iJRASET
International Journal For Research in Applied Science and Engineering Technology



# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 11 **Issue:** V **Month of publication:** May 2023

**DOI:** https://doi.org/10.22214/ijraset.2023.51770

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 11 Issue V May 2023- Available at www.ijraset.com*

# Comparative Analysis of Regression Models for Price Prediction of Ride-On-Demand Services

Pooja Pranavi Nalamothu
*Computer Science Engineering, Jawaharlal Nehru Technological University, India*

***Abstract:*** ***In recent years, Ride-on-Demand (RoD) services such as Uber, Ola, and Rapido have emerged as popular alternatives to traditional taxi/cab services.***
***These services operate 24/7 and cater to tens of thousands of customers. Unlike traditional cabs, RoD services do not offer a fixed price. Instead, they utilize Dynamic Pricing to balance supply and demand, taking into account factors such as location, time of booking, ride demand, and driver availability to improve their service.***
***However, the unpredictable and fluctuating nature of dynamic pricing has posed a significant challenge for customers, leading them to pay higher fares without their knowledge. To address this issue, it is crucial to estimate dynamic prices accurately and provide the lowest possible fares to customers. We leverage datasets of Uber and Lyft, to compare and forecast the services that offer the most competitive pricing.***
***We also evaluate the contribution of different features to dynamic pricing, determining which factors play the most significant role in determining fare prices.***
***By analyzing the relationship between demand pricing and relevant features extracted from the datasets, we are able to carry out price prediction. To accomplish our goal of reducing transportation fares and waiting times while enhancing transport accessibility, we utilize three different machine learning models: K-Nearest Neighbors, SVM and Random Forest and test its output based on real service data from various perspectives. By comparing these models, we identify the best approach for predicting dynamic pricing and generating accurate forecasts for each individual order.***
***Keywords: Machine Learning, Regression, Demand forecasting, Price Prediction, Support Vector Machine, Random Forest, k-Nearest Neighbors, Correlation***

## I. INTRODUCTION

The taxi industry plays a crucial role in passenger transport, providing fast and convenient travel services to people. However, the imbalance of taxi supply(the number of cars on the road) and demand(the number of requests from passengers) is a significant issue in many cities due to the gaps in taxi quantities and demand distributions. The emergence of online ride-hailing services and the unbalanced spatiotemporal distribution of passengers further complicates the matter, making it difficult to reveal the complicated dependencies among different regions and temporal periods. The demand for an area is usually influenced by its surrounding neighbors and correlated with various historical observations, which makes it difficult to efficiently dispatch vehicles and keep operating costs low in on-demand mobility services.

To forecast RoD prices, various algorithms can be used, such as decision trees, support vector machines, and k-nearest neighbors, which help reduce the risk of relying on a single method that may not be effective in all situations. However, selecting the most appropriate algorithm depends on the specific needs of the forecasting task. For example, linear regression is often used for forecasting continuous variables, while decision trees are well-suited for predicting categorical variables.

The objective of this project is to identify the best low-cost RoD services for customers by finding the relations and patterns between attributes that affect dynamic prices and identifying the main dependent attributes of prices. Our aim is to build efficient Machine learning models and select the best among those in the proposed system. The efficiency of the model and errors in predictions will be calculated to ensure accurate results.

The rest of the paper is organized into the following sections. Section 2 discusses the already existing methodology. In section 3 we review the work related to the proposed approach and the techniques used therein. Section 4 explains the proposed approach in detail. Section 5 shows us the implementation and analyzes the existing model. Section 6 presents the results and discussions for various machine learning algorithms. Section 7 presents the summary of the proposed approach.

©IJRASET: All Rights are Reserved 

## II. EXISTING METHOD

Over the past years, the approach to predicting dynamic prices of ride-on-demand (RoD) services has been limited to the use of linear regression and statistical models. While these methods have provided some level of accuracy, they are restricted in their ability to only consider a limited number of attributes for prediction. With the increasing volume and complexity of data, these traditional approaches fall short in their capability to operate in higher dimensional data.

Furthermore, these conventional methods have been known to overlook the importance of certain features in the calculation of the Machine Learning model's accuracy. This can lead to less accurate predictions and undermine the effectiveness of the prediction models. Additionally, the traditional approach to predicting RoD prices has proven to be highly limited in its scope, as it fails to account for all of the relevant factors that may affect the prices of ride-on-demand services. This limitation can result in an incomplete understanding of the factors that influence the prices of these services, which in turn can lead to less accurate predictions. Another major drawback of traditional approaches in predicting RoD prices is their low performance with large datasets. With the increasing volume of data, these approaches may take longer processing times and result in less accurate predictions. Given these limitations, it is essential to explore new and more advanced approaches to predicting RoD prices. Using machine learning algorithms that take into account a wide range of attributes and factors can provide a more comprehensive analysis of the data and potentially lead to more accurate predictions. By staying up-to-date with evolving technologies and exploring new methods for predicting RoD prices, we can ensure that we are providing customers with the most accurate and up-to-date information possible.

## III. RELATED WORK

### *A. Price Prediction on Multi-source Urban Data*

This research paper focuses on the prediction of dynamic prices in ride-on-demand (RoD) services such as Uber and Lynk. The goal of this project is to help passengers make informed decisions about whether to take a ride by providing them with more information about the prices they can expect to pay. To achieve this, the researchers implemented a neural network based on features extracted from multi-source urban data. This neural network is designed to perform the prediction of RoD prices for any passenger request in any location within the city of Beijing. To train their neural network model, the researchers used features with more than 130 dimensions. These features were chosen based on their potential to influence RoD prices. The results of the model showed that it performed much better than a baseline predictor, as measured by the sMAPE metric.[4] The sMAPE metric provides insights into the contribution of each feature and the relationship between different transportation services. According to the results of the model, the availability of taxis and the weather conditions were found to have the strongest influence on RoD prices. In contrast, the distribution of public transportation services such as buses and the metro had a more negligible impact. Overall, this research demonstrates the potential of using a neural network and multi-source urban data to predict dynamic prices in RoD services helps provide passengers with more information to make informed decisions about whether to take a ride.

### *B. A Linear Regression-based Price Prediction*

The research paper described in this paragraph focuses on using data analysis and machine learning techniques to forecast ride-on-demand (RoD) prices. The primary focus of the paper is on exploratory data analysis (EDA), which involves examining the characteristics of the data and identifying patterns and trends. As part of this process, the researchers calculated various metrics to help understand the data and inform the development of their model. The researchers then developed a linear regression model to predict RoD prices based on the independent variables of ’travel distance’ and ’travel time’. Linear regression is a statistical method that is commonly used for forecasting continuous variables, such as prices. The results of the model showed that it performed well, but there was a limitation in the dataset size. As the dataset size increased, the performance of the model gradually decreased. Overall, the research paper presents a detailed analysis of the factors that influence RoD prices and demonstrates the effectiveness of using a linear regression model for forecasting. However, the researchers also note the limitation of the dataset size, which may affect the accuracy of the model in certain situations.[2]

### *C. Predicting real-time surge pricing of ride-sourcing companies*

In this paper, The Data describing the current state of the several urban systems are collected from web APIs and used as features of a log-linear model to predict surge multipliers in Pittsburgh. L1 regularization is used to allow a small number of important features to be selected in a data-driven way from the large number of spatio-temporal features describing the urban state. To allow the linear model to describe non-linear behavior, temporal segmentation and clustering are employed.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Days are segmented into time-of-day windows and separate models are trained for each. Clustering extracts temporal modes from the data and distance to the centroids of these clusters are included as features in the model. The model they developed have found that it performs best to predict Uber surge multipliers in urban areas. For both Uber and Lyft the model tends to under-perform na¨ıve methods for surge multipliers between 1.1 and 1.5 and outperform na¨ıve methods for larger values. This effect is particularly pronounced in the Lyft model, whose poor performance on low surge multipliers substantially degrades its overall performance.[3]

*D. On-demand High-capacity Ride-sharing*

In this paper, they explained a reactive anytime optimal method with scalable real time performance for assigning passenger requests to a fleet of vehicles of varying capacity. They quantify experimentally the tradeoff between fleet size, capacity, waiting time, travel delay, and operational costs for low and medium capacity vehicles, such as taxis in a large-scale city dataset. Under the assumption of one person per ride, they have shown that 98 percent of the taxi rides currently served by over 13,000 taxis could be served with just 3,000 taxis of capacity four.[5] They have observed that a vehicle capacity of two is sufficient for ride-sharing when a small trip delay of 2 min is imposed. If a maximum delay of 5 min or more (comparable to the time spent retrieving a car from parking) is allowed, higher-capacity vehicles (i) increase the service rate significantly,(ii) reduce the waiting time, and (iii) reduce the distance traveled by each vehicle. Our analysis shows that a ride-pooling service can provide a substantial improvement in urban transportation systems and that the system parameters such as vehicle capacity and fleet size depend on quality of service requirements and demand.[1]

## IV. PROPOSED METHODOLOGY

Our primary objective is to enhance user experience and overcome the limitations of current systems. To achieve this, we will be using three Machine Learning algorithms: Random Forest, Support Vector Machine and K-Nearest Neighbors to predict the prices of RoD services. These algorithms possess two critical features, namely the ability to learn an arbitrary function and the ability to control the learning process. They are superior to Linear Regression since they can handle datasets with higher dimensions and provide more efficient outcomes. Rather than selecting fewer ineffective features, utilizing better Machine Learning models would be a more prudent decision. Our ultimate aim is to predict prices more accurately and provide customers with low-cost RoD services.

*A. Workflow*

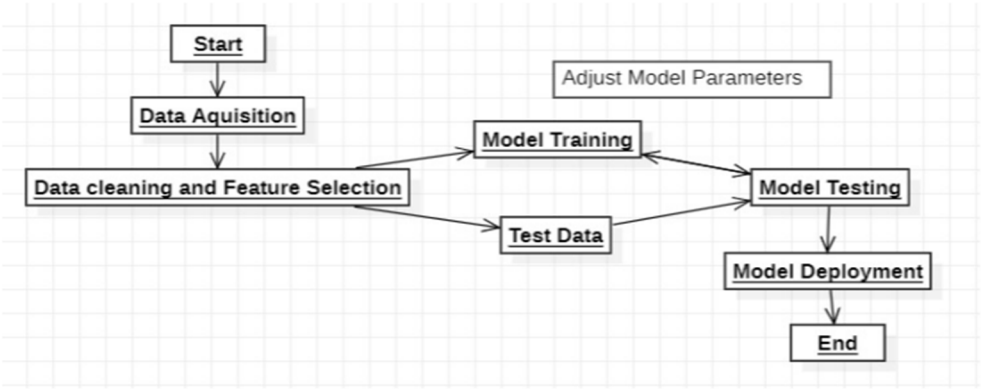



Fig 1. Workflow Architecture

The various phases shown in figure-1 are described below:

1) *Data Acquisition* - This initial step involves acquiring data, which can come from various sources such as downloading data files from the internet, collecting real-world data through sensors or surveys, or obtaining data from industries or organizations.
2) *Data Cleaning* - It is the process of identifying and rectifying errors, inconsistencies and inaccuracies in data prior to analysis. This could include removing irrelevant or duplicate data, correcting incorrectly formatted data or filling in missing values. The goal of data cleaning is to ensure the accuracy and completeness of data for further processing and analysis.
3) *Feature Selection* - This is the process of choosing important and relevant features from a given set of input data. The main goal of feature selection is to improve the performance of the machine learning algorithms.

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 11 Issue V May 2023- Available at www.ijraset.com*

4) *Test data* - This stage can also be called data splitting. There are two common approaches in which data split can take place. One approach is train- test split. Another approach is train- validation- test split. We will be using the first approach. Typically, the dataset is split into two sets, the training set and the test set, with the training set(such as 80% of the original data) being used to fit the model and the test set(such as 20% of the original data) being used to evaluate the performance of the model on unseen data. Figure 2 shows the pictorial representation of Train -Test split.

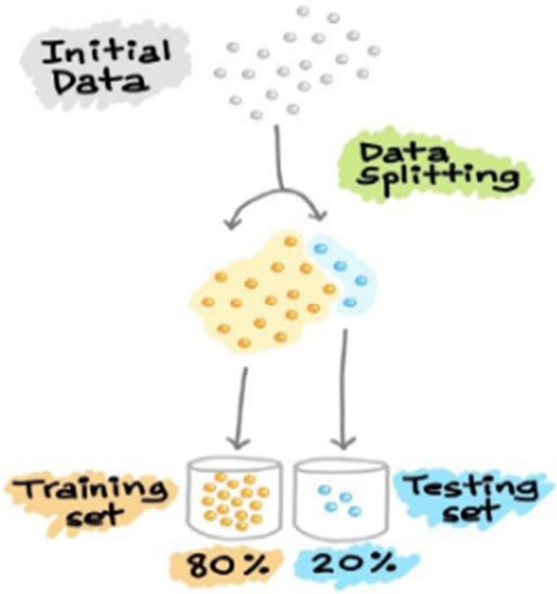



Fig. 2 Pictorial representation of train- test split

A predictive model is developed using the training set and such a trained model is then applied to the test set to make predictions.

5) *Model Training* - At this stage, machine learning algorithms are employed based on the type of target variable (continuous, categorical, etc.) and other relevant factors of the analytical problem. One or more machine learning algorithms are selected to create models for the same problem. Once the model is built, it is trained using the training dataset.
6) *Model Testing* - At this stage, the trained model is assessed by detecting prediction errors. The discrepancies between the actual value and the predicted value (derived from the trained model) of the target variable are computed using the testing dataset. Figure 3 displays the formulas for calculating different types of errors. Additionally, the formula for calculating the R-squared value of the model is depicted in Figure 4. The R-squared value ranges from 0 to 1, with a model being considered good if the R-squared value exceeds 0.8.

```
In [ ]: #mean_absolute_error= 1/n* summation of(actual value- predicted value)
        #mean_squared_error= 1/n* summation of(actual value- predicted value)**2
```

Fig. 3 Formulae for MAE and MSE errors

```
In [ ]: #R**2= 1- (SSR/SST)
        #where SSR= summation ((actual value- predicted value)**2)
        #SST= summation ((actual value- mean of actual values)**2)
```

Fig. 4 Formula to calculate R- squared value of a model

7) Model Deployment - Lastly, the evaluated model is deployed to make predictions of the target or expected variable in advance.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

*B. Algorithms used*

*1) K- Nearest Neighbors Algorithm*

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It can be used for both regression and classification problems. Figure 5 provides a visual representation of the K-Nearest Neighbor (KNN) algorithm.

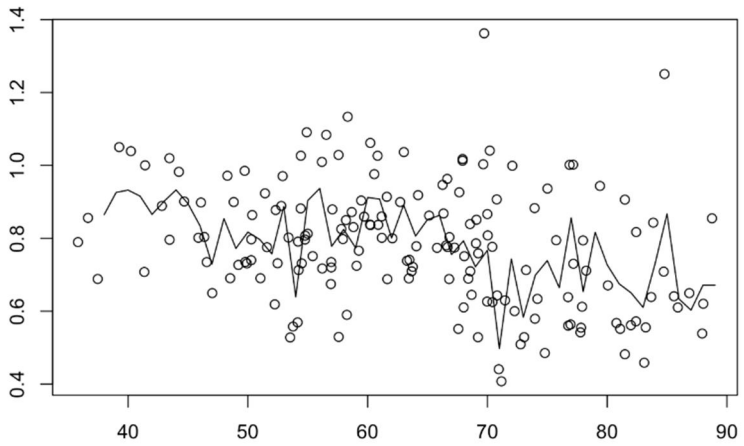



Fig. 5 k-nearest neighbor algorithm

*2) Random Forest Algorithm*

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. Figure-6 gives a pictorial representation of random forest algorithm.

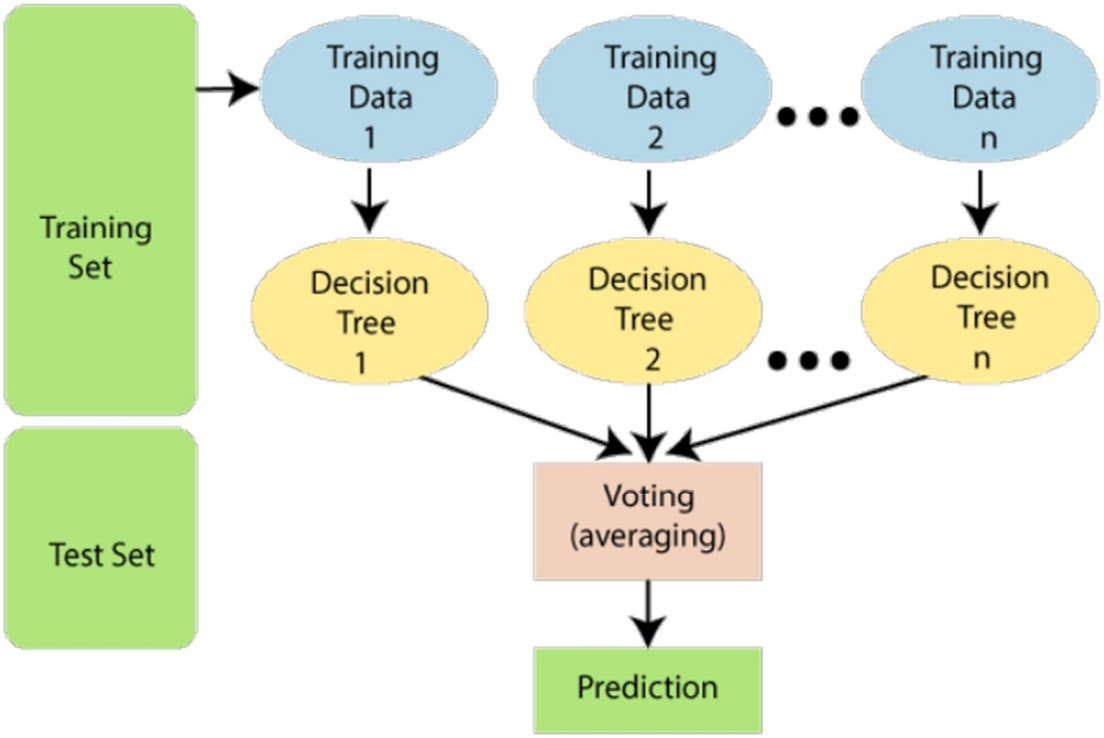



Fig. 6 Random Forest Algorithm

*3) Support Vector Machine Algorithm*

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. The goal of the SVM algorithm is to improve the accuracy of predictions by identifying the most relevant variables and finding the best decision boundary to separate the data points.This best decision boundary is called a hyperplane. Figure-7 gives pictorial representation of Support Vector Machine.

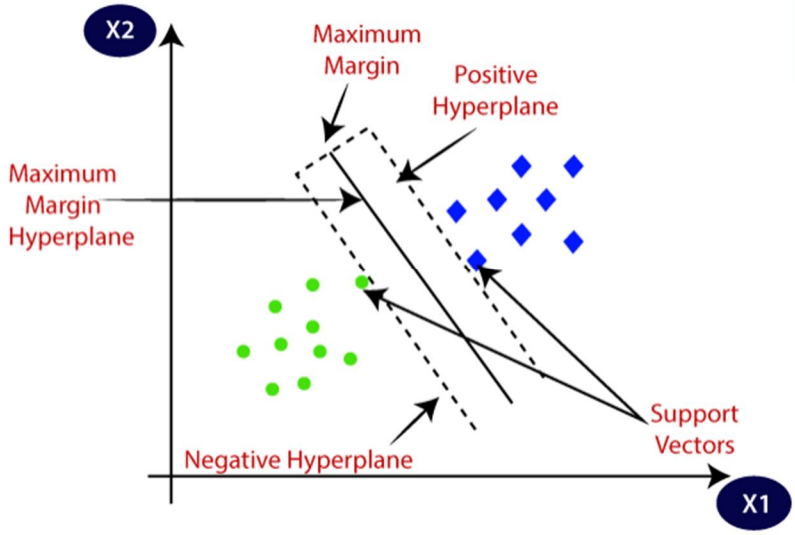



Fig. 7 Support Vector Machine Algorithm

*C. Software Specifications*

Tools used - Jupyter Notebook

Packages used - numpy, seaborn, pandas,scikit-learn, matplotlib.

## V. IMPLEMENTATION AND ANALYSIS

The dataset is initially in the CSV (Comma Separated Value) format. This dataset has been loaded into a pandas library dataframe in Jupyter notebooks. Figure 8 displays a sample of this dataset, including a unique identifier for the customer as 'id', the timestamp of the trip in Epoch format as 'timestamp', the pick-up location as 'source', the drop location as 'destination', the distance between the source and destination as 'distance', the surge multiplier by which the price was increased (default 1) as 'surge_multiplier', and the price estimate for the ride in USD as 'price'. In this case, 'price' is the target variable.

| | distance | cab_type | time_stamp | destination | source | price | surge_multiplier | id | product_id | name |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.44 | Lyft | 1544952607890 | North Station | Haymarket Square | 5.0 | 1.0 | 424553bb-7174-41ea-aeb4-fe06d4f4b9d7 | lyft_line | Shared |
| **1** | 0.44 | Lyft | 1543284023677 | North Station | Haymarket Square | 11.0 | 1.0 | 4bd23055-6827-41c6-b23b-3c491f24e74d | lyft_premier | Lux |
| **2** | 0.44 | Lyft | 1543366822198 | North Station | Haymarket Square | 7.0 | 1.0 | 981a3613-77af-4620-a42a-0c0866077d1e | lyft | Lyft |
| **3** | 0.44 | Lyft | 1543553582749 | North Station | Haymarket Square | 26.0 | 1.0 | c2d88af2-d278-4bfd-a8d0-29ca77cc5512 | lyft_luxsuv | Lux Black XL |
| **4** | 0.44 | Lyft | 1543463360223 | North Station | Haymarket Square | 9.0 | 1.0 | e0126e1f-8ca9-4f2e-82b3-50505a09db9a | lyft_plus | Lyft XL |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **693066** | 1.00 | Uber | 1543708385534 | North End | West End | 13.0 | 1.0 | 616d3611-1820-450a-9845-a9ff304a4842 | 6f72dfc5-27f1-42e8-84db-ccc7a75f6969 | UberXL |
| **693067** | 1.00 | Uber | 1543708385534 | North End | West End | 9.5 | 1.0 | 633a3fc3-1f86-4b9e-9d48-2b7132112341 | 55c66225-fbe7-4fd5-9072-eab1ece5e23e | UberX |
| **693068** | 1.00 | Uber | 1543708385534 | North End | West End | NaN | 1.0 | 64d451d0-639f-47a4-9b7c-6fd92fbd264f | 8cf7e821-f0d3-49c6-8eba-e679c0ebcf6a | Taxi |
| **693069** | 1.00 | Uber | 1543708385534 | North End | West End | 27.0 | 1.0 | 727e5f07-a96b-4ad1-a2c7-9abc3ad55b4e | 6d318bcc-22a3-4af6-bddd-b409bfce1546 | Black SUV |
| **693070** | 1.00 | Uber | 1543708385534 | North End | West End | 10.0 | 1.0 | e7fdc087-fe86-40a5-a3c3-3b2a8badcbda | 997acbb5-e102-41e1-b155-9df7de0a73f2 | UberPool |

693071 rows × 10 columns

Fig. 8 Initial rides Dataset

We check for null values in the initial rides dataset and then drop all the null values in this dataset row-wise.

We then take a second dataset in the CSV (Comma Separated Value) format. This dataset has been loaded into a pandas library dataframe in Jupyter notebooks. This dataset consists of weather data for all the locations taken into consideration in the initial rides dataset. Collected every hour. Figure 9 displays a sample of weather dataset wherein 'temp' is the temperature in fahrenheit F, 'location' is the name of particular location, 'clouds', 'pressure' is measured in mm, 'rain' is measured in inches ,'time_stamp' is the epoch time when row data was collected, 'humidity' is given as %, 'wind' is the wind speed in miles per hour.

| | temp | location | clouds | pressure | rain | time_stamp | humidity | wind |
|---|---|---|---|---|---|---|---|---|
| **0** | 42.42 | Back Bay | 1.00 | 1012.14 | 0.1228 | 1545003901 | 0.77 | 11.25 |
| **1** | 42.43 | Beacon Hill | 1.00 | 1012.15 | 0.1846 | 1545003901 | 0.76 | 11.32 |
| **2** | 42.50 | Boston University | 1.00 | 1012.15 | 0.1089 | 1545003901 | 0.76 | 11.07 |
| **3** | 42.11 | Fenway | 1.00 | 1012.13 | 0.0969 | 1545003901 | 0.77 | 11.09 |
| **4** | 43.13 | Financial District | 1.00 | 1012.14 | 0.1786 | 1545003901 | 0.75 | 11.49 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **6271** | 44.72 | North Station | 0.89 | 1000.69 | NaN | 1543819974 | 0.96 | 1.52 |
| **6272** | 44.85 | Northeastern University | 0.88 | 1000.71 | NaN | 1543819974 | 0.96 | 1.54 |
| **6273** | 44.82 | South Station | 0.89 | 1000.70 | NaN | 1543819974 | 0.96 | 1.54 |
| **6274** | 44.78 | Theatre District | 0.89 | 1000.70 | NaN | 1543819974 | 0.96 | 1.54 |
| **6275** | 44.69 | West End | 0.89 | 1000.70 | NaN | 1543819974 | 0.96 | 1.52 |

6276 rows × 8 columns

Fig. 9 Weather dataset

We again check for null values in the weather dataset and fill these null values or the missing values with 0. By filling the null values with 0, we ensure that the weather data is complete and consistent for further analysis or modelling. After filling these null values, the weather dataset is grouped by the 'location' column. Then the mean of each numeric column is computed for each group. We then pass an argument to ensure that the 'location' column becomes a regular column instead of being used as the index of the dataset. Next, the 'time_stamp' column is dropped.'

The resulting dataset contains the mean values of the numeric columns in the initial weather dataframe grouped by location. This can be useful for further analysis or modeling tasks where the mean values of weather parameters for different locations are required.

Finally, Merging both initial rides dataset and weather dataset we get the final dataset.

We check for any null or missing values in this final dataset and the results can be shown in Figure - 10.

```
distance                0
cab_type                0
time_stamp              0
destination             0
source                  0
price                   0
surge_multiplier        0
id                      0
product_id              0
name                    0
source_temp             0
source_clouds           0
source_pressure         0
source_rain             0
source_humidity         0
source_wind             0
destination_temp        0
destination_clouds      0
destination_pressure    0
destination_rain        0
destination_humidity    0
destination_wind        0
dtype: int64
```

Fig. 10

It is evident from the dataset that there are no missing values present, thus there is no requirement for further data cleaning for the purpose of model building. Furthermore, there are no categorical columns of object type in the dataset.
Figure - 11 given below gives the correlation with the target variable.

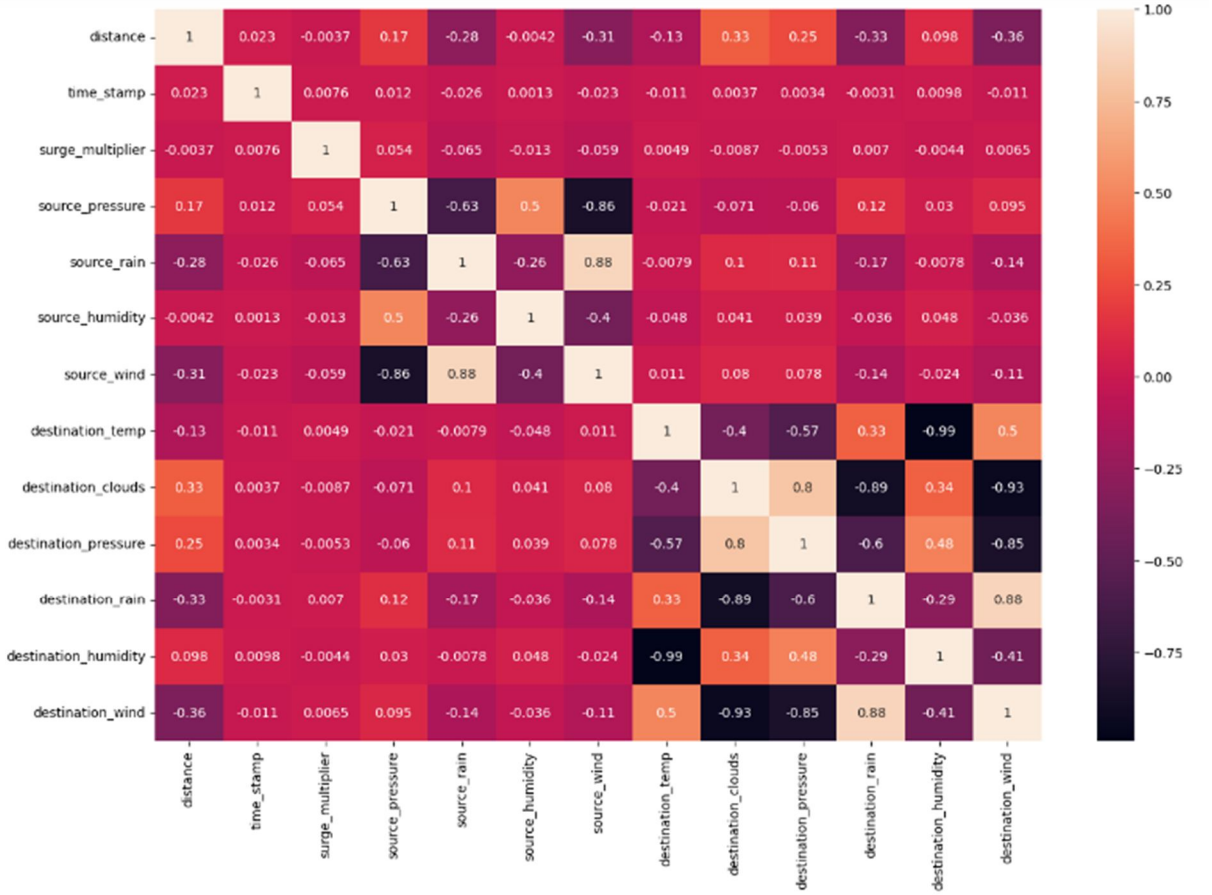



Fig. 11 Heatmap visualizing correlation among variables

To visually represent our predictions we use scatter plots. The plot showcases the predicted values for the target variable based on the values of the input features. The scatter plot indicates the relationship between the predicted values and the actual values for the test data. Each point on the scatter plot represents a single observation. The closer a point is to the diagonal line, the better the prediction accuracy of the Machine learning algorithm. The diagonal line represents the ideal scenario where predicted values perfectly match the actual values. This visualization helps to understand the overall performance of the algorithm in predicting the target variable.
Before we split our data into training data and test data, we create dummy variables with the help of one hot encoding.
The reason we create dummy variables with the help of one hot encoding is to represent categorical variables as numerical variables, which can be used as input features for machine learning models. In many machine learning algorithms, mathematical equations are used to make predictions, and these equations require numerical values as input. However, categorical variables like color, gender, or location cannot be used as numerical input directly because they are not numeric in nature. One hot encoding creates a separate binary column for each category of a categorical variable. In each binary column, a value of 1 is assigned for the presence of the category and a value of 0 for the absence of the category. This allows the categorical variable to be converted into a set of numerical variables, which can be used as input for machine learning models. By creating dummy variables with one hot encoding, we ensure that the machine learning algorithm is able to process all the features of the dataset, including the categorical variables, thereby improving the accuracy and effectiveness of the model.
Now we split our dataset into x and y, where y is the entire “price” column and x is the entire final dataset without the “price” column. We then split our data into train and test.

The next phase is the model training. Before we build our three proposed models, we calculate the accuracy of the existing model, i.e Linear Regression model.

Figure 12 shows the scatter plot of true values v/s predicted values (predicted by built model) of price. It is clear that the plot of true and predicted values comes out to be almost linear, which is a good sign.

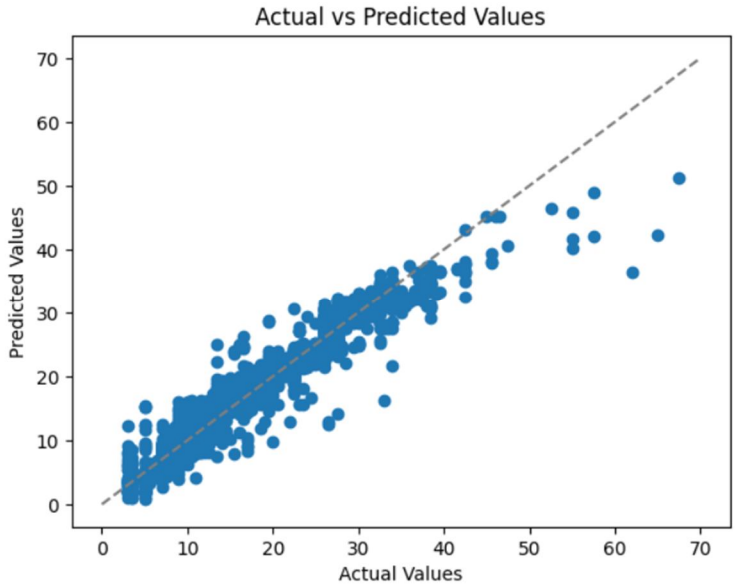



Fig. 12 Scatter plot of actual vs predicted prices

The R- squared value for the model is shown in Figure 13. It is clear that the performance of our linear regression model comes out to be 92.92%.

```
In [38]: #R**2= 1- (SSR/SST)
         #where SSR= summation ((actual value- predicted value)**2)
         #SST= summation ((actual value- mean of actual values)**2)
         print("Test R^2 Score: {:.5f}".format(model.score(X_test, y_test)))

         Test R^2 Score: 0.92923
```

Fig. 13 R-squared value of linear regression model

Figure 14 shows various kinds of errors calculated for the linear regression model we built.

```
In [36]: #mean_absolute_error= 1/n* summation of(actual value- predicted value)
         #mean_squared_error= 1/n* summation of(actual value- predicted value)**2
         #Root_mean_squared_error= squareroot(1/n* summation of(actual value- predicted value)**2)
         print('Mean Absolute Error(MAE):', metrics.mean_absolute_error(y_test, y_pred_lr))
         print('Mean Squared Error(MSE):', metrics.mean_squared_error(y_test, y_pred_lr))
         print('Root Mean Squared Error(RMSE):', np.sqrt(metrics.mean_squared_error(y_test, y_pred_lr)))

         # Calculate the absolute errors
         errors = abs(y_pred_lr - y_test)
         # Print out the mean absolute error (mae)
         print('Mean Absolute Error(MAE):', round(np.mean(errors), 2), 'degrees.')

         # Calculate mean absolute percentage error (MAPE)
         mape = 100 * (errors / y_test)
         # Calculate and display accuracy
         accuracy = 100 - np.mean(mape)
         print('Accuracy of linear regression model:', round(accuracy, 2), '%.')

         Mean Absolute Error(MAE): 1.6932657342769708
         Mean Squared Error(MSE): 5.846370906092069
         Root Mean Squared Error(RMSE): 2.4179269852690073
         Mean Absolute Error(MAE): 1.69 degrees.
         Accuracy of linear regression model: 86.72 %.
```

Fig. 14 Calculated errors for Linear Regression Model

## VI. RESULTS AND DISCUSSION

We will now proceed with implementing our proposed methodologies, beginning with the k nearest neighbor model. As seen in Figure 15, the scatter plot comparing true values to predicted values (generated by the knn model) of price displays a nearly linear relationship.

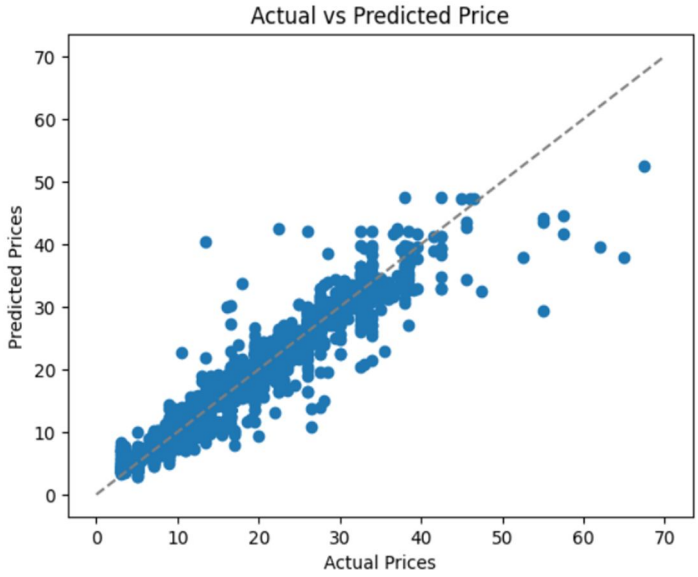



Fig. 15 Scatter plot for k- nearest neighbor

Additionally, Figure 16 shows the R-squared value of the k nearest neighbor model, indicating a performance of 92.84%. It is evident that the model has performed well in its prediction capabilities.

```
In [45]: print("Test R^2 Score: {:.5f}".format(knn_model.score(X_test, y_test)))
Test R^2 Score: 0.92842
```

Fig. 16 R-squared value of k- nearest neighbor

Let us take a look at the actual and predicted prices given by this model(shown in Figure 17).

```
In [41]: df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred_knn})
         df
```

Out[41]:

| | Actual | Predicted |
|---|---|---|
| 9953 | 13.5 | 13.500000 |
| 3850 | 9.0 | 7.000000 |
| 4962 | 11.0 | 10.000000 |
| 3886 | 33.5 | 31.166667 |
| 5437 | 22.5 | 20.500000 |
| ... | ... | ... |
| 5273 | 8.0 | 9.500000 |
| 8014 | 10.5 | 12.000000 |
| 8984 | 11.5 | 8.000000 |
| 6498 | 8.5 | 7.666667 |
| 6327 | 19.5 | 20.500000 |

3000 rows × 2 columns

Fig. 17 Actual and predicted prices of k-nearest neighbor

©IJRASET: All Rights are Reserved 

Various kinds of errors calculated for the k- nearest neighbor model we built along with this model's accuracy are illustrated in Figure 18.

```
In [42]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_knn))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_knn))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_knn)))

         # Calculate the absolute errors
         errors = abs(y_pred_knn - y_test)
         # Print out the mean absolute error (mae)
         print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

         # Calculate mean absolute percentage error (MAPE)
         mape = 100 * (errors / y_test)
         # Calculate and display accuracy
         accuracy = 100 - np.mean(mape)
         print('Accuracy:', round(accuracy, 2), '%.')

Mean Absolute Error: 1.3873333333333333
Mean Squared Error: 5.9132037037037035
Root Mean Squared Error: 2.431707980762432
Mean Absolute Error: 1.39 degrees.
Accuracy: 90.22 %.
```

Fig. 18 Calculated errors for k -nearest neighbor

Moving on to the Support Vector Machine model, its performance can be observed in Figure 18, which displays a scatter plot comparing the true values to the predicted values (generated by the SVM model) of price, demonstrating a nearly linear relationship between them.

Additionally, Figure 19 provides the R-squared value of the SVM model, indicating a performance of 92.94%.

```
In [62]: print("Test R^2 Score: {:.5f}".format(svr_model.score(X_test, y_test)))

Test R^2 Score: 0.92940
```

Fig. 19 R-squared value of SVR model

To get a better sense of the actual and predicted prices given by this model, refer to Figure 20 and further insights into the performance of this model, including various kinds of errors and accuracy, are illustrated in Figure 21.

```
In [47]: df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred_svr})
         df
```

Out[47]:

| | Actual | Predicted |
|---|---|---|
| **9953** | 13.5 | 13.132178 |
| **3850** | 9.0 | 8.115831 |
| **4962** | 11.0 | 10.088228 |
| **3886** | 33.5 | 31.373056 |
| **5437** | 22.5 | 19.581115 |
| **...** | ... | ... |
| **5273** | 8.0 | 8.808487 |
| **8014** | 10.5 | 11.392737 |
| **8984** | 11.5 | 8.106139 |
| **6498** | 8.5 | 7.692410 |
| **6327** | 19.5 | 19.534798 |

3000 rows × 2 columns

Fig. 20 Actual and predicted prices by SVR model

```
In [48]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_svr))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_svr))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_svr)))

         # Calculate the absolute errors
         errors = abs(y_pred_svr - y_test)
         # Print out the mean absolute error (mae)
         print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

         # Calculate mean absolute percentage error (MAPE)
         mape = 100 * (errors / y_test)
         # Calculate and display accuracy
         accuracy = 100 - np.mean(mape)
         print('Accuracy:', round(accuracy, 2), '%.')

Mean Absolute Error: 1.279348074262564
Mean Squared Error: 5.832457656433188
Root Mean Squared Error: 2.4150481685534118
Mean Absolute Error: 1.28 degrees.
Accuracy: 91.16 %.
```

Fig. 21 Calculated errors for SVR model

The final model used in this proposed methodology is the Random Forest model. Figure 22 displays a scatter plot comparing true values to predicted values (generated by the Random Forest model) of price, which shows a near-linear relationship between the two.

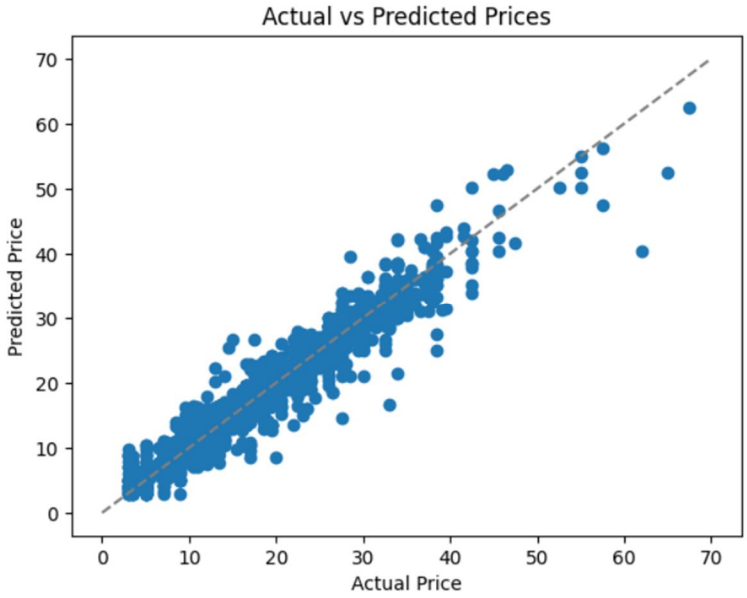



Fig. 22 Scatter plot for Random forest model

The Random Forest model's R-squared value is shown in Figure 23, indicating a performance of 94.70%.

```
In [64]: print("Test R^2 Score: {:.5f}".format(rf_model.score(X_test, y_test)))

Test R^2 Score: 0.94708
```

Fig. 23 R- squared value of Random Forest model

Actual and predicted prices given by this model are shown in Figure 24.

```
In [65]: df=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred_rf})
         df
```

Out[65]:

| | Actual | Predicted |
|---|---|---|
| **9953** | 13.5 | 13.50 |
| **3850** | 9.0 | 5.25 |
| **4962** | 11.0 | 9.00 |
| **3886** | 33.5 | 32.50 |
| **5437** | 22.5 | 19.50 |
| ... | ... | ... |
| **5273** | 8.0 | 10.00 |
| **8014** | 10.5 | 10.50 |
| **8984** | 11.5 | 8.50 |
| **6498** | 8.5 | 8.50 |
| **6327** | 19.5 | 19.50 |

3000 rows × 2 columns

Fig. 24 Actual and predicted prices of Random forest model

Figure 25 illustrates various types of errors calculated for the Random Forest model we built, along with its accuracy.

```
In [66]: print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_rf))
         print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_rf))
         print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_rf)))
         # Calculate the absolute errors
         errors = abs(y_pred_rf - y_test)
         # Print out the mean absolute error (mae)
         print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

         # Calculate mean absolute percentage error (MAPE)
         mape = 100 * (errors / y_test)
         # Calculate and display accuracy
         accuracy = 100 - np.mean(mape)
         print('Accuracy:', round(accuracy, 2), '%.')
```

```
Mean Absolute Error: 1.3190833333333334
Mean Squared Error: 4.371854166666667
Root Mean Squared Error: 2.090897933105934
Mean Absolute Error: 1.32 degrees.
Accuracy: 89.99 %.
```

Fig. 25 Calculated errors for random forest model

Lastly, we create a line plot(Figure-26) to compare the outputs of three regression models (KNN, Random Forest and SVR) against the input values (y_test).

The x-axis represents the input values, while the y-axis represents the output values predicted by each model. Each model is represented by a colored line, with a legend identifying each one.

The title of the plot is "Comparison of Regression Models", which indicates its purpose.

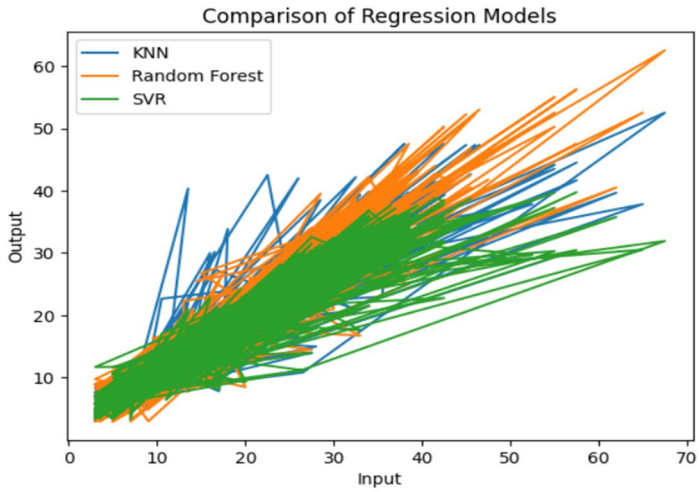



Fig. 26 Comparing the regression models

## VII. CONCLUSION

The field of on-demand ride services has grown significantly in recent years, with the use of technology and the increased demand for convenient transportation services. One of the key factors that impact the success of these services is the pricing strategy used by ride-sourcing companies. In order to provide customers with the best possible experience, it is essential for these companies to accurately predict the prices of their services, taking into account the various factors that may affect them.

In order to achieve accurate predictions of ride-on-demand (RoD) prices, it is important to utilize multiple forecasting algorithms. This approach provides a more comprehensive analysis of the data and can potentially result in more accurate forecasts. As technology continues to evolve, it is critical to stay up-to-date with the latest developments and seek out better approaches than existing classical systems.

However, it is also important for customers to understand the concept of dynamic pricing, a pricing strategy that is commonly used by ride-sourcing companies. Dynamic pricing allows companies to fluctuate prices based on demand and supply, leading to higher prices during periods of high demand and lower prices during periods of low demand. While this strategy may result in higher prices at certain times, it can also provide customers with the opportunity to save money during periods of low demand.

In summary, forecasting RoD prices using multiple algorithms can significantly improve the accuracy of predictions and keep up with the latest technological advancements. However, it is equally important for customers to be aware of the concept of dynamic pricing and how it may affect the prices of their rides. By understanding these factors and methods used by ride-sourcing companies to determine their pricing, customers can make informed decisions and optimize their RoD experience.

## REFERENCES

[1] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. Proceedings of the National Academy of Sciences, 114(3):462–467, 2017.

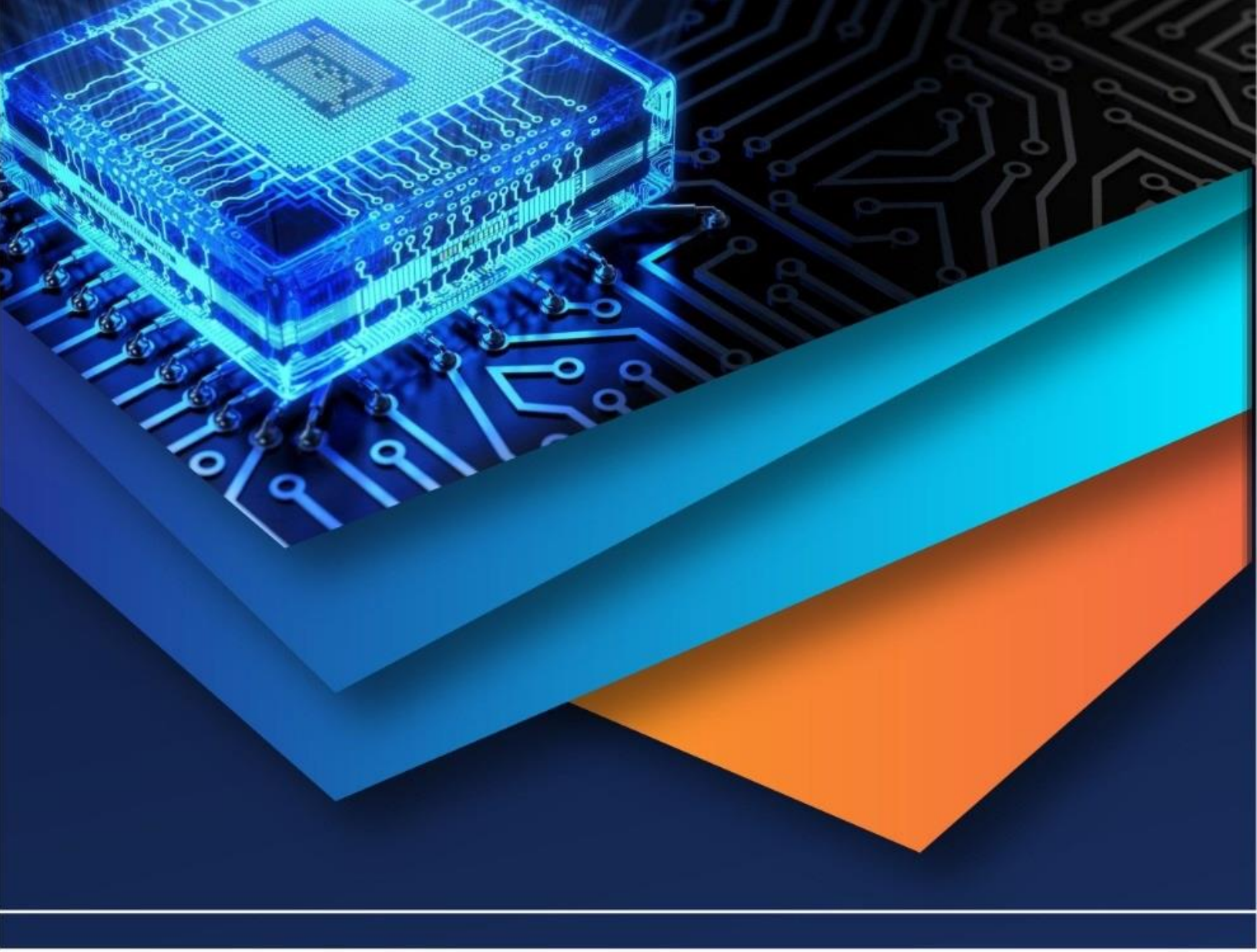
[2] Kunal Arora, Sharanjit Kaur, Vinod Sharma, et al. Prediction of dynamic price of ride-on-demand services using linear regression. International Journal of Computer Applications and Information Technology, 13(1):376–389, 2021.

[3] Matthew Battifarano and Zhen Sean Qian. Predicting real-time surge pricing of ride-sourcing companies. Transportation Research Part C: Emerging Technologies, 107:444–462, 2019.

[4] Suiming Guo, Chao Chen, Jingyuan Wang, Yaxiao Liu, Ke Xu, and Dah Ming Chiu. Dynamic price prediction in ride-on-demand service with multi-source urban data. pages 412–421, 2018.

[5] Liteng Zha, Yafeng Yin, and Zhengtian Xu. Geometric matching and spatial pricing in ride-sourcing markets. Transportation Research Part C: Emerging Technologies, 92:58–75, 2018.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)