	5	1	1	0.000000
2	2022-01-01	7	65	2022	5	1	1	0.309179
3	2022-01-01	8	1	2022	5	1	1	0.000000
4	2022-01-01	10	8	2022	5	1	1	0.033816
...	...	...	...	...	...	...	...	...
16223	2022-04-30	254	1	2022	5	30	4	0.000000
16224	2022-04-30	255	208	2022	5	30	4	1.000000
16225	2022-04-30	256	113	2022	5	30	4	0.541063
16226	2022-04-30	260	25	2022	5	30	4	0.115942
16227	2022-04-30	265	12	2022	5	30	4	0.053140

16228 rows × 8 columns

Figure 41

Sample Validation Dataset

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month	normalized_pickups
16228	2022-05-01	1	5	2022	6	1	5	0.019324
16229	2022-05-01	3	3	2022	6	1	5	0.009662
16230	2022-05-01	7	81	2022	6	1	5	0.386473
16231	2022-05-01	10	35	2022	6	1	5	0.164251
16232	2022-05-01	12	46	2022	6	1	5	0.217391
...	...	...	...	...	...	...	...	...
20492	2022-05-31	256	17	2022	1	31	5	0.077295
20493	2022-05-31	257	5	2022	1	31	5	0.019324
20494	2022-05-31	259	1	2022	1	31	5	0.000000
20495	2022-05-31	260	20	2022	1	31	5	0.091787
20496	2022-05-31	265	13	2022	1	31	5	0.057971

4269 rows × 8 columns

Figure 42

Sample Testing Dataset

	pickup_date	PULocationID	daily_pickups	Pickup_year	pickup_dayofweek	day	month	normalized_pickups
20497	2022-06-01	1	2	2022	2	1	6	0.004831
20498	2022-06-01	7	45	2022	2	1	6	0.212560
20499	2022-06-01	10	29	2022	2	1	6	0.135266
20500	2022-06-01	11	1	2022	2	1	6	0.000000
20501	2022-06-01	12	49	2022	2	1	6	0.231884
...	...	...	...	...	...	...	...	...
24783	2022-06-30	255	23	2022	3	30	6	0.106280
24784	2022-06-30	256	17	2022	3	30	6	0.077295
24785	2022-06-30	258	4	2022	3	30	6	0.014493
24786	2022-06-30	260	13	2022	3	30	6	0.057971
24787	2022-06-30	265	18	2022	3	30	6	0.082126

4291 rows × 8 columns

3.6 Data Statistics

Table 12 show the modification of datasets through various processes followed during different phases of the project. In this data processing pipeline, the initial stage involves handling data entry issues in the raw datasets for each month from January to June, along with a lookup file dataset and a shape file dataset. After removing the data entry issues, the datasets are reduced in size.

Table 12

Summary Of Dataset Rows And Columns After Different Processes

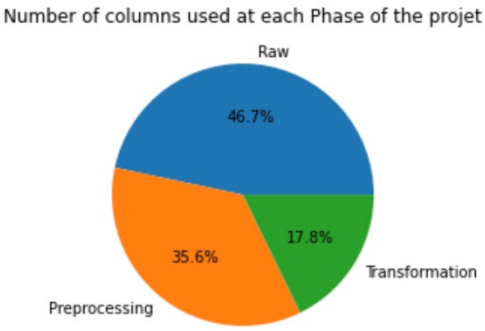
Stage	Phase	Process	After Process [Rows X Col]
Raw	Before Removing Data entry issues	January Dataset	2463931 x 19
		February Dataset	2979431 x 19
		March Dataset	3627882 x 19
		April Dataset	3599920 x 19
		May Dataset	3588295 x 19
		June Dataset	3558124 x 19
		Lookup file dataset	265 x 4
		Shape file dataset	263 x 7
	After Removing Data entry issues	January Dataset	2463879 x 19
		February Dataset	2979367 x 19
		March Dataset	3627793 x 19
		April Dataset	3599762 x 19
		May Dataset	3588152 x 19
		June Dataset	3557612 x 19
Pre-processing	Merged Data	6 months data	19816565 x 19
	Dataset After Null Value columns are Removed	6 months data	19816565 x 16

	Dataset After Null Value rows are Removed	Lookup file dataset	263 x 4
	After Removing Unnecessary columns	Shape file dataset	263 x 3
Transformation	Transformed dataset	6 months data	39307 x 3
	Feature extraction	6 months data	39307 x 7
	Feature dataset after outliers removed	6 months data	26456 x 7
	Data Smoothing	6 months data	26456 x 7
	Data Normalization	6 months data	26456 x 8
	PCA	6 months data	26456 x 8
	Data Regularization	6 months data	26456 x 8
Preparation	Training	6 months data	16228 x 8
	Validation	6 months data	4269 x 8
	Testing	6 months data	4291 x 8

The data from all six months is then merged, resulting in a dataset with 19816565 rows and 19 columns. Null value columns are removed, reducing the dataset to 19816565 rows and 16 columns. Null value rows are also removed from the lookup file dataset, leaving it with 263 rows and 4 columns. Unnecessary columns are removed from the shape file dataset, resulting in dimensions of 263 rows by 3 columns.

Figure 43

Number of Columns Used in Each Phase



The transformation stage involves further transforming the dataset into a different representation, resulting in a transformed dataset of dimensions 39307 rows by 3 columns. Feature extraction is performed on the transformed dataset, resulting in a feature dataset of dimensions 39307 rows by 7 columns. Outliers are removed from the feature dataset, resulting in a dataset with 26456 rows and 7 columns. Data smoothing is applied to this

dataset, followed by data normalization, resulting in a dataset with 26456 rows and 8 columns.

Next, PCA (Principal Component Analysis) is performed on the dataset, reducing the dimensionality to 8 components while retaining the same number of rows. However, for our time series data PCA does not have any impact on the 8 feature components and proceeding with the previous normalized data having 2456 rows and 8 columns. Finally, data regularization is applied to the dataset, resulting in a dataset with dimensions of 26456 rows by 8 columns.

In the preparation stage, the transformed and processed dataset is divided into three sets: training, testing, and validation. The training set hass dimensions of 16228 rows by 8 columns, the testing set has dimensions of 4291 rows by 8 columns, and the validation set also has dimensions of 4291 rows by 8 columns. The specific dimensions of each set are provided to give an understanding of the number of instances (rows) and the number of features (columns) in each set. These dimensions will vary based on the problem's specific requirements., the available data, and the chosen ratio for splitting the dataset.

Figure 44

Distribution of Train, Test and Validation Data

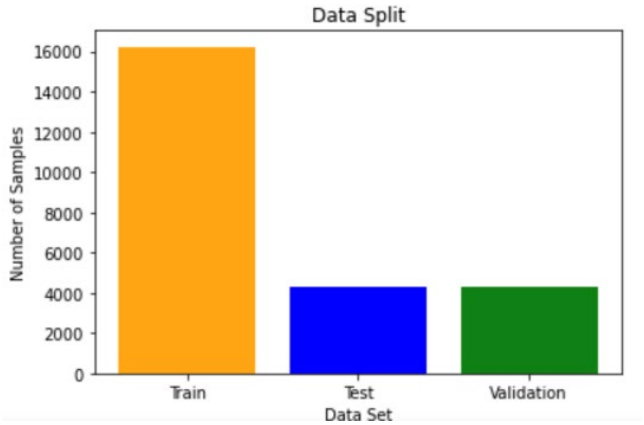
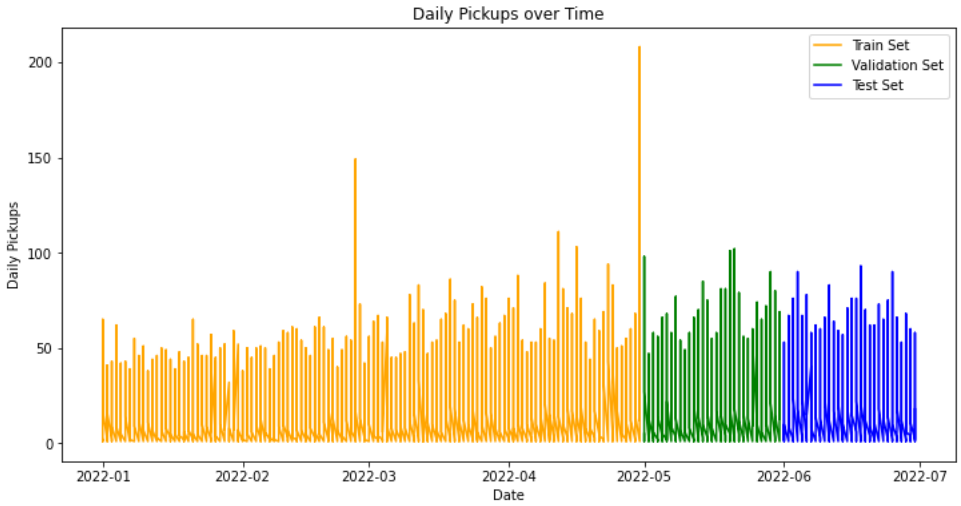


Figure 45

Line plot for Train, Test and Validation over Time



Note. Distribution of Train, Test and Validation Data based on Daily Pickups over Time

3.7 Data Analytics Results

The Figure 46 visualizes the trend in the number of daily pickups over a period of six months in 2022 using a line plot. Each point on the plot represents the number of pickups recorded on a specific date, providing an overview of the fluctuations in daily pickups during that time period.

Figure 46

Line graph on Number of Daily pickups

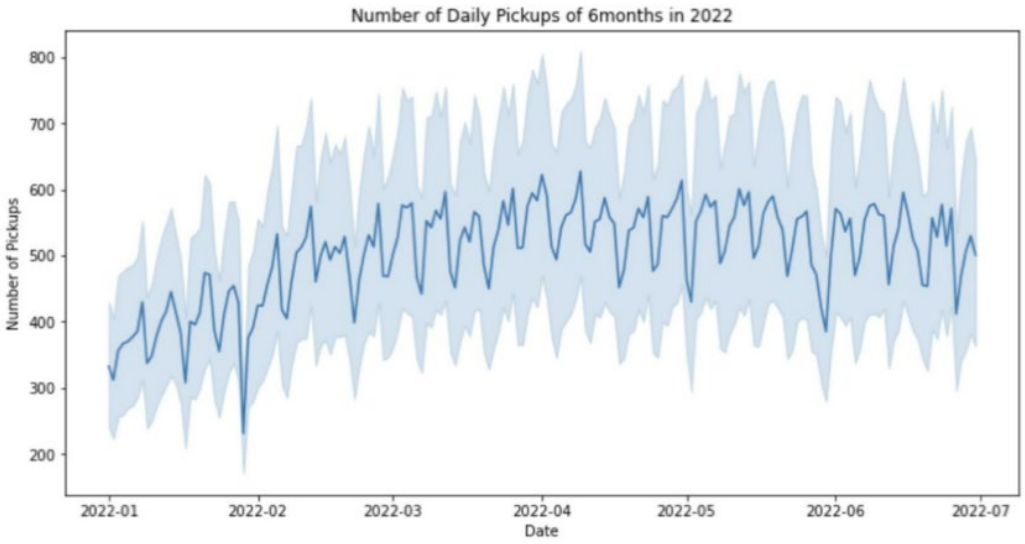
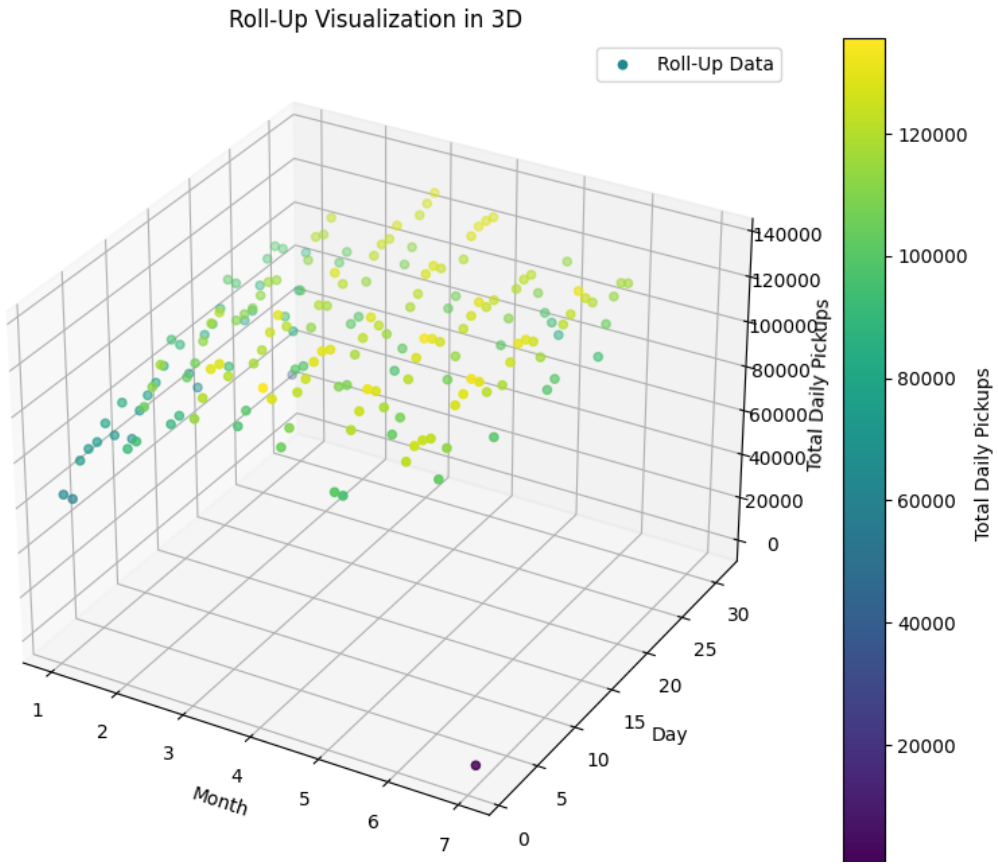


Figure 47

Total Pickups by Month and Day

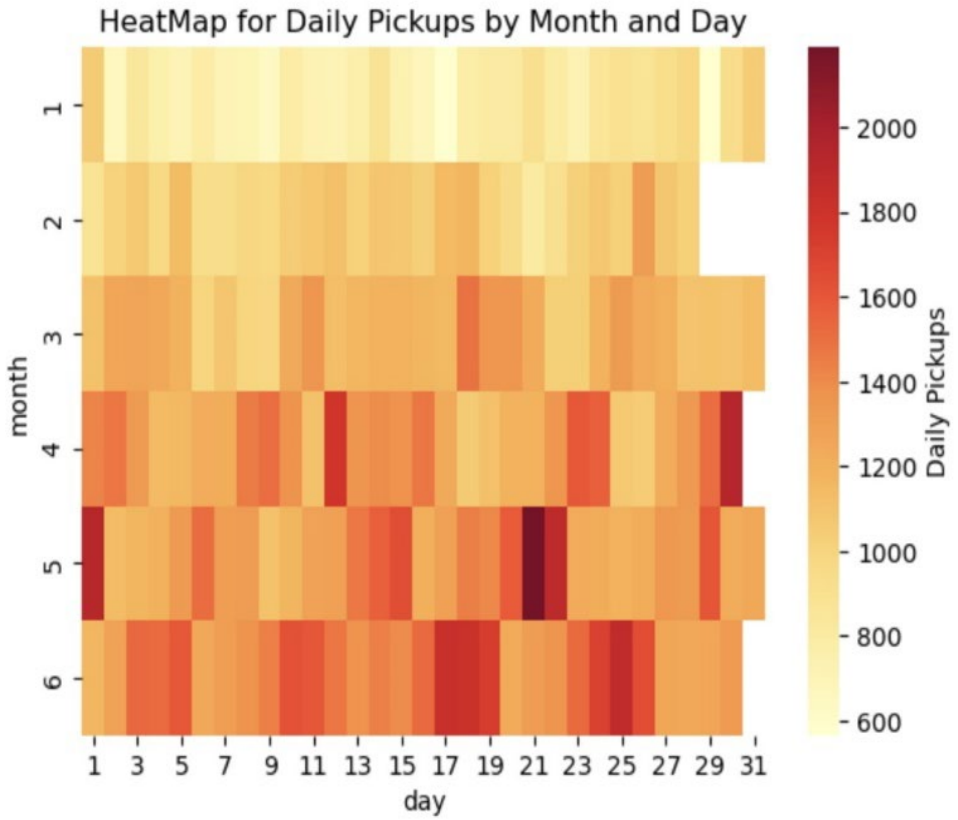


The Figure 47 is generated by performing roll-up visualization in a 3D plot using matplotlib. It groups the data by year, month, and day, and calculates the total daily pickups. The resulting data is plotted in a 3D scatter plot, where the x-axis is the month, y-axis represents the day, and z-axis represents the total daily pickups. The color of each point is determined by the value of the total daily pickups using the viridis colormap. The resulting plot provides a visual representation of the variations in total daily pickups across different months and days.

In the Figure 48, a heat map is generated to show the relation between months, days and number of pickups. This map makes it easier to know the trends and patterns of months and days on the pickups. It is clear from the map that may month have the highest number of pickups.

Figure 48

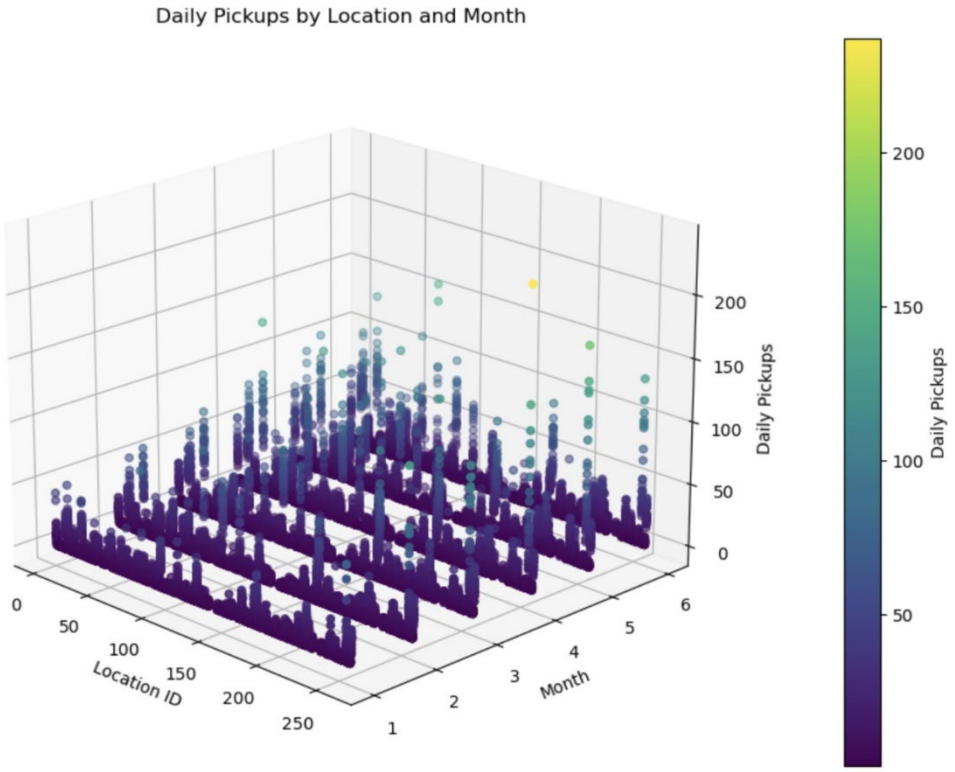
Heatmap for Daily Pickups by Month and Day



This graph shows the enhanced version of the heat map generated. This is a 3d plot where visualization is much more clearer.

Figure 49

Daily Pickups by Location and Month



MODEL DEVELOPMENT

4.1 Model Proposals

In the modeling phase, four algorithms were implemented that include Random Forest Regressor, Extreme Gradient Boosting, Facebook prophet and In the research, Li et al.

(2018) extracted different temporal features such as the day, day of the week, month, and year from the existing data using feature extraction techniques and transformed the time series data into a regression model for predicting the data-driven real-time taxi demand. In terms of model performance, they evaluated the standard benchmarks of different models such as ARIMA, SARIMA, and SVM and found that the random forest model outperformed the traditional time series model in terms of accuracy and efficiency. Another research by Liu et al. (2021) supported that random forest regressor performs better accuracy when compared to the regression models in energy consumption prediction. In both types of research, it is clear that predicting the future value from the past data random forest is the best-suited model. The model can better capture the underlying patterns in the data, resulting in more accurate predictions because of data consolidation from all the trees in the forest.

The significant reason for choosing XGBoost algorithm is that in a paper by Chen and Guestrin (2016), it is mentioned that XGBoost is known for its scalability as it uses automatic parallel and distributed computation, where multiple processors are used for computation by dividing a large problem into smaller ones which reduces training time. This paper also mentions the ability of XGBoost in handling Complex and large datasets and mentions their key features and the math behind them. In a paper by Noorunnahar et al. (2023) it is stated that XGBoost is a robust gradient-boosting algorithm that is ideally adapted for time-series tasks as it can handle missing values and feature interactions and prevent overfitting with its built-in Regularization. Therefore, the use of the XGBoost algorithm for the NYC driver demand project is an ideal choice as it consists of time series data where the key component of forecasting the target is dependent on it.

Prophet mainly focuses on forecasting any business model and performed well in terms of accuracy, better fitting, and does not have any fluctuations with null values as described in the research by Saiktishna et al. (2022). Moreover, according to Zunic et al.

(2021), this algorithm performed better when the data has a long historical record and has a frequency of pattern in the data. This model is highly adaptable and readily modifiable to account for seasonal and trend changes, making it an indispensable component for rapidly training hundreds of models and robust to outliers and missing values which are directly handled without much intervention in data preprocessing as per the authors Yang et al. (2019).

LSTM networks have been widely recognized for their capacity to handle long-range dependencies and preserve information over extended time intervals. They have displayed the superior performance of LSTM in time series prediction tasks and its successful application in various domains such as stock market forecasting, speech recognition, and natural language processing. According to Connor et al. (1994) and also by Schmidhuber (2015) the traditional artificial neural network (ANN) is limited in its capacity to identify the temporal dependencies of data based on time series. On the other hand, a (RNN) recurrent neural network is structured with temporally organized connections between units, thereby enhancing the modeling of time series characteristics.

Random Forest Regressor

It is one of the popular machine learning algorithms that combine the power of decision trees to create a robust and accurate regression model. This model mainly uses the ensemble learning approach with bagging to perform regression analysis. Ensemble learning is the process of predicting the target variable with the help of different machine learning algorithms or predicting the target variable from the same algorithm numerous times to predict more accurate predictions. It is very popular and widely used in the industry, as this model is generally less prone to overfitting problems when compared to other models such as decision trees. Decision trees are sensitive by nature with respect to the underlying data. If there is any change in the data, the predictions from the decision trees are highly corrupted.

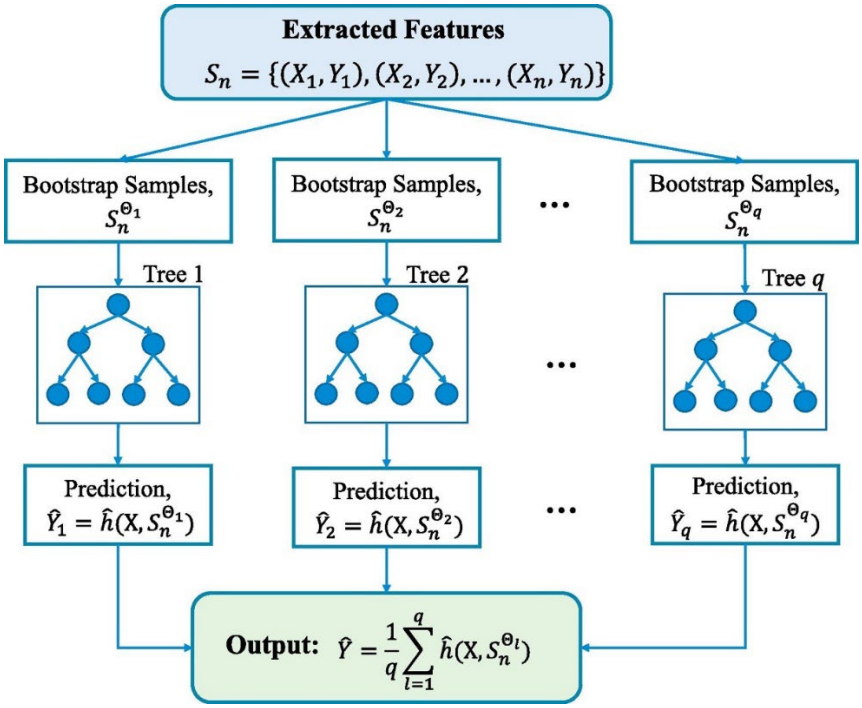
With the use of bagging along with the decision trees, it is less concerned about overfitting the training data for each decision tree.

According to Li et al., (2018) building a random forest algorithm has different stages and can be explained as follows. The process of training the model involves a number of steps. In the initial stages, the selected variables are identified, and then pass the data to the model as input parameters. Then, random subsets of the data are picked with the help of the bagging concept (also called Bootstrap aggregation) to build all trees in the forest. This technique is used to reduce the variance of an algorithm that has a high variance. Then each tree in the forest used the recursive binary splitting method to choose the best feature split based on the criteria using variance reduction or mean square error. In the final phase of predictions, the output from all the individual decision trees is collected and then combined to form the average of their predictions**.**

In Figure 50 the architecture for random forest regressor is given by, Li et al. (2018) and suggested formulae for calculating in the forest as it where $X = \{X_1, X_2, \ldots., X_n\}$ represents the input data, $Y = \{Y_1, Y_2, \ldots, Y_N\}$ values indicate the target data or final output data. Sn refers group of n samples as $S_n = \{ (X_1, Y_1), (X_2, Y_2), \ldots., (X_n, Y_n)\}$. At the time of training, several bootstrap samples $(S_1, S_2, \ldots, S_n)$ are created and applied to all trees in the forest to build q trees. This ensemble algorithm generates the output for each tree and produces q outputs in the forest, such as $\widehat{y_1} = \hat{h}(X, S_n^{\theta_1})$, $\widehat{y_2} = \hat{h}(X, S_n^{\theta_2})\ldots.$ $\widehat{y_q} = \hat{h}(X, S_n^{\theta_q})\ldots..$ Finally, consolidation is performed by taking the values from all the trees in the forest and producing the result as $\hat{y} = \frac{1}{q}\sum_{l=1}^{q} \hat{h}\ (X, S_n^{\theta_l})$.

Figure 50

Architecture for Random Forest Regression



Note. This Figure is taken from paper by Li et al. (2018)

In random forests, all the decision trees are run in parallel, which does not have any dependency or interaction while building the trees. Each decision tree in a random forest has a root node and is further divided into sub-nodes until it reaches its final leaf node. According to the research by Breiman (1984), each decision tree uses the CART algorithm for both random forest regression and classification problems. The CART algorithm is an iterative approach where the data is split in terms of a binary method. The partitioned data is divided into two subsets based on the selected variable and the threshold value at each node. This process is repeated until it reached the stopping criteria by getting the values of how many trees are needed and how many levels of data need to be split at each tree which is specified in the stop criteria. By default, 100 trees are defined with the help of n_estimators from the skit-learn library.

Research by Genuer (2012) mentioned that splitting criteria relies on the threshold value which is calculated by the variance reduction or mean square error to minimize the loss at the time of splitting. The threshold value at each node is defined based on the variance calculated, which acts as an impurity for the random forest regressor. The higher value of

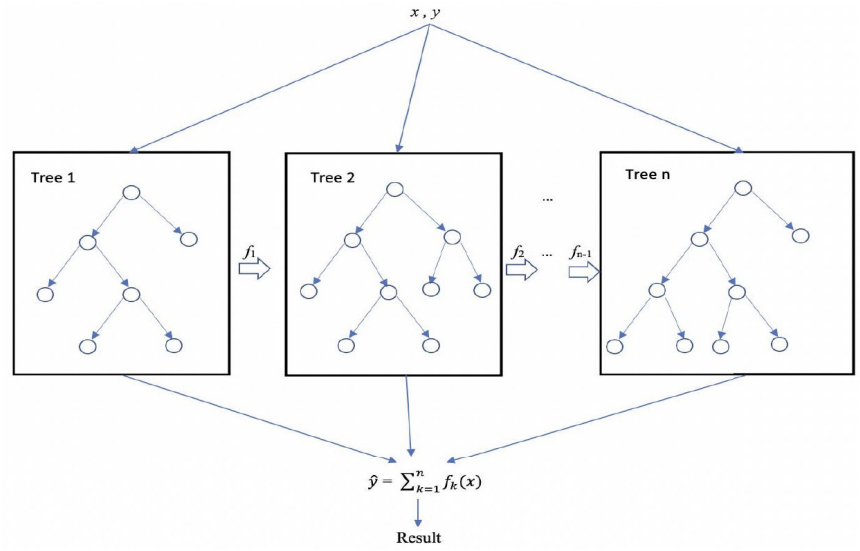
variance represents greater impurity in regressions. As a result, the model will consider the high variance value when it calculates the variance between the root node and child nodes. Variance reduction is used for the regression problems, while the classification problems use the entropy or Gini index.

XGBoost

The XGBoost algorithm considers each decision tree as a weak learner. Then combines all the weak learners together to make a stronger model by adjusting and improving fellow learners. This adjustment and improvement is made by training the model in a sequential way where weak learners try to correct the errors of the previous one.

Figure 51

XG-Boost Model Architecture



Note. Architecture of XGBoost taken from Anggraeni et al. (2021)

The above Figure 51 shows the architecture of XGBoost taken from the paper by Anggraeni et al. (2021).

From the above XGBoost architecture it can be understood that the final prediction score is the aggregate of all the constructed decision trees (improved weak learners). Below is the equation 1 from the paper by Chen and Guestrin (2016) explaining tree ensemble model

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F} \tag{1}$$

Here, we are basically predicting the i^{th} instance. Chen and Guestrin (2016) mentioned that F represents space of regression trees and f_k represents an independent tree structure.

The XGBoost algorithm first creates an initial decision tree using the data, and the loss will be calculated using the objective function. The model is trained by minimization of the regularized objective function. A loss function is combined with a regularization term to form an objective function. Below is the equation 2 to calculate the loss function taken from the paper by Chen and Guestrin (2016).

$$\mathcal{L}(\phi) = \sum_{i} l(\hat{y}_i, y_i) + \sum_{k} \Omega(f_k) \tag{2}$$

Here, *l* represents the deviation between predicted and actual objective value. There are several loss functions like MSE, RMSE, and R-square to calculate the errors of regression tasks. Out of Which, MSE is the most popular metric for the XGBoost algorithm. The algorithm's primary objective is to reduce these errors by improving the model's parameters. This loss function also helps in finding the best-split point of a tree. If a split point has the lowest loss function, it will be chosen as the optimal split point in the tree.

The MSE formula used by XGBoost is shown below in equation 3.

$$\ell = 1/n \, \Sigma \, (y_i - \hat{y}_i)^2 \tag{3}$$

Here, $\hat{y}_I$ represents predicted value, y_i represents actual true value.

The second term Regularization is used to prevent overfitting during the training of decision trees. Regularization is about including addition term to the loss function to rescue the model from overfitting the training set and from becoming overly complex.

Below equation 4 is the formula for Regularization, taken from the paper by Chen and Guestri (2016).

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2 \tag{4}$$

Here, λ is the regularization parameter of both L1 and L2 Regularization. Where w is the weight of the point.

In this manner, the XGBoost algorithm iteratively constructs new trees that prioritize data points that the previous tree incorrectly classified. New trees are added to the ensemble, and their predictions are combined with those of the previous ones. The tree is grown up until a stopping requirement, such as the maximum tree depth or the minimum sample size required to divide a node. This process is continued till a model performance is improved, which can result in a powerful performance of an ensemble model that predicts the values of a target feature. All this Process enables XGBoost to build strong decision trees that adapt well to new data and predict with reasonable scores.

Facebook Prophet

This additive model involves three components in the prophet model seasonality, holidays, and trend, as demonstrated in Equation 5 which was given by Taylor and Letham (2018).

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t. \tag{5}$$

Here, the value g(t) refers to trend functionality which captures the non-periodic pattern in the data that describes the fundamental movement and scale of time series data, thereby offering valuable insights into the general upward or downward trend and the associated rate of change. s(t) and h(t) captures the periodic pattern.

The term ϵ_t in the model signifies any unique variations that are not accounted for by the model. The trend model is further divided into two models namely the saturated growth

model and the piecewise linear model. Out of these two models, the default model is the linear model. However, In the downline, the logistic model is discussed first and then the linear model because the terms and parameters are clearly understood if there is a transition from the logistic model to the linear model as there are common parameters except the carrying capacity.

According to Taylor and Letham (2018) the constant value of carrying capacity lags in the real-time scenario because there is a chance that carrying capacity might increase with respect to time, and also the growth rate is not constant in every instance. There are a series of growth rate modifications S that take place at certain intervals and are designated as s_j for j = 1 to S and there is vector $\delta \in R^S$ to describe these changes, where j stands for the rate adjustment at time s_j. At a specific time point, t, the rate is calculated by multiplying the initial growth rate k and all previous adjustments and the value becomes $k + \Sigma_{j:t>sj} \delta_j$. Using a binary vector with the values $a(t) \in \{0,1\}^S$ is given as shown in Equation 6 which was given by Taylor and Letham (2018).

$$a_j(t) = \begin{cases} 1, & \text{if } t \geq s_j, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

After modifying the carrying capacity and growth rate with respect to time, the offset value m also has to be adjusted along with the other parameters which ultimately modify the logistic model equation. On similar lines, the linear model is also defined with the same parameters except for the exponential function and the carrying capacity. Both the equations which were given by Taylor and Letham (2018) are shown in Equation 7 and Equation 8.

$$g(t) = \frac{C(t)}{1 + \exp(-(k + \mathbf{a}(t)^\intercal \boldsymbol{\delta})(t - (m + \mathbf{a}(t)^\intercal \boldsymbol{\gamma})))}. \tag{7}$$

$$g(t) = (k + \mathbf{a}(t)^\intercal \boldsymbol{\delta})t + (m + \mathbf{a}(t)^\intercal \boldsymbol{\gamma}) \tag{8}$$

The above Equation 8 is the main function used for the present project to predict the trend line of taxi demand. The above linear model includes all the parameters that are mentioned in the logistic but the offset parameter gamma value varies as $-s_j * \delta_j$. Mathematically, seasonality component follows the Fourier series. The dataset of this project involves monthly ad daily seasonality and for the y(t) outcome it is represented as s(t) as shown in Equation 9 which was given by Taylor and Letham (2018)

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right) \tag{9}$$

Figure 52

Pseudo Code For The Prophet Model with only linear model enabled

```
model {
  // Priors
  k ~ normal(0, 5);
  m ~ normal(0, 5);
  epsilon ~ normal(0, 0.5);
  delta ~ double_exponential(0, tau);
  beta ~ normal(0, sigma);

  // Logistic likelihood
  y ~ normal(C ./ (1 + exp(-(k + A * delta) .* (t - (m + A * gamma)))) +
             X * beta, epsilon);
  // Linear likelihood
  y ~ normal((k + A * delta) .* t + (m + A * gamma) + X * beta, sigma);
}
```

Note. This Pseudo Code is taken from the paper by Taylor and Letham (2018)

Next, The Holiday component (h(t)) that is present in this prophet model is not used in the prediction process as the data does not contain any holiday or event-related data. that works with the approximation and normalization of initial value k, offset m, and delta δ in the

trend component, and beta β in the seasonality component. Above Figure 52 is the Pseudo code for the prophet model as described in the paper Taylor and Letham (2018).

Long term Short Memory(LSTM)

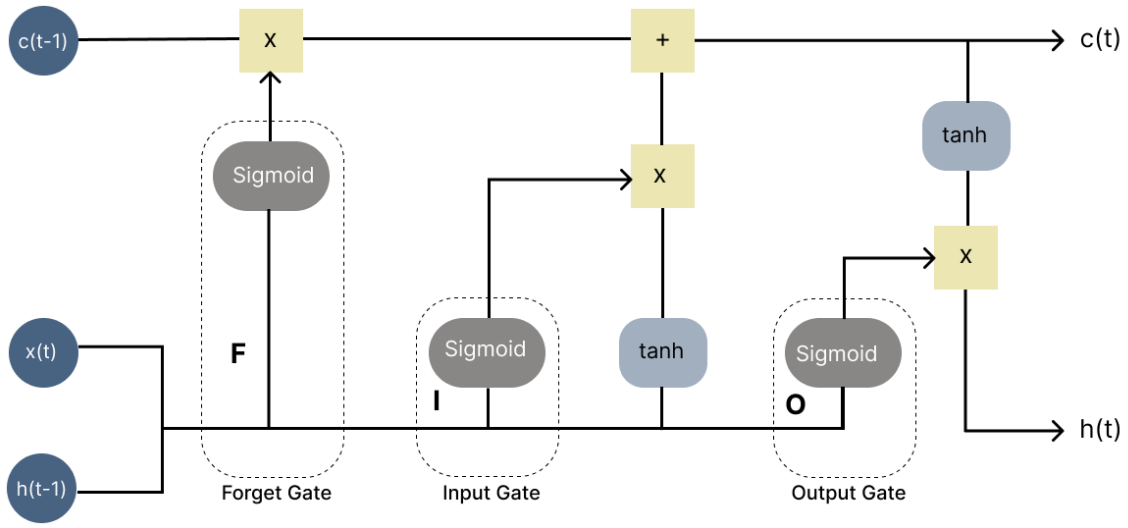
The LSTM architecture is distinctively engineered to mitigate the challenge of long-term dependency and has demonstrated remarkable efficiency across a diverse range of problem domains. A unit in LSTM consists of three main gates which are specifically known as the input, output and the forget gates. The usage of these three gates is mainly to control the passage of information through the unit. The chained structure throughout the gates transmits both a concealed state and the last cell.

The shared knowledge of the concealed state and cell state can counteract the effects of vanishing gradient descent. In addition to maintaining a cell state, the LSTM unit is able to transport information across time steps. The forget gate eliminates unnecessary the data of the preceding cell state after gaining the knowledge of the present input, i.e., the incoming data, and compares it with data from the previous hidden state and these are also called as dark layers of LSTM.

The input gate makes decision regarding which new information is to be taken as the incoming source for the cell state at that instance of a time step, depending on what is currently taken as the input and past hidden state. Whereas the output gate controls which information will be the output by considering the updated cell state and present input. The output may be used to make a prediction or as input for the subsequent LSTM cell in the sequence. The below Figure 53 illustrates the LSTM Layer Architecture in more detailed manner.

Figure 53

LSTM Layer Architecture



Note. The structure has been adapted from the work by Hochreiter and Schmidhuber (1997)

In an LSTM (Long Short-Term Memory) network, information is processed at each time step using the input sequence x(t) and the previous hidden state h(t-1). This results in an updated hidden state h(t) and cell state c(t) as shown in below Table 13 and pseudocode is given in Table 14.

Table 13

Formula for Gates and Cell Updates in LSTM

Purpose	Formula
Forget gate	$f(t) = \sigma(W_f * [h(t-1), x(t)] + b_f)$
Input gate	$i(t) = \sigma(W_i * [h(t-1), x(t)] + b_i)$
Cell State Update	$c'(t) = \tanh(Wc * [h(t-1), x(t)] + bc)$
Updated cell state	$c(t) = f(t) * c(t-1) + i(t) * c'(t)$
Output gate	$o(t) = \sigma(W_o * [h(t-1), x(t)] + b_o)$
Hidden state	$h(t) = o(t) * \tanh(c(t))$

Note. These formulae are mentioned by Hochreiter and Schmidhuber (1997) in their study

Table 14

Pseudocode for LSTM layer

Algorithm 1 Pseudocode of LSTM layer

1: **Intialize the LSTM parameters: weights W_f, W_i, W_c, W_o and biases b_f, b_i, b_c, b_o**

2: **Intitialize LSTM states: h_0 and c_0**

3: **For each time step t:**

Calculate the input gate $i(t) = \sigma(W_i * [h(t-1), x(t)] + b_i)$

Calculate the forget gate $f(t) = \sigma(W_f * [h(t-1), x(t)] + b_f)$

Calculate the Cell state update $c(t) = \tanh(W_c * [h(t-1), x(t)] + b_c)$

Update the cell state using $c(t) = f(t) * c(t-1) + i(t) * c(t)$

Calculate the output gate $o(t) = \sigma(W_o * [h(t-1), x(t)] + b_o)$

Calculate the hidden state $h(t) = o(t) * \tanh(c(t))$

4: **Return the final hidden state h(t) as the output**

Note. Adapted based on the study by Hochreiter and Schmidhuber (1997)

The forget gate is controlled by the sigmoid activation function (σ), weight matrix (W_f), prior hidden state (h(t-1)), input at present time step will be (x(t)), and bias term (b_f). The updated cell state (c(t)) is obtained with the help of a special function called hyperbolic tangent (tanh) will be used element-wise for output of the linear transformation of the concatenation of weight matrix (W_c) with (b_c) which is the bias term, h(t-1) and x(t), where (tanh) stand for the activation function. The output is computed on the basis of the present cell state and the output gate which is given by h(t) for LSTM at time t.

LSTM (Long Short-Term Memory) networks are capable of incorporating multiple hidden layers, consisting of memory cells are positioned among the two different layers of input and output. The memory cell will be triggered upon the triggered within a layer is determined by an activation function that computes the weighted sums by incorporating bias. This inclusion of bias facilitates the introduction of non-linearity into the output of a memory cell. In LSTM networks, two widely used activation functions are Rectified Linear Units (ReLu) and Sigmoid. ReLu activation is commonly employed in the hidden layers of LSTM networks, producing output values ranging from zero to infinity. According to Yang et al. (2020b) the Sigmoid (logistic) activation function is used in the output layer of LSTM

networks, especially when predicting probabilities. The sigmoid function ensures that the output falls within the range of zero to one.

4.2 Model Supports

Environment, Tools and the Platform

The models were initially built and evaluated on the local machine using Jupyter Notebook before being executed on the Google Colab server. The project's later code is uploaded to colab. The Google account is protected by multi-factor authentication (MFA), which acts as an additional layer of security and prevents fraudulent activity. To manage permissions, a group was constructed, and everyone in the group was granted permissions based on their default and custom responsibilities. The processed data are stored on Google Cloud's container, and the data is distributed across two distinct regions, namely US Central and US West. Data distributed across multiple regions can be of great assistance in disaster recovery.This project's primary hardware component is an Apple M1 Pro processor equipped with a 12-core CPU, 16-core GPU, 16GB RAM, and 512 SSD storage. All of these hardware components enable models to execute more quickly by maximizing component utilization. In addition, the security of a MAC operating system is superior to that of other operating systems. Google Colab and Jupyter Notebook were used as the Integrated Development Environment (IDE) in this project.

Even though a high-end configuration of the Apple M1 processor with the requisite Python notebooks is utilized, the notebook cannot be shared so that multiple users can simultaneously work on the same file. Google colab was utilized to avoid this scenario and to tackle the problem of concurrent users operating on the same notebook file. In addition, Google Colab incorporates GPUs and TPUs for use in model development.

Python, which is supported by both the Google Colab and Jupyter Notebook platforms, is used for the entirety of the development in the present research. This language

has a large number of libraries that provide a variety of functions that facilitate the development and implementation of machine learning algorithms, as well as their evaluation using standard metrics. This development makes extensive use of the Python libraries Pandas, NumPy, Pyarrow, Seaborn, DateTime, Matplotlib, Scipy, and Sklearn. The libraries used in the project are mentioned below in detailed in Table 15

Table 15

Libraries Used for Model Development

Libraries		Method	Purpose
Pandas	Dataframe	info, describe, column, drop,replace	Used to read ,write and modify the data
Numpy	random	seed	Use of arrays which help to modify the data
Matplotlib and Seaborn	pyplot		Visualization are performed
	sklearn.ensemble	RandomForestRegressor	Implement the random forest regressor algorithm
	sklearn.metrics	root_mean_squared_error, mean_squared_error, r-squared, mean_absolute_percentage_error	Evaluation of the model is performed using various metrics.
	sklearn.model_selection	train_test_split	Splitting of data into train, validation and test
	sklearn.model_selection	GridSearchCV	Best hyperparameters is set using grid search
	sklearn.decomposition	PCA	Dimensionality Reduction
	sklearn.linear_model	Lasso and Ridge	Regularization of data is done
scikit	sklearn.preprocessing	StandardScaler, MinMaxScaler	Data is normalized
	prophet	Prophet	Implementation of prophet algorithm with its parameter tuning
	prophet.diagnostics	cross_validation	Cross validation is implemented
prophet	prophet.plot	plot_cross_validation_metric	Metrics of prophet model is calculated
	keras.models	Sequential	Initializes a sequential model in Keras.
	keras.layers	Dense	Adds a fully connected dense layer to the model.
	keras.layers	LSTM	Adds an LSTM layer to the model.
		add	Adds layers to the sequential model.
		compile	Compiles the sequential model with a specified loss function and optimizer.
		fit	Trains the model using the provided data, specified epochs, batch size, and verbosity level.
keras		predict	Makes predictions using the trained model on the data.
xgboost	xgb	XGBRegressor	Extreme Gradient Boosting is implemented

Data Flow and Architecture

The data has been preprocessed from its raw form with the help of all these libraries. Following preprocessing, the dataset contains 26456 rows and 8 columns, including the desired column 'Number of Pickups.' The available processed data is divided into three categories: Training, Validation, and Testing. In time series prediction, ratio-based data

division cannot be used because the data must be ordered with respect to date and time. If the data is split using sklearn.train_test_split, it is split randomly, and the order of the date column in the dataset becomes inconsistent and unacceptable, which completely deviates from the sequential pattern necessary to predict the number of collections. Instead, it is divided by month, with the first four months, from January 2022 to April 2022, containing training data, followed by validation data and test data, respectively.

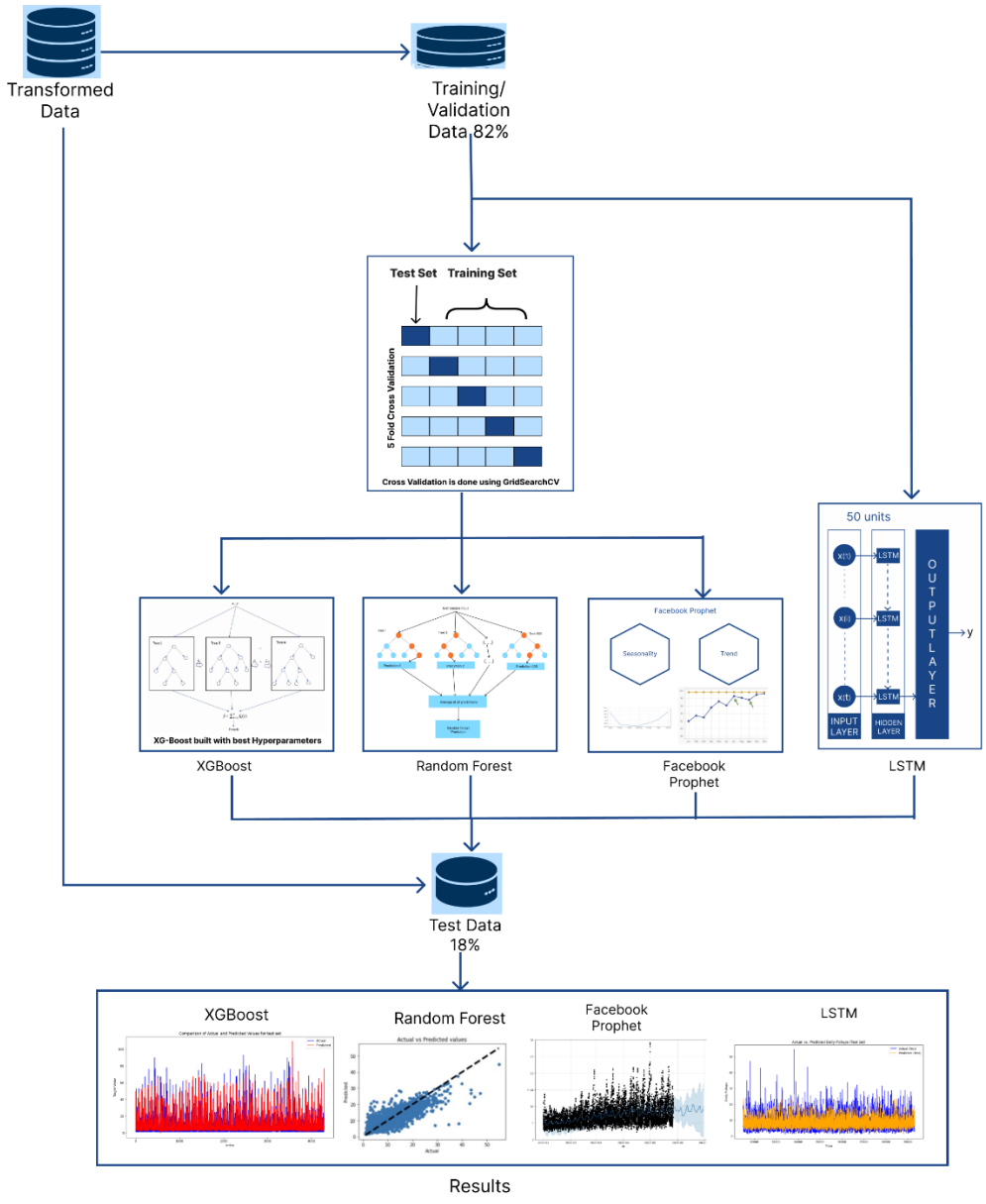
The libraries used to import the algorithms are xgboost, prophet, keras.layers and sklearn.ensemble.RandomForestRegressor and are developed using hyperparameter tuning. For both Random Forest and XGBoost ensemble models, a param_grid dictionary containing various hyperparameters is utilized. n_estimators is one of the most frequently used parameters, as it helps to suit the model with multiple trees. When the model selects a larger number of trees, the performance may be enhanced, but there is a risk of overfitting; therefore, it is essential to select the number based on the data and problem statement. Even if it is the same as the max_depth parameter, increasing the depth of the tree improves performance, but there is a risk of overfitting if the number is excessively large. Using learning_rate it prevents overfitting. In general, the values of the learning rate are reduced; the shrinkage step value or learning rate value for this undertaking is 0.01. In order to prevent overfitting, the subsample and colsample_bytree parameters are also used to randomly sample fractions of observations and features for each tree. Moreover, increasing the value of the gamma parameter makes the model more conservative.

In addition, the prophet model is fine-tuned based on several variables, including change_point_priority_scale, seasonality_priority_scale, and seasonality_mode. The changepoint_priority_scale controls the sensitivity of the Prophet model's trend to changes in the underlying data. The parameter seasonality_prior_scale governs the adaptability of the seasonal components, thereby influencing their ability to capture the inherent trends in the

data. The seasonality_mode parameter specifies whether seasonal effects are assumed to be additive or multiplicative .

Figure 54

Architecture and Model Support Diagram



The LSTM parameters are the dropout rate, the learning rate, and the activation function. Where the dropout employs regularization to prevent overfitting in the model. The learning rate, whose value is 0.05, also inhibits from overfitting. Tuning the layers of an

LSTM model can also enhance its performance. In addition, window sizes are increased during modeling to improve the model's prediction accuracy.

All of these parameters are optimized using a grid search. Grid search examines numerous hyperparameter combinations and identifies the optimal one. This grid search therefore employs cross-validations with a CV value of 5 to evaluate the efficacy of each hyperparameter combination. As the training set is fitted to the grid-search object, the grid_search.best_estimator parameter returns the optimal set of hyperparameters. After tuning with the optimal hyperparameters, the model is prepared to predict test data, and the results are compared to those of other developed models. The above Figure 54 illustrates the architecture of the whole project.

4.3 Model Comparison and justification

Though any machine learning algorithm that can handle multivariate time series data is a good choice for the NYC driver prediction, there are a few strengths and weaknesses for each of them; where the algorithm that is chosen should be based on the data which is to be analyzed or forecasted. Because of the parallel processing architecture of random forest, it can produce better results when compared with the other models.

When it comes to XGBoost, it is known for its scalability, where it has a high computation speed compared to other models. As the project is of time-series data, the seasonality, trends, and pattern finding of the data is essential; this can be easily handled by Prophet when compared to the others.

LSTM is the most powerful algorithm when it is provided with a huge amount of data and gives the best prediction results out of all the other metrics. Because of all these advantages, algorithms have been chosen for the current project. The below Table 16 gives the comparison among all the models.

Table 16

Model Comparison

Models	Advantages	Disadvantages
Random Forest Regressor	Handles high-dimensional data well	May overfit with noisy data
	Provides feature importance feature	Slower on large datasets
	Robust to outliers and also good with small datasets	Limited interpretability
	Reduces overfitting through bagging	Unable to capture complex non-linear patterns
XG Boost	Excellent performance and accuracy	Prone to overfitting with high learning rates
	Handles complex feature interactions	Requires careful tuning of hyperparameters
	Provides feature importance feature	Relatively slower training time
	Supports parallel processing	Memory-intensive for large datasets
LSTM	Capture long-term dependencies	Sensitive to the choice of hyperparameters
	Handles sequential and time series data	Requires larger amounts of training data
	Model easily detects complex non-linear patterns	Longer training time compared to other models
Prophet	Specifically designed for time series forecasting	Unable to deal with high Dimensional data
	Handles missing data and outliers	Not suitable for complex non-linear patterns
	Provides intuitive model diagnostics	Limited flexibility for custom model changes
	Incorporates seasonality and trends	Require manual feature engineering

4.4 Model evaluation metrics

Mean Squared Error

According to Géron (2019), MSE is a statistical metric commonly used in time series data. By squaring the differences, MSE amplifies the impact of larger errors, making it a valuable metric for understanding the overall goodness of fit of the model. When the MSE values are low the model predictions are similar to the observed values suggesting higher accuracy.

Equation 10 gives a formula for calculating MSE that involves obtaining the summation of the squared variations between the anticipated and factual values, which is then divided by the overall count of observations in the dataset. This average squared difference provides a quantifiable measure of the model's performance in minimizing prediction errors:

$$MSE = (1/n) * \Sigma(\hat{y} - y)^2 \quad (10)$$

R-square

Another important metric for evaluating the LSTM model's performance on a time series dataset is R-square, also known as the coefficient of determination. Hastie et al. (2013) states that it measures variance of the data that can be explained by the independent variable which are the model's predictions. When the value is near to 0 it indicates that the model is predicting incorrect values and not much of variance is explained. Whereas when it is near to 1 the variance is clearly explained and indicates that the model is performing well.

The determination coefficient (R^2) shown in equation 13 is derived through the computation of the Sum of Squared Residuals (SSR) given as equation 11, and its comparison with the total sum of squares (SST) where the calculation is given as equation 12.

$$SSR = \Sigma(y_i - \bar{y})^2 \quad (11)$$

$$SST = \Sigma(y_i - \bar{\bar{y}})^2 \quad (12)$$

$$R\text{-square} = 1 - (SSR/SST) \quad (13)$$

A higher R-square value helps the model in explaining a larger proportion of the variability in the time series data, indicating a better fit. However, it's important to note that R-square alone may not provide a complete information of the model's performance, since it can be influenced by the complexity of the dataset and the number of predictors incorporated into the model.

Mean Absolute Percentage Error

In addition to MSE and R-square, another commonly used evaluation metric for the data collected and analyzed over time is Mean Absolute Percentage Error (MAPE) as

mentioned by Hastie et al. (2013). MAPE is a metric that quantifies the average percentage change between the values that are predicted (ŷ) and the actual numbers (y) in a given time series dataset. This metric provides a relative measure of the model's accuracy, allowing for comparisons across different time series datasets.

The equation 14 shows the formula for calculating MAPE involves taking the absolute difference between the values which are predicted and actual/original values of the dataset, dividing it by the actual value, and multiplying it by 100 to express it as a percentage. By averaging these percentage differences across all observations in the dataset, MAPE provides an overall measure of the model's accuracy:

$$\text{MAPE} = (1/n) * \Sigma(|(\hat{y} - y)/y|) * 100 \quad (14)$$

MAPE is particularly useful when the magnitude of the errors is the pri and needs to be expressed as a percentage of the actual values. However, it's worth noting that MAPE has a limitation of being sensitive to datasets with zero or close-to-zero actual values, as it can result in undefined or extremely large percentage errors.

By considering MSE, R-square, and MAPE together, we gain a comprehensive understanding of the LSTM model's predictive capabilities and its accuracy in forecasting future values in a time series dataset

Root Mean Square Error

RMSE serves as a widely used metric for evaluating the accuracy of predictive models in time series analysis mentioned by Hyndman and Athanasopoulos (2018). It calculates the mean magnitude of differences between predicted and observed values across a sequence of time steps. Particularly in the context of the data having time as the main feature, RMSE considers sequential nature of the information, making it an effective tool for assessing the performance of forecasting models like autoregressive models, moving average models, ARIMA, or RNNs.

At each time step the differences are measured between the factual and predicted data. It enables researchers and practitioners to quantitatively analyze the performance of their models and compare them to alternative approaches given by Hyndman & Athanasopoulos (2018). RMSE also assists in identifying potential forecasting errors and guides the refinement of models to enhance their predictive capabilities. To compute RMSE the formula is shown in equation 15, one needs to take square root of the MSE which is the mean of the squared variances between the expected and factual values at each time step.

$$RMSE = \sqrt{(1/n * \Sigma(\hat{y} - y)^2)} \tag{15}$$

Model Evaluation and Validation

Initially the base models are used to train the data and make predictions on the test data. As the model is not fine-tuned, it is possible that the predictions will not be accurate, but this helps the model discover the true patterns by preventing it from becoming hyperparameter-sensitive. Then the models are tuned with their hyperparameters to make the predictions more accurate. The results before and after hyperparameter tuning are shown below.

Random Forest

Results Before Hyperparameter tuning

Figure 55

Results Before Hyperparameter Tuning

```
Results Before Hyperparameter Tuning

validation Set:
MSE: 12.36165143922335
RMSE: 3.51591407842602
R-squared: 0.883673694694515
Test Set:
MSE: 15.74201645118868
RMSE: 3.967621564080605
R-squared: 0.8524553998439073
```

The baseline default parameters used by random forest are number of estimators of 80, the maximum depth of 6 and the random state is 42. The Figure 55 illustrates the result of MSE, RMSE and R-square respectively. RMSE results are 3.51 and 3.96 as shown in Figure 56. These results look quite good when compared to the MSE metric.

Results after using Hyperparameter tuning

After identifying the best or optimal parameters for this dataset, the model should undergo the evaluation process again and be analyzed with the same metrics. Figure 56 captures the best parameters with the help of the GridSearchCV algorithm.

Figure 56

Best Score results

```
Best parameters:  {'max_depth': 15, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150}
Best score:  0.7849567662701178
```

After looking at the results again, hyperparameter tuning plays a crucial role in identifying the best parameters required for the model and increasing the accuracy scores for both the validation and training datasets. Figure 57 represents the complete results for both validation and testing datasets, explaining the three metrics after hyperparameter tuning.

Figure 57

Results After Hyperparameter Tuning

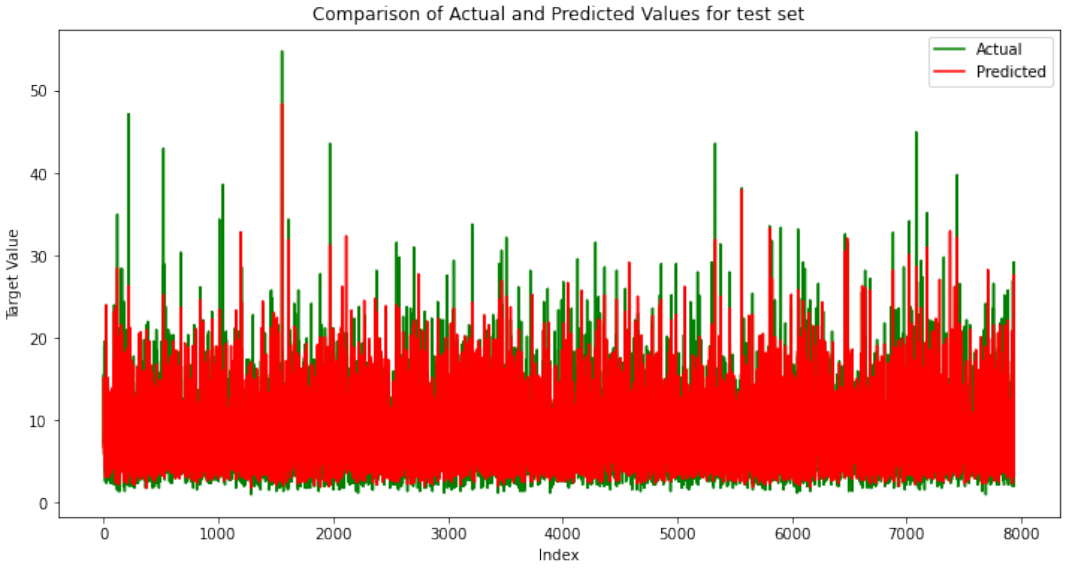
```
Results after Hyperparameter Tuning

validation Set:
MSE:  5.159834824624151
RMSE: 2.871626979770215
R-squared: 0.9312115854733723
Test Set:
MSE: 8.8207452476765294
RMSE: 2.978683488085655
R-squared: 0.9040583015999093
```

Looking at the MSE scores, the MSE value has increased to 9.040 from 8.167. This is a marginal difference in the value change, and the algorithm works properly by having no major impact on this metric. RMSE doesn't have much impact and gave similar results for this metric. However, the R-squared metric showed little change and decreased the efficiency from 0.932 to 0.904. Figure 58 represents the complete results for both validation and testing datasets, explaining the three metrics after hyperparameter tuning.

The below plot give information about actual and predicted values by the random forest algorithm. The red line indicates the predicted values by the model while the green line indicates the actual values. Figure 58 indicates almost 90 percent of the data is accurately predicting the target variables.

Figure 58

Comparison of Actual and Predicted Values for Test Dataset



XGBoost

Results Before Hyperparameter tuning

The baseline model default parameters for XGBoost algorithm are number of trees of 80, maximum depth of 6 and learning rate of 0.03. So the modelling will be carried based on

these parameters.The scores obtained after executing the baseline model are depicted in Figure 59

Figure 59

Validation and test results for baseline model

```
Validation Set Metrics:
MSE: 14.55431791618272
RMSE: 3.8150121777240398
R-squared: 0.8634814608604374

Test Set Metrics:
MSE: 16.116241510944207
RMSE: 4.0145038935021855
R-squared: 0.8174732540038019
```

The difference in error is more, so hyperparameter tuning must be applied to reduce it.

Results after hyperparameter Tuning

Grid search is used with cross-validations, CV= 5 to evaluate the performance of each of the hyperparameter combinations. And the learning rate is set to 0.1 to make the model robust to overfitting.

Moreover, the other parameter n_jobs, which is set to -1, which specifies that all CPU cores are to be used, which can speed up the search process.

As the fitting of the training set to the grid-search object is done, the best set of hyperparameters is obtained with the grid_search.best_estimator parameter.Below Figure 60 shows the best hyperparameter found for this model.

Figure 60

Best Hyperparameters for the model

```
Best parameters found:  {'colsample_bytree': 1.0, 'gamma': 0, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 1
00, 'subsample': 0.5}
```

Below Figure 61 shows the results of all the metrics used to evaluate the performance of validation and test sets.

Figure 61

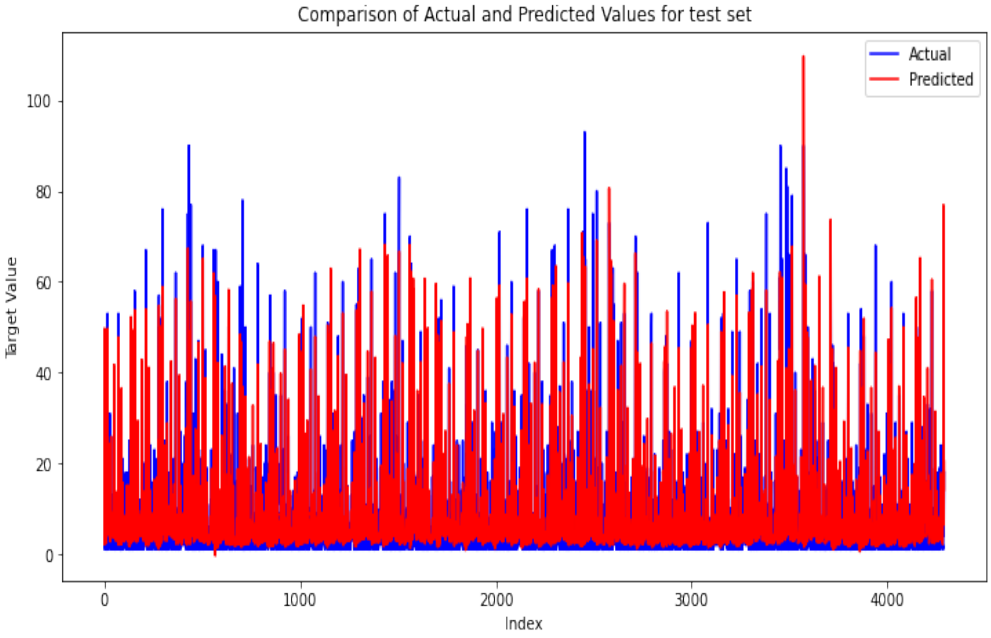
Validation and Test Results for XGBoost Model after using hyperparameters

```
Validation Set Metrics:
MSE: 8.246241510944207
RMSE: 2.871626979770215
R-squared: 0.9218115854733723

Test Set Metrics:
MSE: 9.047452476765294
RMSE: 3.0078983488085655
R-squared: 0.8940583015999093
```

Below Figure 62 show the visual representation on comparison between predicted and the actual values.

Figure 62

Actual vs predicted results comparison



Prophet

The results that are performed before hyper parameter tuning involve the default parameters with change_point_prior_scale (Flexibility of changepoints regularization) seasonality_prior_scale (Strength of seasonality modeling), and seasonality_model (Type of

seasonality modeling) which are set to the values 0.05, 10, and additive respectively. The below Figure 63 gives the result before hyper parameter tuning.

Results before Hyperparameters tuning

Figure 63

Validation and test results for baseline model

```
Before Hyper Parameter Tuning
Validation Set:
MAPE: 0.379439563465
R2 Score: 0.573985348756
RMSE: 3.348873465845723
MSE: 11.2112347568532

Testing Set:
MAPE: 0.421837465856
R2 Score: 0.5108374659702
RMSE: 3.8403487562785
MSE: 14.7423569293846
```

Results after Hyperparameters tuning

The Grid Search method is used to tune the parameters of this model. The deciding factor for setting the values of parameters is the MAPE Score. This approach systematically explores a range of parameter permutations in order to identify the best-performing parameter configuration. The process involves iterative parameter adjustment and selection of the combination that results in the minimum MAPE score. The best Hyperparameters are found which are shown below in Figure 64

Figure 64

Best Hyperparameters After Tuning

```
Best hyperparameters:
changepoint_prior_scale: 0.01
seasonality_mode: multiplicative
seasonality_prior_scale: 0.01
```

After modifying the tuned parameters the result has improved for every metric, especially for the R squared score. It showed a significant increase from 0.57 to 0.681 in the validation or

training accuracy. This shows how the changepoint_prior_scale has a great impact in improving the R^2 score which helps to predict the changepoints in the timeline and perfectly fits with the data. The below Figure 65 shows the results after hyperparameter tuning is performed

Figure 65

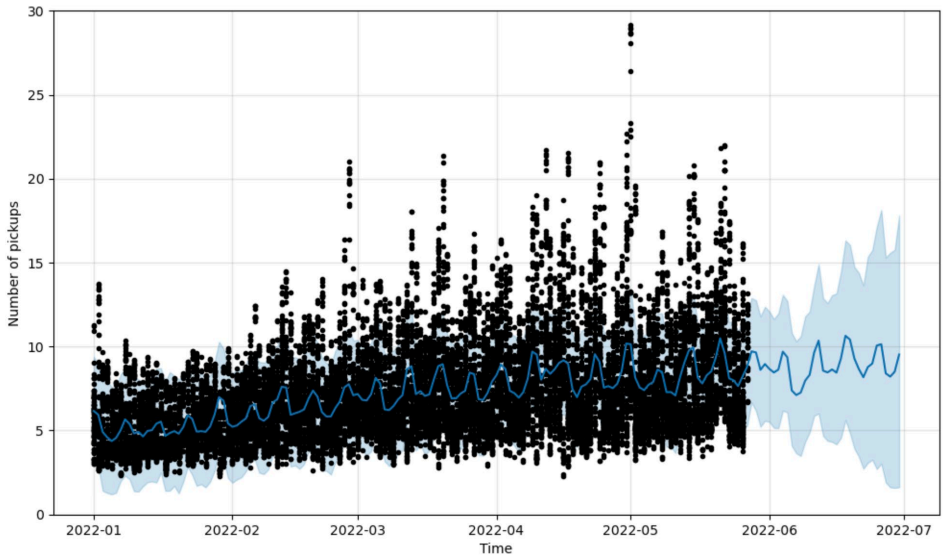
Validation and Test Results for Prophet Model after using hyperparameters

```
After Hyper Parameter Tuning
Validation Set:
MAPE: 0.32522474973587
R2 Score: 0.681434645934639
RMSE: 2.990573648543649
MSE: 8.94013498374923

Testing Set:
MAPE: 0.3387732320156649
R2 Score: 0.662378463873578
RMSE: 3.0887638464837
MSE: 9.409754264858394
```

The future prediction of the month June 2022 is plotted with the prophet model trained with the hyperparameters. The forecasting projection for the future month of June 2022 is shown in Figure 66

Figure 66

The forecasting of the number of pickups using Prophet Algorithm



LSTM

Results after Hyperparameters tuning

During the validation phase, the model achieved an MSE of 14.34 and an RMSE of 3.78. The R-Square value of 0.74 indicates that the model provides 74% of the variance in the validation dataset. However, the MAPE value of 0.41 suggests that the model struggled to capture the underlying temporal patterns during training. This indicates the need for further improvement in capturing the complex dynamics of the time series data. The below Figure 67 shows the validation results of LSTM model.

Figure 67

LSTM result for Validation dataset

```
LSTM Evaluation:
Validation Data:
MSE: 14.344980671756037
RMSE: 3.7874768212830077
R-Square: 0.7379717625842463
MAPE: 0.40803117651654
```

Moving on to the test data evaluation, the LSTM model obtained an MSE of 13.57 and an RMSE of 3.68. The slightly lower MSE and RMSE compared to the validation phase indicate a relatively better performance on unseen time series data. The R-Square value of 0.71 implies that the model is attaining for 71% of the variance in the previously unseen test dataset. However, it is important to note that the MAPE increased to 0.44, suggesting a relatively weaker performance in capturing the underlying patterns compared to the validation phase. This indicates a need for further refinement to enhance the model's accuracy in capturing temporal patterns on unseen data.

Overall, the LSTM model demonstrated a moderate degree of generalization ability within the time series context, as evidenced by a test set accuracy of 0.572. The results of validation set and the actual vs predicted are shown in Figure 68 and Figure 69 respectively.

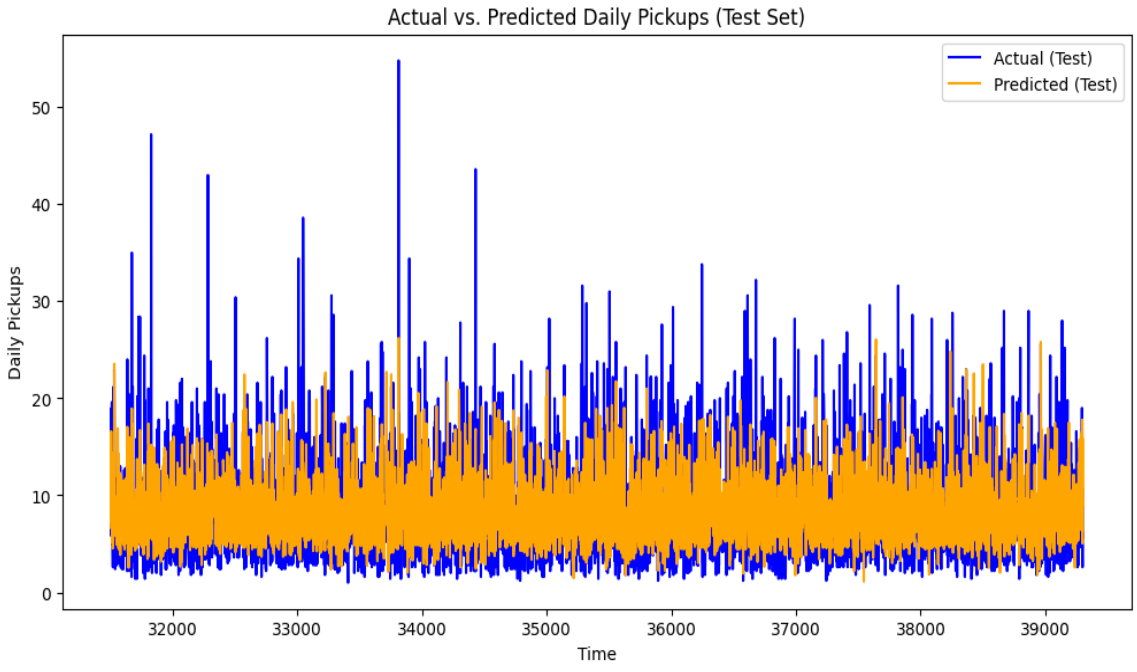
Figure 68

LSTM result for testing dataset

```
LSTM Evaluation:
Test Data:
MSE: 13.569547866772124
RMSE: 3.683686722126642
R-Square: 0.7123498486745695
MAPE: 0.43612637415131
```

Figure 69

Actual vs Predicted Values of Test Dataset



Final Results comparison:

The below Table 16 represents how the four algorithms are compared based on various metrics.

Table 16

Model Comparison Metric Results

Models	Validation			Testing		
	RMSE	MSE	R-square	RMSE	MSE	R -square
Random Forest	2.27	5.15	93.12	2.97	8.82	90.4
LSTM(Long Short Term Memory)	3.78	14.34	73.79	3.68	13.56	71.23
Facebook Prophet	2.99	8.94	68.14	3.08	9.4	66.23
XGBoost	2.87	8.24	92.18	3.007	9.04	89.4

Conclusion

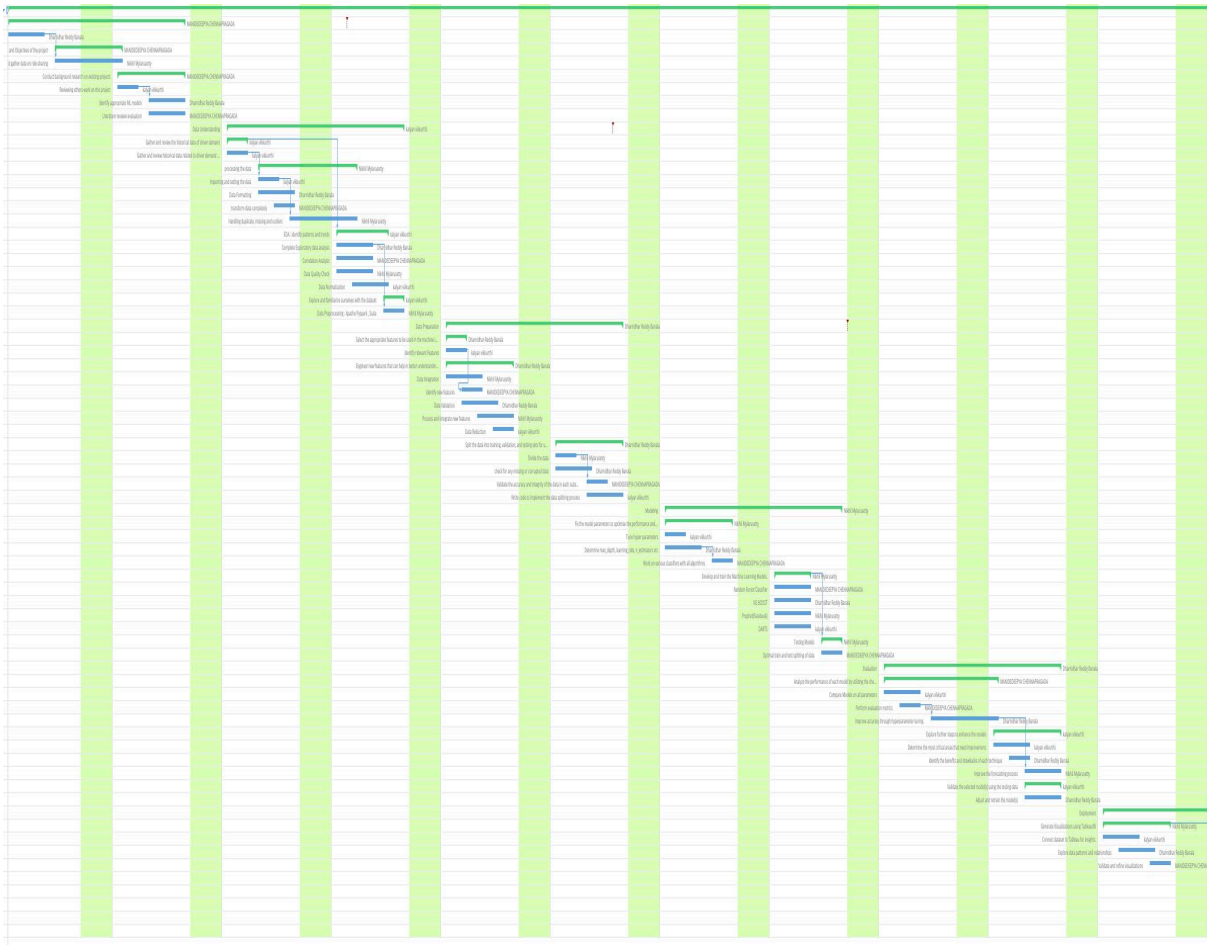
The R^2 values of the RF and XGBoost algorithms are greater than those of the Prophet and LSTM algorithms. This indicates that both the ensemble models are handling non-linear relationships in the data.The data is smoothed in order to model Prophet and LSTM because predictions with the original data are inaccurate. That being the reason the RMSE and MSE scores of Prophet and LSTM got improved and all the algorithms are having RMSE scores with not much of difference. However, the data is only considered for a 6-month period, so the efficacy of the models will change if more months and years of massive data are provided. Due to the short period of time, it is difficult for algorithms to identify patterns and trends. In terms of execution speed, Prophet is the quickest algorithm, followed by XGBoost, Random Forest, and LSTM.

Limitations and Future Scope:

The dataset size can be increased to get the better performance of all the algorithms. In the future, all forms of taxis will be able to be combined with yellow taxi data, allowing for more accurate data-driven predictions through the development of efficient algorithms. In the future this project have the scope of combining strengths of individual algorithms together by stacking techniques. Combinations of various algorithms can be used to create

these stacked models; this can enhance the model's ability by making it more robust to any type of data and improve predictions.

Appendix A



Appendix A. Complete gantt chart for the Driver Demand Prediction Project

Appendix B

Preprocessing Code

This appendix includes crucial steps for project implementation. Figure B1shows code snippets on removal process of unnecessary values in the data. Figure B2 shows the smoothing process performed on taxi data. Figure B3 shows the Dimensionality Reduction technique

```
# Remove trips with invalid pickup or dropoff locations
valid_locs = list(range(1, 266))
df = df[(df['PULocationID'].isin(valid_locs)) & (df['DOLocationID'].isin(valid_locs))]

# Remove trips with invalid passenger counts or distances
df= df[(df['passenger_count'] > 0) & (df['trip_distance'] > 0)]

# Remove trips with negative or zero fare amounts
df= df[df['fare_amount'] > 0]

# Remove trips with excessively long or short durations
df=df[(df['trip_duration'] >= 1) & (df['trip_duration'] <= 720)]

# Calculate the average speed of the trip in miles per hour
df['average_speed'] = df['trip_distance'] / (df['trip_duration'] / 60)

# Replace NULL values with NaN
df.replace('', np.nan, inplace=True)

# Check for NULL values
print(df.isnull().sum())
```

Figure B1. Code for removal of null values

```
# calculate the rolling mean using a window of size 7 days
rolling_mean = final_target['daily_pickups'].rolling(window=7).mean()

# create a new DataFrame with the smoothed data
df_smoothed = pd.DataFrame({'pickup_date': final_target['pickup_date'], 'daily_pickups': rolling_mean, 'PULocationID': 
df_smoothed.dropna(inplace=True)

# plot the original data and the smoothed data
ax = final_target.plot(x='pickup_date', y='daily_pickups', figsize=(10, 5), label='Original')
df_smoothed.plot(x='pickup_date', y='daily_pickups', label='Smoothed', ax=ax)
```

Figure B2. Code for smoothing

```python
from sklearn.decomposition import PCA

# Select the features you want to include in the PCA
features = ['PULocationID', 'Pickup_year', 'pickup_dayofweek', 'day', 'month']

# Standardize the data
from sklearn.preprocessing import StandardScaler
x = final_target.loc[:, features].values
x = StandardScaler().fit_transform(x)

# Perform PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data=principalComponents, columns=['principal component 1', 'principal component 2'])

final_target = final_target.reset_index(drop=True)

# Concatenate the principal components with the target variable and other features
df_pca = pd.concat([principalDf, final_target['daily_pickups']], axis=1)

# Display the explained variance ratio of each principal component
print(pca.explained_variance_ratio_)
```

Figure B3. Code for Dimensionality Reduction

Appendix C

Code for Implementing XGBoost Regressor

This Appendix describes the code that is implemented for all the four algorithms. Figure C1 shows the code implementation of XGBoost. Figure C2 shows the code implementation of Random Forest Regressor. Figure C3 shows the code implementation of Prophet. Figure C4 shows the code implementation of LSTM Algorithm.

```
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

# training X and y
X = final_target[['PULocationID','day', 'month','Pickup_year','pickup_dayofweek']]
y = final_target['daily_pickups']

# Split the data into training, validation, and testing sets
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, random_state=42)

# Perform hyperparameter tuning using grid search on the validation set
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 1.0],
    'subsample': [0.5, 0.75, 1.0],
    'colsample_bytree': [0.5, 0.75, 1.0],
    'gamma': [0, 0.1, 0.5]
}

xgb_model = xgb.XGBRegressor(random_state=42)
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Build an XGBoost model using the best hyperparameters on the combined training and validation set
best_xgb = grid_search.best_estimator_
best_xgb.fit(X_train_val, y_train_val)

# Evaluate the model on the validation set
y_val_pred = best_xgb.predict(X_val)
mse_val = mean_squared_error(y_val, y_val_pred)
rmse_val = np.sqrt(mse_val)
r2_val = r2_score(y_val, y_val_pred)
accuracy_val = r2_score(y_val, y_val_pred)

print("Validation Set Metrics:")
print("MSE:", mse_val)
print("RMSE:", rmse_val)
print("R-squared:", r2_val)
#print("Accuracy score:", accuracy_val)

# Evaluate the model on the testing set
y_test_pred = best_xgb.predict(X_test)
mse_test = mean_squared_error(y_test, y_test_pred)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test, y_test_pred)
accuracy_test = r2_score(y_test, y_test_pred)

print("\nTest Set Metrics:")
print("MSE:", mse_test)
print("RMSE:", rmse_test)
print("R-squared:", r2_test)
#print("Accuracy score:", accuracy_test)
```

Figure C1. Code Implementation of XGBoost.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

# training X and y
X_train = final_target[['PULocationID','day', 'month','Pickup_year','pickup_dayofweek']]
y_train = final_target['daily_pickups']

# Split the data into training and testing sets
X_temp, X_test, y_temp, y_test = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# Split the training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.2, random_state=42)

# Perform hyperparameter tuning using grid search
param_grid = {
    "n_estimators": [50, 100, 150],
    "max_depth": [5, 10, 15],
    "min_samples_split": [2, 4, 6],
    "min_samples_leaf": [1, 2, 4]
}

rf = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Build a random forest model using the best hyperparameters
best_rf = grid_search.best_estimator_

# Evaluate the model on the validation dataset
y_pred_val = best_rf.predict(X_val)
mse_val = mean_squared_error(y_val, y_pred_val)
rmse_val = np.sqrt(mse_val)
r2_val = r2_score(y_val, y_pred_val)
accuracy_val = r2_score(y_val, y_pred_val)
print("Validation MSE:", mse_val)
print("Validation RMSE:", rmse_val)
print("Validation R-squared:", r2_val)
print("Validation accuracy score:", accuracy_val)

# Evaluate the model on the testing dataset
y_pred_test = best_rf.predict(X_test)
mse_test = mean_squared_error(y_test, y_pred_test)
rmse_test = np.sqrt(mse_test)
r2_test = r2_score(y_test, y_pred_test)
accuracy_test = r2_score(y_test, y_pred_test)
print("Testing MSE:", mse_test)
print("Testing RMSE:", rmse_test)
print("Testing R-squared:", r2_test)
print("Testing accuracy score:", accuracy_test)
```

Figure C2. Code Implementation of Random Forest Regressor.

```
import pandas as pd
from prophet import Prophet

# Convert 'pickup_date' to 'ds' and 'daily_prophet_pickups' to 'y' for Prophet
X_prophet['pickup_date'] = pd.to_datetime(X_prophet['pickup_date'])
X_prophet = X_prophet.rename(columns={'pickup_date': 'ds'})
y_prophet = y_prophet.rename('y')

# Divide the data into training, testing, and validation sets
train_end_date = pd.to_datetime('2022-01-01') + pd.DateOffset(months=4)
test_end_date = train_end_date + pd.DateOffset(months=1)
validation_end_date = test_end_date + pd.DateOffset(months=1)

train_df = pd.concat([X_prophet[X_prophet['ds'] <= train_end_date], y_prophet[X_prophet['ds'] <= train_end_date]], axis
test_df = pd.concat([X_prophet[(X_prophet['ds'] > train_end_date) & (X_prophet['ds'] <= test_end_date)], y_prophet[(X_p
validation_df = pd.concat([X_prophet[X_prophet['ds'] > test_end_date], y_prophet[X_prophet['ds'] > test_end_date]], axi

# Create a Prophet model and fit the training data
model = Prophet(
    seasonality_mode='multiplicative',
    daily_seasonality=True,
    weekly_seasonality=True,
    yearly_seasonality=False
)
# Add additional seasonality components
model.add_seasonality(
    name='monthly',
    period=30.5,
    fourier_order=12
)
model.add_seasonality(
    name='daily',
    period=1,
    fourier_order=15
)

model.fit(train_df)
# Generate forecasts for the test and validation sets
future_test = model.make_future_dataframe(periods=len(test_df))
forecast_test = model.predict(future_test)

future_validation = model.make_future_dataframe(periods=len(validation_df))
forecast_validation = model.predict(future_validation)

# Evaluate the model's performance on the test and validation sets
y_prophet_test_predicted = forecast_test[-len(test_df):]['yhat']
y_prophet_validation_predicted = forecast_validation[-len(validation_df):]['yhat']
# Convert the predicted values to an array
y_prophet_test_predicted = np.array(y_prophet_test_predicted)
y_prophet_validation_predicted = np.array(y_prophet_validation_predicted)

# Get the actual values from the test and validation sets
y_prophet_test_actual = np.array(test_df['y'])
y_prophet_validation_actual = np.array(validation_df['y'])

# Calculate MAPE for test and validation sets
mape_test = mean_absolute_percentage_error(y_prophet_test_actual, y_prophet_test_predicted)
mape_validation = mean_absolute_percentage_error(y_prophet_validation_actual, y_prophet_validation_predicted)

# Calculate R2 score for test and validation sets
r2_test = r2_score(y_prophet_test_actual, y_prophet_test_predicted)
r2_validation = r2_score(y_prophet_validation_actual, y_prophet_validation_predicted)

# Calculate RMSE and MSE for test and validation sets
rmse_test = np.sqrt(mean_squared_error(y_prophet_test_actual, y_prophet_test_predicted))
rmse_validation = np.sqrt(mean_squared_error(y_prophet_validation_actual, y_prophet_validation_predicted))
mse_test = mean_squared_error(y_prophet_test_actual, y_prophet_test_predicted)
mse_validation = mean_squared_error(y_prophet_validation_actual, y_prophet_validation_predicted)

print("Test Set:")
print("MAPE:", mape_test)
print("R2 Score:", r2_test)
print("RMSE:", rmse_test)
print("MSE:", mse_test)
```

Figure C3. Code Implementation of Prophet.

```
# Perform train-validation-test split
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.18, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.23, random_state=42)

# Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
y_train_scaled = scaler.fit_transform(y_train.values.reshape(-1, 1))
y_val_scaled = scaler.transform(y_val.values.reshape(-1, 1))
y_test_scaled = scaler.transform(y_test.values.reshape(-1, 1))

# Reshape the input data to include a new dimension for timesteps
X_train_reshaped = np.reshape(X_train_scaled, (X_train_scaled.shape[0], X_train_scaled.shape[1], 1))
X_val_reshaped = np.reshape(X_val_scaled, (X_val_scaled.shape[0], X_val_scaled.shape[1], 1))
X_test_reshaped = np.reshape(X_test_scaled, (X_test_scaled.shape[0], X_test_scaled.shape[1], 1))

# Define the LSTM model
model = Sequential()
model.add(LSTM(units=50, activation='relu', return_sequences=True, input_shape=(X_train_reshaped.shape[1], X_train_resh
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')

# Train the model
model.fit(X_train_reshaped, y_train_scaled, epochs=100, batch_size=32, verbose=2)

# Make predictions on the training, validation, and test sets
y_train_pred_scaled = model.predict(X_train_reshaped)
y_val_pred_scaled = model.predict(X_val_reshaped)
y_test_pred_scaled = model.predict(X_test_reshaped)

# Rescale the predictions

y_train_rescaled = scaler.inverse_transform(y_train_scaled)
y_test_rescaled = scaler.inverse_transform(y_test_scaled)
y_train_pred = scaler.inverse_transform(y_train_pred_scaled.reshape(-1, 1))
y_val_pred = scaler.inverse_transform(y_val_pred_scaled.reshape(-1, 1))
y_test_pred = scaler.inverse_transform(y_test_pred_scaled.reshape(-1, 1))
y_train_rescaled = scaler.inverse_transform(y_train_scaled)
y_val_rescaled = scaler.inverse_transform(y_val_scaled)
y_test_rescaled = scaler.inverse_transform(y_test_scaled)

# Calculate the evaluation metrics for training set
mse_train = mean_squared_error(y_train_rescaled, y_train_pred)
rmse_train = np.sqrt(mse_train)
r2_train = r2_score(y_train_rescaled, y_train_pred)

# Calculate the evaluation metrics for validation set
mse_val = mean_squared_error(y_val_rescaled, y_val_pred)
rmse_val = np.sqrt(mse_val)
r2_val = r2_score(y_val_rescaled, y_val_pred)
```

Figure C4. Code Implementation of LSTM.

References

Anggraeni, F., Adytia, D., & Ramadhan, A. W. (2021). *Forecasting of Wave Height Time Series Using AdaBoost and XGBoost, Case Study in Pangandaran, Indonesia*. https://doi.org/10.1109/icodsa53588.2021.9617524

Askari, B., Quy, T. L., & Ntoutsi, E. (2020). *Taxi Demand Prediction using an LSTM-Based Deep Sequence Model and Points of Interest*. (2020, July 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/9202791

Boumeddane, S., Hamdad, L., Bouregag, A. A. E. F., Damene, M., & Sadeg, S. (2021). *A Model Stacking Approach for Ride-Hailing Demand Forecasting: a Case Study of Algiers*. (2021, February 9). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9378731

Breiman, L. (1984). Classification and Regression Trees. *Biometrics*, *40*(3), 874. https://doi.org/10.2307/2530946

Carson-Bell, D., Adadevoh-Beckley, M., & Kaitoo, K. (2021, February 25). *Demand Prediction of Ride-Hailing Pick-Up Location Using Ensemble Learning Methods*. Journal of Transportation Technologies; Scientific Research Publishing. https://doi.org/10.4236/jtts.2021.112016

Chen, T., & Guestrin, C. (2016). *XGBoost*. https://doi.org/10.1145/2939672.2939785

Chen, Z., Zhao, B., Wang, Y., Duan, Z., & Zhao, X. (2020). Multitask Learning and GCN-Based Taxi Demand Prediction for a Traffic Road Network. *Sensors*, *20*(13), 3776. https://doi.org/10.3390/s20133776

Connor, J. N. L., Martin, R. M., & Atlas, L. (1994). Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, *5*(2), 240–254. https://doi.org/10.1109/72.279188

Genuer, R. (2012). Variance reduction in purely random forests. *Journal of Nonparametric Statistics*, *24*(3), 543–562. https://doi.org/10.1080/10485252.2012.677843

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. "O'Reilly Media, Inc."

Gupta, B., Awasthi, S., Gupta, R., Ram, L., Kumar, P., Prasad, B. R., & Agarwal, S. (2018a, January 1). *Taxi Travel Time Prediction Using Ensemble-Based Random Forest and Gradient Boosting Model*. Advances in Intelligent Systems and Computing; Springer Nature. https://doi.org/10.1007/978-981-10-7200-0_6

Hastie, T., Tibshirani, R., & Friedman, J. (2013). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Jamil, M., & Akbar, S. (2017, October 1). *Taxi passenger hotspot prediction using automatic ARIMA model*. International Conference on Science in Information Technology. https://doi.org/10.1109/icsitech.2017.8257080

Kankanamge, K. D., Witharanage, Y. R., Withanage, C. S., Hansini, M., Lakmal, D., & Thayasiva, U. (2019). *Taxi Trip Travel Time Prediction with Isolated XGBoost Regression*. IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8818915

Li, Y., Wang, Y., Berecibar, M., Nanini-Maury, E., Chan, J. C., Van Den Bossche, P., Van Mierlo, J., & Omar, N. (2018). Random forest regression for online capacity estimation of lithium-ion batteries. *Applied Energy*, *232*, 197–210. https://doi.org/10.1016/j.apenergy.2018.09.182

Liu, Y., Chen, H., Zhang, L., & Feng, Z. (2021). Enhancing building energy efficiency using a random forest model: A hybrid prediction approach. *Energy Reports*, *7*, 5003–5012. https://doi.org/10.1016/j.egyr.2021.07.135

Liu, Z., Chen, H., Li, Y., & Zhang, Q. (2020). Taxi Demand Prediction Based on a Combination Forecasting Model in Hotspots. *Journal of Advanced Transportation*, *2020*, 1–13. https://doi.org/10.1155/2020/1302586

Liu, Z., Sun, X., & Chen, H. (2020). *Data-Driven Real-Time Online Taxi-Hailing Demand - ProQuest*. (n.d.). https://www.proquest.com/docview/2533959024?pq-origsite=gscholar&fromopenview=true

Makridakis, S., Spiliotis, E., Assimakopoulos, V., Semenoglou, A., Mulder, G., & Nikolopoulos, K. (2022b). Statistical, machine learning and deep learning forecasting methods: Comparisons and ways forward. *Journal of the Operational Research Society*, 1–20. https://doi.org/10.1080/01605682.2022.2118629

Naji, H. a. H., Xue, Q., Zhu, H., & Li, T. (2021). Forecasting Taxi Demands Using Generative Adversarial Networks with Multi-Source Data. *Applied Sciences*, *11*(20), 9675. https://doi.org/10.3390/app11209675

Noorunnahar, M., Chowdhury, A. H., & Mila, F. A. (2023). A tree based eXtreme Gradient Boosting (XGBoost) machine learning model to forecast the annual rice production in Bangladesh. *PLOS ONE*, *18*(3), e0283452. https://doi.org/10.1371/journal.pone.0283452

Poongodi, M., Malviya, M., Kumar, C., Hamdi, M., V., Nebhen, J., & Alyamani, H. J. (2021). New York City taxi trip duration prediction using MLP and XGBoost. *International Journal of Systems Assurance Engineering and Management*, *13*(S1), 16–27. https://doi.org/10.1007/s13198-021-01130-x

Rayle, L., Dai, D., Chan, N., Cervero, R., & Shaheen, S. (2016, January 1). *Just a better taxi? A survey-based comparison of taxis, transit, and ridesourcing services in San Francisco*. Transport Policy; Elsevier BV. https://doi.org/10.1016/j.tranpol.2015.10.004

Saiktishna, C., Sumanth, N., Rao, M. M. S., & Thangakumar, J. (2022). Historical Analysis and Time Series Forecasting of Stock Market using FB Prophet. In *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*. https://doi.org/10.1109/iciccs53718.2022.9788231

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Silveira-Santos, T., González, A. B. R., Rangel, T., Pozo, R. F., Vassallo, J. M., & Díaz, J. J. V. (2022). Were ride-hailing fares affected by the COVID-19 pandemic? Empirical analyses in Atlanta and Boston. *Transportation*. https://doi.org/10.1007/s11116-022-10349-x

Stefenon, S. F., Seman, L. O., Mariani, V. C., & Coelho, L. D. S. (2023, January 29). *Aggregating Prophet and Seasonal Trend Decomposition for Time Series Forecasting of Italian Electricity Spot Prices*. Energies; MDPI. https://doi.org/10.3390/en16031371

Taylor, S. J., & Letham, B. (2017). *Forecasting at Scale*. https://www.tandfonline.com/doi/abs/10.1080/00031305.2017.1380080

Taylor, S. V., & Letham, B. (2018). Forecasting at Scale. *The American Statistician*, *72*(1), 37–45. https://doi.org/10.1080/00031305.2017.1380080

Vanichrujee, U., Horanont, T., Pattara-atikom, W., Theeramunkong, T., & Shinozaki, T. (2020). *Taxi Demand Prediction using Ensemble Model Based on RNNs and*

XGBOOST. (2018, May 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/8442063

Vitanov, N. K., Hoffmann, N., & Wernitz, B. (2014). Nonlinear time series analysis of vibration data from a friction brake: SSA, PCA, and MFDFA. *Chaos Solitons & Fractals*, *69*, 90–99. https://doi.org/10.1016/j.chaos.2014.09.010

Wang, Y., & Mi, X. (2018). *A Comparative Study on Demand Forecast of Car Sharing Users Based on ARIMA and LSTM*. (2020, May 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/9237552?casa_token=LpxKbFJoVBQAAAAA:HRPyIzepgOZxLZjFzsNwq9w4EOEhgu78eB0i5dZ4X0Ad2mKC1IrbNfHT2ditdMFqJFEhIohf

Wu, Z., & Lian, G. (2019). *A novel dynamically adjusted regressor chain for taxi demand prediction.* (2020b, July 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9207160

Xu, J., Rahmatizadeh, R., Boloni, L., & Turgut, D. (2017). *Real-Time Prediction of Taxi Demand Using Recurrent Neural Networks*. (2018, August 1). IEEE Journals & Magazine IEEE Xplore. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8082792

Xu, T. (2022). *Demand Analysis of Taxi Passenger-carrying Hot Spot Areas Based on XGBoost Algorithm*. (2022, May 27). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/9820001

Yang, H., Zheng, H., & Chen, X. (2017). Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach. *Transportation Research Part C-Emerging Technologies*, 85, 591–608. https://doi.org/10.1016/j.trc.2017.10.016

Yang, S., Ma, W., Pi, X., & Qian, Z. (2019). A deep learning approach to real-time parking occupancy prediction in transportation networks incorporating multiple spatio-temporal data sources. *Transportation Research Part C-emerging Technologies*, *107*, 248–265. https://doi.org/10.1016/j.trc.2019.08.010

Zhao, J., Chen, C., Huang, H., & Xiang, C. (2020b). Unifying Uber and taxi data via deep models for taxi passenger demand prediction. *Personal and Ubiquitous Computing*. https://doi.org/10.1007/s00779-020-01426-y

Zhu, X., Zhang, F., Deng, M., Liu, J., He, Z., Zhang, W., & Gu, X. (2022). A Hybrid Machine Learning Model Coupling Double Exponential Smoothing and ELM to Predict Multi-Factor Landslide Displacement. *Remote Sensing*, *14*(14), 3384. https://doi.org/10.3390/rs14143384

Zunic, E., Korjenic, K., Delalic, S., & Subara, Z. (2021). Comparison Analysis of Facebook's Prophet, Amazon's DeepAR+ and CNN-QR Algorithms for Successful Real-World Sales Forecasting. *International Journal of Computer Science and Information Technology*, *13*(2), 67–84. https://doi.org/10.5121/ijcsit.2021.13205