RIDE SHARING SERVICES PRICE PREDICTION

A PROJECT REPORT

Submitted by

ANAHITA GUPTA [Reg No: RA2011003010189]

SANCHITA GHOSH [Reg No: RA2011003010191]

Under the Guidance of

Dr. Madhu Mitha K

Associate Professor, Department of Computing Technologies

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR– 603 203

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR–603 203

BONAFIDE CERTIFICATE

Certified that 18CSP109L / I8CSP111L project report titled "**Ride Sharing Services Price Prediction**" is the bonafide work of **Anahita Gupta [Reg. No: RA2011003010189]** and **Sanchita Ghosh [Reg. No: RA2011003010191]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Dr. Madhu Mitha K
SUPERVISOR
Associate Professor
Department of Computing Technologies

Dr. M. Eliazer
PANEL HEAD
Associate Professor
Department of Computing Technologies

Dr. M. PUSHPALATHA
HEAD OF THE DEPARTMENT
Department of Computing Technologies

Department of Computing Technologies

SRM Institute of Science and Technology

Own Work Declaration Form

Degree/Course	: B.Tech in Computer Science and Engineering
Student Names	: Anahita Gupta, Sanchita Ghosh
Registration Number:	RA2011003010189, RA2011003010191
Title of Work	: Ride Sharing Services Price Prediction

I/We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is our own except where indicated, and that we have met the following conditions:

- Clearly references / listed all sources as appropriate.
- Referenced and put in inverted commas all quoted text (from books, web,etc.)
- Given the sources of all pictures, data etc that are not my own.
- Not made any use of the report(s) or essay(s) of any other student(s)either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website.

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above. **Student 1 Signature:** **Student 2 Signature:** **Date:**
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

ACKNOWLEDGEMENT

I express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

I extend my sincere thanks to **T.V.Gopal**, Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

I extend my sincere thanks to **Dr. Revathi Venkataraman**, Professor & Chairperson, School of Computing, SRM Institute of Science and Technology, for her invaluable support.

I wish to thank **Dr. M. Pushpalatha**, Professor & Head, Department of Computing Technologies, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

I am extremely grateful to my Panel Head, **Dr. M. Eliazer**, Assistant Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for his inputs during the monthly reviews.

I register my immeasurable thanks to my Faculty Advisor, **Dr Vinod D.**, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

My inexpressible respect and thanks to my guide, **Dr. Madhu Mitha K** , Assistant Professor, Department Computing Technologies. I sincerely thank staff and students of the Computing Technologies Department, SRM Institute of Science and Technology, for their help during my project work.

Finally, I would like to thank my parents, our family members and our friends for their unconditional love, constant support and encouragement.

Anahita Gupta [RA2011003010189]

Sanchita Ghosh [RA2011003010191]

ABSTRACT

Urban transportation has undergone a monumental shift with the emergence of ride-sharing platforms, offering unparalleled convenience and reshaping commute dynamics. A key challenge in this arena is the accurate prediction of ride fares, especially given the multifaceted variables influencing cost. Among these variables, weather conditions stand out as a significant determinant, influencing both demand and ride duration. This project focuses on harnessing machine learning techniques, primarily the Random Forest algorithm, to predict ride fares with an emphasis on weather-related variables.

Utilizing a comprehensive dataset encompassing historical trips and corresponding meteorological data, the study aims to discern the intricate relationships between various weather conditions - such as temperature, precipitation, and humidity - and ride fares. Feature engineering plays a pivotal role, with weather data transformations and integrations enhancing the prediction model's robustness.

The Random Forest algorithm, known for its capability to handle high-dimensional data and its inherent feature-importance estimation, is employed to model the fare prediction system. Rigorous evaluations, including cross-validation and hyperparameter tuning, ensure the model's accuracy and generalizability.

from this research underscore the profound impact of weather conditions on ride fare predictions. Furthermore, the success of the Random Forest algorithm in this context demonstrates its potential in handling complex prediction tasks in the transportation domain. This study paves the way for advanced research in ride-sharing dynamics, emphasizing the intersection of meteorological factors and transportation economics.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
XGBoost	Extreme Gradient Boosting
ARIMA	Auto Regressive Integrated Moving Average
LSTM	Long Short Term Memory
PCA	Principal Component Analysis

CHAPTER – 1
INTRODUCTION

1.1 General

With the explosive growth of the urban population and the imperative need to address environmental and traffic congestion concerns, ride-sharing services have positioned themselves as a quintessential alternative to traditional modes of transportation. Platforms like Uber and Lyft have not only transformed the way we think about commuting but have also brought about a digital revolution in the transportation industry. As these services gain ground, there is a growing need to understand, anticipate, and optimize pricing strategies to benefit both riders and providers.

This project report delves into the realm of "Ride Sharing Services Price Prediction," aiming to provide a comprehensive analysis and prediction model for ride-sharing prices. By predicting prices more accurately, we can achieve better service optimization, offer competitive pricing to users, and ensure that drivers are compensated fairly. This not only enhances the user experience but also ensures the sustainability and growth of ride-sharing platforms in a fiercely competitive market.

In this project, we will explore the various factors that influence ride-sharing prices, from the base factors like distance and time to more dynamic elements like surge pricing, and mainly the externality like weather conditions. Utilizing data science techniques, we aim to construct a robust predictive model that captures the intricacies of ride-sharing pricing.

1.2 Machine Learning

Machine Learning (ML) provides algorithms, techniques, and models that allow computers to improve their performance on tasks through experience. In the realm of price prediction, ML models can discern patterns from historical data to predict future prices.
The following are the steps in the life cycle of a machine learning model:

1. Data Collection:
Sources: Depending on the domain, data can be sourced from historical databases, online marketplaces, financial instruments, and more.
Relevance: Only data pertinent to the prediction should be collected. For ride-sharing prices, factors like distance, time of day, and weather conditions might be relevant.

2. Data Preprocessing:
Cleaning: Remove duplicates, deal with missing values, and identify any outliers.
Normalization and Standardization: Bring all the numeric variables to a standard scale.
Encoding: Convert categorical variables into a format that can be provided to ML algorithms, typically using techniques like one-hot encoding.

3. Feature Engineering:
Selection: Not all input data is useful. Some features might be redundant, while others can be highly indicative of the price.
Transformation: Combining two or more features, for instance, using the day of the week and the hour to derive a 'peak traffic' feature for a ride-sharing price prediction.
Dimensionality Reduction: Using techniques like PCA (Principal Component Analysis) to reduce the number of features if there are too many.

4. Model Selection & Training:
Linear Regression: A foundational algorithm in which an output variable (price) is predicted based on one or more input features.
Decision Trees & Random Forests: Decision trees divide the data according to feature values, while random forests combine forecasts from several trees to provide a prediction that is more reliable and accurate.

Neural Networks: Multi-layered structures that can model complex non-linear relationships.
Time Series Models: ARIMA and LSTM are tailored for datasets where chronological order matters.

5. Validation & Testing:
Train-Test Split: There are two sets of the dataset: a test set and a training set. On the training set, the model is trained, and on the test set, it is assessed.
Cross-Validation: To avoid overfitting, the data is divided into multiple chunks, and the model is trained and validated several times, each time with a different chunk as the test set.
Metrics: Models are evaluated using metrics depending on the prediction type and domain.

6. Deployment & Real-time Prediction:
Once a model is trained, validated, and tested, it's deployed in a real-world environment. Continuous monitoring and regular updates ensure its sustained accuracy and reliability.

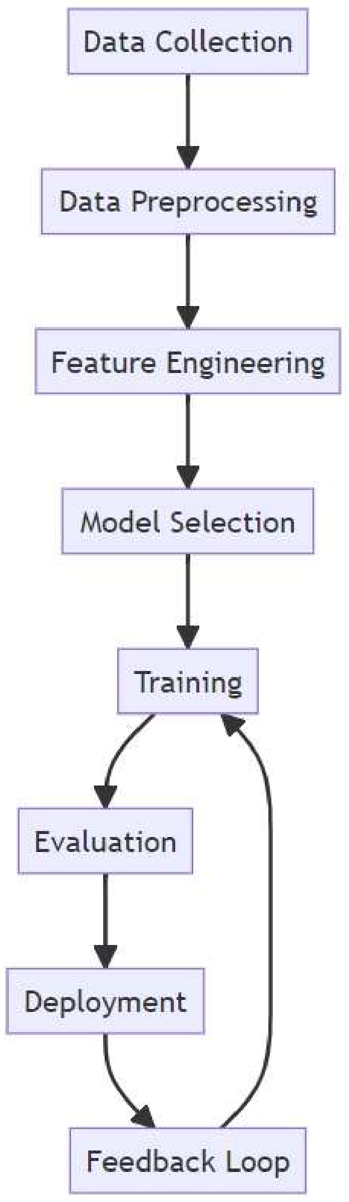


Fig 1.1 Machine Learning Life Cycle

1.3 Machine Learning Models

1.3.1 Linear Regression

Linear regression, a foundational algorithm in statistics and machine learning, seeks to establish a linear relationship between independent and dependent variables. By fitting the best straight line through the data, it provides a clear, interpretable mechanism for understanding and predicting outcomes. However, when deployed to predict ride-sharing prices nuanced by weather conditions, its efficacy encounters hurdles.

At the heart of the matter lies the inherent assumption of linearity in linear regression. This means it assumes a consistent rate of change in the dependent variable (ride-sharing prices) for a unit change in the predictors (weather conditions). Real-world scenarios, especially those influenced by multifaceted factors like weather, rarely conform to such simplistic linear relationships. Weather's impact on ride-sharing prices is likely to be nonlinear, with certain conditions causing disproportional price fluctuations. For instance, the transition from clear skies to light rain might not have the same impact on prices as the shift from light rain to a torrential downpour.

Further complicating matters is the potential interactivity of weather features. Temperature, humidity, wind speed, and precipitation don't exist in isolation; their combined effects can create compound influences on ride-sharing demand and subsequently, prices. Linear regression, in its basic form, struggles to encapsulate these intricate interactions without additional, often complex, feature engineering.

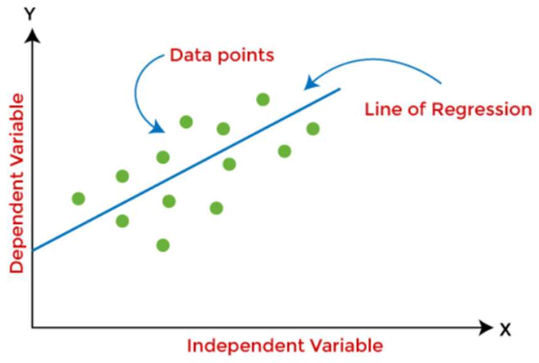


Fig 1.2 Linear Regression

1.3.2 Decision Trees

Decision trees, a cornerstone of machine learning, operate on a flowchart-like model of decisions. At every internal node of the tree, a decision is made based on a feature, leading us down a branch to another decision or an eventual outcome. Simple, interpretable, and straightforward, decision trees have carved a niche in various domains due to their visual appeal and easy-to-understand logic. However, when navigating the intricate waters of predicting ride-sharing prices influenced by weather conditions, decision trees may not be the most efficacious choice, and here's why:

Foremost is the issue of overfitting. Decision trees, particularly when allowed to grow deep, have a tendency to fit too closely to their training data. In a domain like ride-sharing, where countless variables interact — from traffic patterns to local events to the myriad nuances of weather — a decision tree might capture noise rather than genuine underlying patterns. This excessive adaptability to training data can detrimentally affect the model's generalization capability on unseen data, leading to unreliable predictions.

Weather, by its very nature, is a continuous and dynamic variable. Decision trees, however, work on the principle of splitting data based on discrete decisions. This means that subtle changes in weather patterns, which might have a tangible impact on ride-sharing prices, could be glossed over. The granularity and continuity of weather conditions — the difference between light drizzle and a downpour or a temperature gradient through the day — might not be captured adequately by the discrete splits of a decision tree.

Furthermore, decision trees can sometimes be too simplistic. While their interpretable structure is undoubtedly an asset, it can also be a limitation when modeling complex, non-linear relationships. The interplay between weather conditions and ride-sharing prices is multifarious, with various weather components interacting with one another, leading to compound effects on pricing. A single decision tree might struggle to capture these layered interactions, leading to oversimplified models that miss nuanced trends.

1.3.3 Random Forest

An ensemble machine learning technique called Random Forest combines the output of several decision trees to provide predictions that are more reliable and accurate. The Random Forest model uses the predictions made by each decision tree to generate an output, which can be the average prediction for regression tasks or the mode of the classes for classification tasks. The following are the reason why Random Forest Regression is one of the better suited algorithms for price prediction:

- Handling Non-Linearity: The relationship between ride-sharing prices and various weather conditions might not always be linear. Random Forest, being inherently non-linear, is adept at modeling these complex, intertwined relationships.
- Feature Interactions: Weather encompasses diverse factors – temperature, precipitation, wind speed, humidity, and more. These features can interact in intricate ways to influence ride demand and pricing. Random Forest's structure is well-suited to capture and account for these feature interactions.
- Reduction of Overfitting: While individual decision trees can sometimes overfit to training data, the ensemble approach of Random Forest reduces this risk, offering better generalization to unseen data.
- Feature Importance: One of Random Forest's strengths is its ability to rank features by their importance in prediction. This can provide crucial insights, highlighting which weather conditions have the most significant impact on ride-sharing prices.
- Handling Missing Values: Weather datasets can occasionally have missing values. Random Forest has mechanisms to handle these gaps both during training and prediction, ensuring the model remains robust in the face of incomplete data.
- Robustness to Outliers: Sudden and extreme weather changes can introduce outliers in the data. Random Forest's ensemble nature makes it less sensitive to such anomalies, ensuring predictions remain consistent.

Weather's multifaceted nature, combined with its unpredictable shifts, requires a model that is both versatile and resilient. Random Forest, with its capability to process intricate datasets, manage non-linearities, and provide insights into feature relevance, emerges as an especially fitting choice. Its ensemble-based approach acts as a safeguard against erratic predictions – an essential trait when navigating the unpredictability of weather patterns.

Hence, for the task of predicting ride-sharing prices based on weather conditions, Random Forest not only provides a technically sound solution but also aligns with the industry's practical challenges and requirements.

Table 1.1 Difference between Random Forest Classification and Regression

Aspects	**Random Forest Classification**	**Random Forest Regression**
Purpose	Predicting categorical outcomes	Predicting continuous outcomes
Output	Category/Class (e.g., Yes/No)	Numerical Value (e.g., $25.50)
Aggregation method	Mode of the trees' outputs	Mean of the trees' outputs
Evaluation Metrics	Accuracy, Precision, Recall, F1-Score	Mean Absolute Error (MAE), R-Squared (R^2), Mean Squared Error (MSE)
Example Use Case	Will there be high demand tomorrow? (Yes/No)	Predicting the price for a ride tomorrow
Handling of Imbalanced Data	Often requires techniques like oversampling, under sampling, or using balanced class weights	Less sensitive to data imbalances, focuses on minimizing prediction errors

1.3.4 Neural Networks

Neural networks, underpinning many advancements in the realm of deep learning, emulate the human brain's interconnected neuron structure to process and deduce insights from data. Their rise to prominence is anchored in their unparalleled ability to discern intricate patterns within vast and convoluted datasets. However, when applied to predicting ride-sharing prices influenced by weather conditions, their efficacy isn't as clear-cut. Despite their innate power, neural networks might not always be the most astute choice for this particular context. The reasons are manifold: Firstly, the risk of overfitting looms large. Deep neural networks, inherently complex with myriad parameters, can become too attuned to the training data's nuances. In scenarios like ride-sharing, where daily or even hourly weather fluctuations occur, a network might erroneously interpret these ephemeral changes as significant, long-term patterns. Such misinterpretations can compromise the model's generalizability, making it less reliable for real-world predictions.

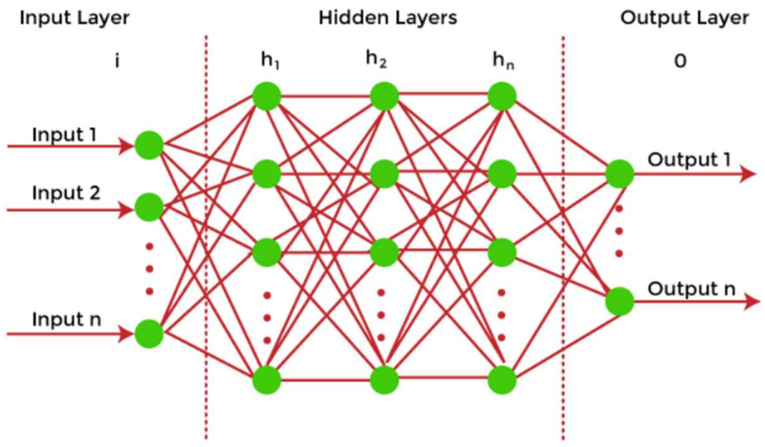


Fig. 1.3 Neural Networks

Moreover, the very complexity that is often a strength of neural networks can, in this case, be an Achilles' heel. The multifaceted relationship between ride-sharing prices and weather, while undeniably intricate, might not require the profound depth a neural network provides. There's a real possibility that simpler models could achieve comparable, if not better, accuracy in predictions while also being more transparent and interpretable.

This brings us to another limitation: the notorious "black box" nature of deep neural networks. In industries like ride-sharing, stakeholders often seek not just predictions, but also comprehensible reasons behind those predictions. Knowing, for instance, how specific weather features influence price shifts can be pivotal for strategy and decision-making. Neural networks, however, tend to obfuscate these causal pathways, making it challenging to glean actionable insights from their predictions.

Furthermore, the data-intensive appetite of neural networks poses another challenge. Their optimal performance hinges on having access to vast, diverse datasets. If there's a paucity of historical data detailing ride-sharing trends under various weather conditions, neural networks may falter, delivering subpar predictive accuracy.

Lastly, practical constraints cannot be overlooked. Training a robust neural network is not only resource-intensive, demanding considerable computational power, but it can also be time-consuming. For an industry like ride-sharing, which might necessitate real-time or swift predictions to dynamically adjust prices, this latency can be a significant impediment.

CHAPTER -2

LITERATURE SURVEY

2.1 Motivation

The rapid transformation and digital innovation in the transportation sector, especially in ride-sharing services, has revolutionized urban mobility. Uber, being at the forefront of this transformation, is constantly striving to provide the best service to its customers. However, the dynamic nature of pricing in ride-sharing platforms, influenced by a myriad of factors such as traffic, weather, and demand-supply dynamics, poses challenges in predicting accurate costs for rides. This unpredictability often results in customers being unsure about the fare they might have to pay, leading to potential dissatisfaction or reduced usage of the service.

The research paper by Bharti Vidhury and team stems from the realization that while several software developers have devised multiple strategies for Uber-like platforms, there hasn't been a comprehensive system that takes into account the holistic needs of a customer's transportation in specific areas [1]. The motivation further deepens when understanding the direct impact of accurate fare prediction on customer satisfaction, trust, and the overall success of ride-sharing platforms.

The overarching objective of the research is to create a Ride Sharing Services price prediction system that can accurately forecast the cost of a ride. Given the multitude of factors influencing the fare – from the time of day, current weather conditions, traffic patterns, to the very surge multiplier dynamic of Uber and Lyft – the challenge is not just to predict, but to predict with a high degree of accuracy [2]. Ride-hailing services have raised concerns about increased traffic congestion and environmental effects, prompting the need for measures like ridesharing or pooling to alleviate these issues. To understand what influences individual choices in ridesharing, researchers employed a machine learning model. Their findings indicate that shorter trips are more likely to be taken individually, and this shift away from sharing can be attributed to the rising costs associated with shared travel on a per-mile basis [3].

Accurate ride-hailing demand predictions are crucial for optimizing vehicle dispatch and driver-passenger matching in urban areas. Machine learning models can help by adapting to the constantly changing supply and demand dynamics across different parts of the city, ultimately enhancing overall traffic efficiency [4].

Recent years have seen the emergence of online ride-hailing services as a crucial element in urban transportation. These services offer convenience to residents, impact travel habits, and introduce diversity into urban mobility. This study conducts a comprehensive review of machine-learning methods applied to on-demand ride-hailing services. It underscores the significance of on-demand ride-hailing in shaping the spatiotemporal dynamics of urban traffic. Additionally, it highlights the value of machine-learning-based research at a macro level in guiding the development, planning, operation, and management of intelligent transportation systems in urban areas [5].

The effectiveness of algorithms like random forest is highly valuable for social scientists, but their utility depends on their ability to access and utilize implementations of these algorithms in their research [6] . Random forest is a powerful technique for predictive modeling. It involves creating multiple decision trees, with each tree being built using a random subset of the data. This approach offers several advantages, including ease of use, low computational requirements, strong predictive performance, versatility, and the ability to provide insights into the data [7].

It delves into the Random Forest model, widely regarded as one of the most effective and commonly employed models within the Machine Learning framework. Random Forest is a powerful ensemble learning technique known for its ability to deliver high predictive accuracy, flexibility, and real-time responsiveness [8]. Random Forest is a popular ensemble classification algorithm known for its effectiveness in handling large datasets. It offers several valuable features, such as Variable Importance measurement, detection of Out-of-Bag errors, assessing proximity between features, and addressing imbalanced datasets [9].

In response to the increasing importance of network traffic analysis and prediction in various sectors, we propose a novel approach. This method involves forecasting traffic congestion levels through the utilization of time series data and machine learning techniques.

This research aims to make valuable contributions to this ever-expanding field [10]. Extreme weather conditions increase the risk of road accidents, making it important to measure the connection between adverse weather and road safety. This data is valuable for planning safety measures, handling vehicle fleets, and setting up alert systems [11].

Weather conditions significantly affect traffic speed in metropolitan cities, making it crucial to effectively utilize weather data for predicting traffic speed up to one week in advance. This study suggests a method of adjusting speed data using weather parameters before inputting it into deep learning algorithms [12]. Its primary objectives include delineating key facets of ride-sharing, such as the online platforms involved, factors related to users, and barriers impacting the adoption of ride-sharing services. Ultimately, the paper aims to distill valuable insights that can inform the effective implementation of ride-sharing initiatives [13].

Random Forest (RF) is an ensemble classification method that has gained significant recognition for its ability to deliver high accuracy and superior results. Recently, the research community has been actively exploring ways to enhance RF's performance, reflecting a shared objective to further improve its capabilities [14]. Random Forest has emerged as a promising classifier for the future due to its notable performance, which is on par with other ensemble techniques such as bagging and boosting. This suggests that Random Forest holds great potential for wider adoption in classification tasks [15].

Cities in North America are increasingly focusing on sustainable transportation due to longer commute times, volatile fuel costs, and a growing concern for the environmental effects of transportation decisions [16]. A study investigates the applicability of the Random Forest machine learning technique for predicting house prices with a focus on price variance rather than specific values. This approach is considered more realistic and appealing for various real-world applications. By using Random Forest, the research aims to provide a more comprehensive understanding of the potential price range and fluctuations in the housing market, which can be valuable for both buyers and sellers [17]. This paper addresses the growing interest in investing in clean energy companies due to factors like climate change, green consumer preferences, energy security, fossil fuel divestment, and technological innovation. It acknowledges the challenges in predicting stock prices but suggests that forecasting stock price direction might be more reliable. To achieve this, the paper employs the machine learning technique of random forests to predict the direction of stock prices for clean energy exchange traded funds [18]. This paper underscores the significant challenges scientists face in accurately predicting weather, particularly for tropical systems, due to the complexity of numerical climate models. The primary focus of the paper is to highlight the crucial role of machine learning (ML) techniques in enhancing weather prediction [19]. Complex numerical climate models present significant challenges to scientists when predicting weather, particularly for tropical systems. This paper emphasizes the significance of weather prediction through machine learning (ML) techniques. Many researchers have suggested that ML models can generate accurate weather forecasts, even in the absence of a deep understanding of atmospheric physics [20].

2.2 Objective

The objective of the research is to create a Ride Sharing Services price prediction system that can accurately forecast the cost of a ride. Given the multitude of factors influencing the fare – from the time of day, current weather conditions, traffic patterns, to the very surge multiplier dynamic of Uber and Lyft – the challenge is not just to predict, but to predict with perfect accuracy.

The paper aims to delve deep into the vast repositories of historical trip data, extracting patterns and understanding correlations between different influencing factors. By employing a combination of machine learning algorithms, the objective is to discern which model or combination of models provides the most accurate fare prediction. This includes exploring linear regression, decision trees, random forests, gradient boosting, and more.

Another key objective is to reduce the unpredictability associated with Uber fares. By achieving a higher level of accuracy in fare prediction, the system can mitigate the uncertainty that users often face, thereby enhancing user experience and trust in the platform.

CHAPTER -3

ARCHITECTURE AND ANALYSIS

3.1 Block/Architecture Diagram

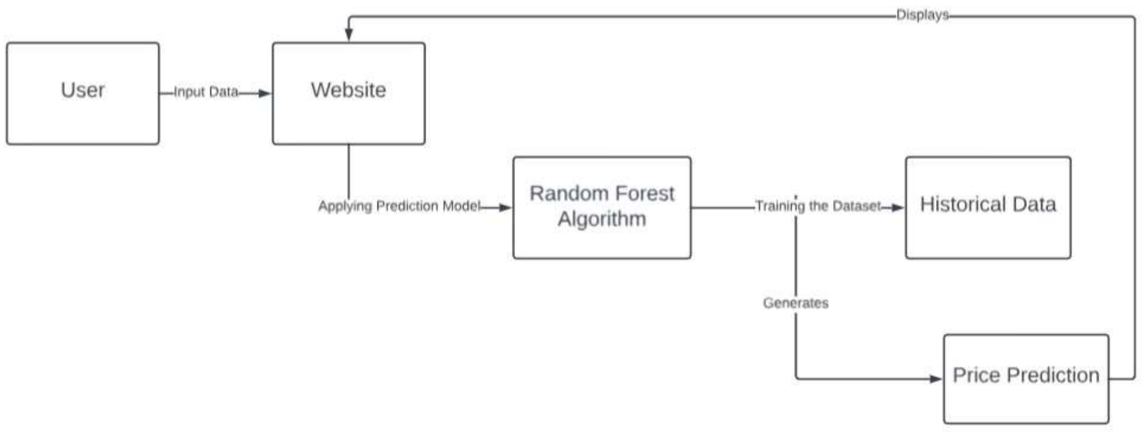


Fig 3.1 Block diagram for Ride Sharing Services Price Prediction

The process is described as follows:

1. User Input: Users provide relevant data, such as the starting location, destination, and the weather conditions such as source and destination temperature, humidity etc, on a website.

2. Data Collection: The user's input is collected and sent to the system for analysis. This data includes the ride details and the weather conditions at the time of the ride.

3. Data Cleaning: After data collection, the system performs data cleaning, which involves preprocessing the user input to ensure it is in a suitable format and free from errors or inconsistencies. This step is essential for preparing the data for analysis.

4. Random Forest Classifier Algorithm: The cleaned and processed data is then fed into a Random Forest Classifier algorithm. In this context, the algorithm is used to make predictions based on historical data. The algorithm takes into account the input data, cleaned and outlier-treated, and the historical weather and ride pricing information to classify the potential ride price.

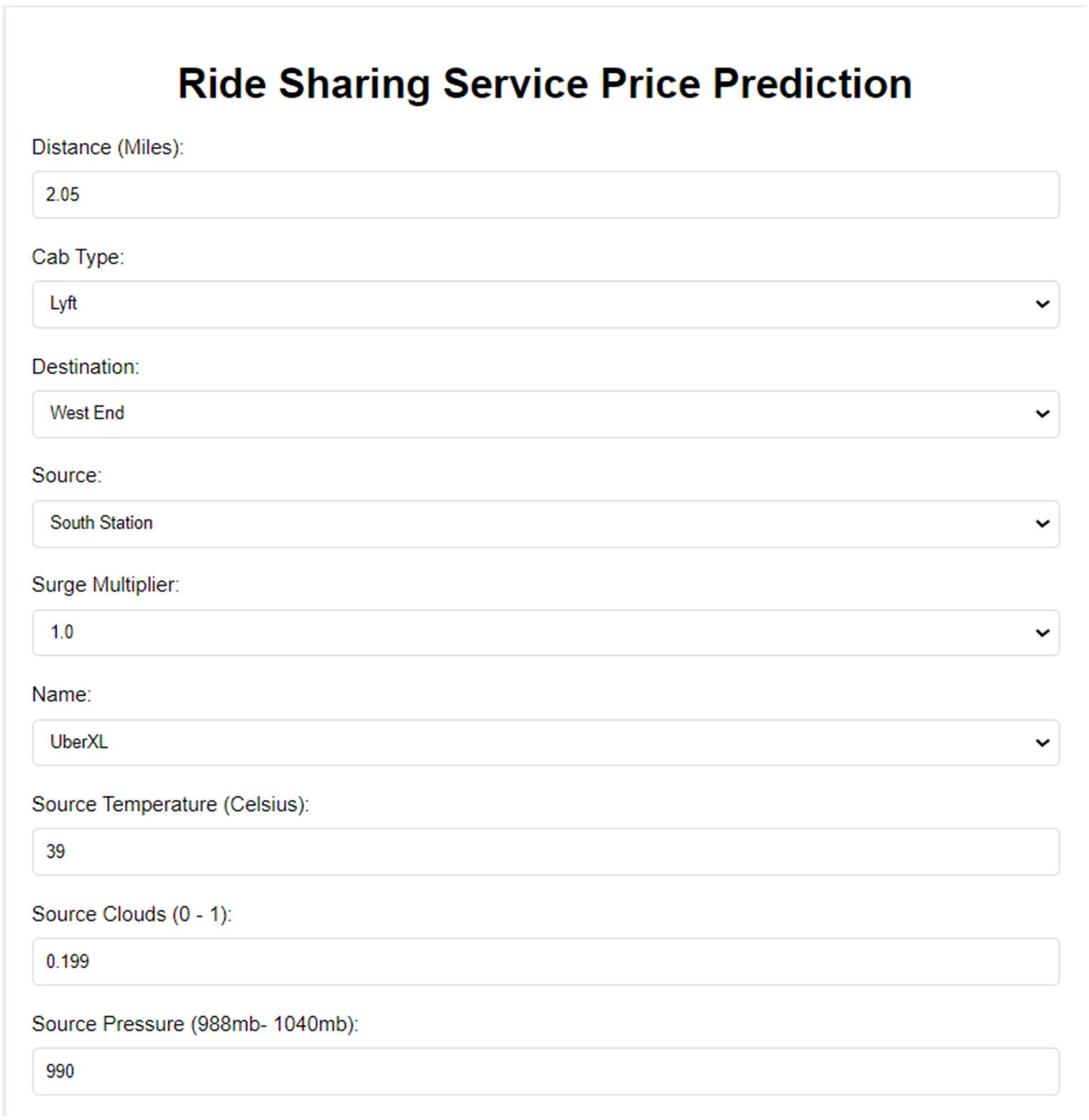
5. Training with Historical Data: The Random Forest Classifier algorithm is trained with historical data, which includes a large dataset of past ride information and corresponding weather conditions. This training helps the algorithm understand the patterns and relationships between weather and ride pricing.

6. Dataset Generation: After training, the algorithm generates a dataset that contains predictions for ride prices based on the input data. This dataset is a collection of potential price predictions.

7. Price Prediction Analysis: The generated dataset is then used for price prediction analysis. The algorithm provides a predicted ride price based on the user's input and the associated weather conditions, taking into account the cleaned data.

8. Website Display: The results of the price prediction analysis are displayed on the website or application. Users can view the predicted ride price, which is now influenced by the weather conditions at the specified date and time.

3.2 Frontend Design



Ride Sharing Service Price Prediction

Distance (Miles): 2.05

Cab Type: Lyft

Destination: West End

Source: South Station

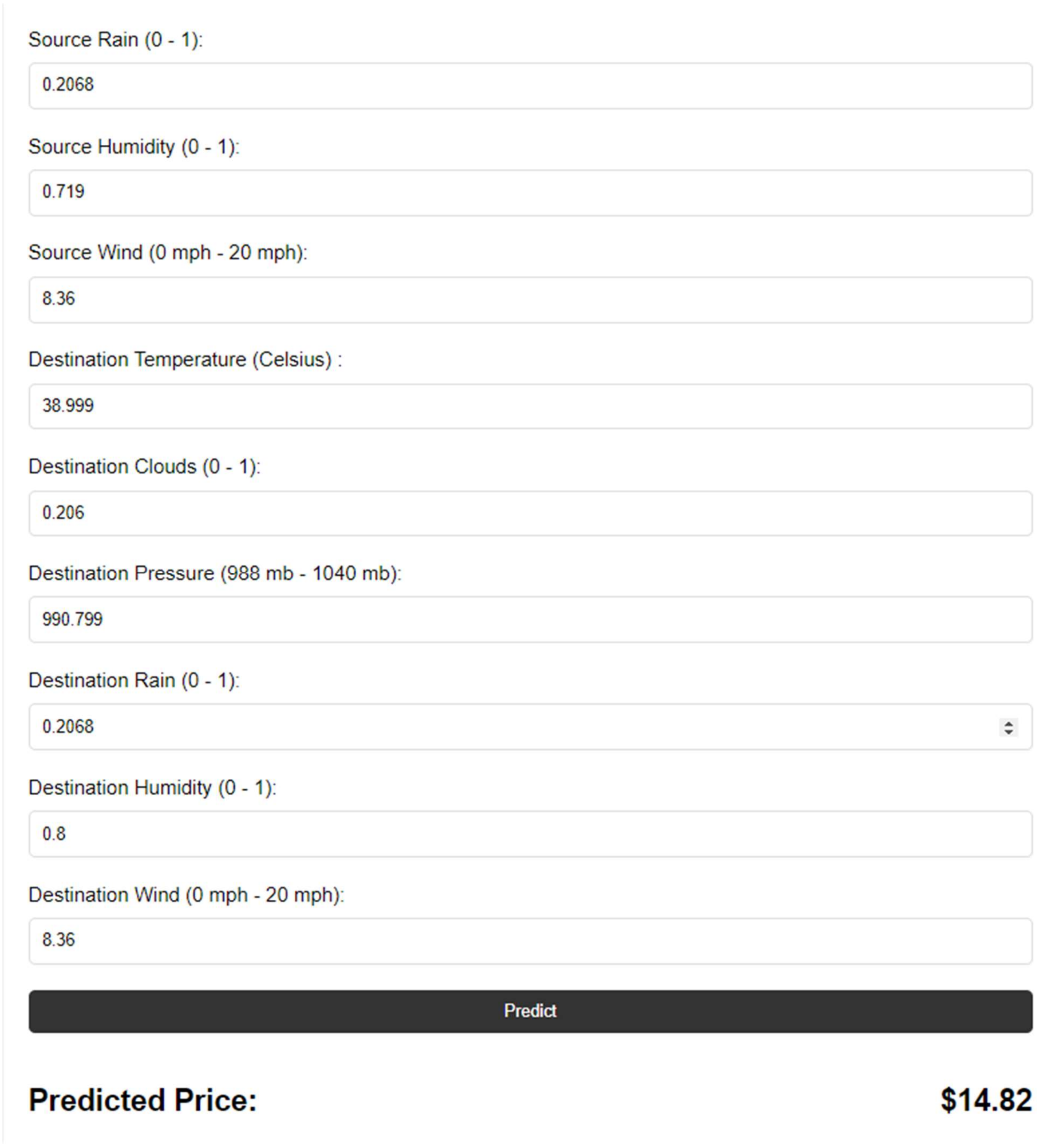
Surge Multiplier: 1.0

Name: UberXL

Source Temperature (Celsius): 39

Source Clouds (0 - 1): 0.199

Source Pressure (988mb- 1040mb): 990



Source Rain (0 - 1):
0.2068

Source Humidity (0 - 1):
0.719

Source Wind (0 mph - 20 mph):
8.36

Destination Temperature (Celsius) :
38.999

Destination Clouds (0 - 1):
0.206

Destination Pressure (988 mb - 1040 mb):
990.799

Destination Rain (0 - 1):
0.2068

Destination Humidity (0 - 1):
0.8

Destination Wind (0 mph - 20 mph):
8.36

Predicted Price: $14.82

Fig 3.2 User Interface for Ride Sharing Services Price Prediction

3.3 Backend Design

i) main.ipynb

```
import pandas as pd

import joblib



# Load the datasets

rides_df = pd.read_csv('cab_rides.csv')

weather_df = pd.read_csv('weather.csv')



# Display the first few rows of each dataset

rides_df_head = rides_df.head()

weather_df_head = weather_df.head()



rides_df_head, weather_df_head



# Display the summary information of each dataset

rides_info = rides_df.info()

weather_info = weather_df.info()



# Display the number of missing values in each dataset

rides_na_sum = rides_df.isna().sum()

weather_na_sum = weather_df.isna().sum()
```

```
rides_na_sum, weather_na_sum


# Handling missing values
ride_df = rides_df.dropna(axis=0).reset_index(drop=True)
weather_df['rain'] = weather_df['rain'].fillna(0)


# Grouping weather data by location and calculating the mean for each location
avr_weather_df = weather_df.groupby('location').mean().reset_index(drop=False)
avr_weather_df = avr_weather_df.drop('time_stamp', axis=1)


# Renaming columns of the grouped weather data to prepare for merging
source_weather_df = avr_weather_df.rename(
    columns={
        'location': 'source',
        'temp': 'source_temp',
        'clouds': 'source_clouds',
        'pressure': 'source_pressure',
        'rain': 'source_rain',
        'humidity': 'source_humidity',
        'wind': 'source_wind'
    }
)
destination_weather_df = avr_weather_df.rename(
    columns={
        'location': 'destination',

        'temp': 'destination_temp',

        'clouds': 'destination_clouds',

        'pressure': 'destination_pressure',

        'rain': 'destination_rain',

        'humidity': 'destination_humidity',

        'wind': 'destination_wind'

    }

)


# Merging datasets based on 'source' and 'destination' columns

data = ride_df\

    .merge(source_weather_df, on='source')\

    .merge(destination_weather_df, on='destination')


data.head()


import matplotlib.pyplot as plt

import seaborn as sns


numerical_data = data.select_dtypes(include=['float64', 'int64'])


# Calculate correlations for these numerical columns

correlations = numerical_data.corr()
```

```
# Plotting the heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(correlations, cmap='coolwarm', annot=True, fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

def label_encode(df, column):
    df = df.copy()
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    return df, le

def preprocess_inputs_label_encode(df):
    df = df.copy()

    # Drop unnecessary columns
    df = df.drop(['id', 'time_stamp', 'product_id'], axis=1)

    # Binary encode cab_type column
```

```
df['cab_type'] = df['cab_type'].replace({'Lyft': 0, 'Uber': 1})


# Label encode categorical columns and save the encoders
label_encoders = {}  # Dictionary to store the encoders
for column in ['destination', 'source', 'name']:
    df, encoder = label_encode(df, column)
    label_encoders[column] = encoder


# Save the label encoders
for column, encoder in label_encoders.items():
    joblib.dump(encoder, f"{column}_encoder.pkl")


# Split df into X and y
y = df['price']
X = df.drop('price', axis=1)


# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=True,
 random_state=1)


# Scale X and save the scaler
scaler = StandardScaler()
scaler.fit(X_train)
```

```
    # Save the scaler
    joblib.dump(scaler, "scaler.pkl")

    X_train = pd.DataFrame(scaler.transform(X_train), columns=X.columns)
    X_test = pd.DataFrame(scaler.transform(X_test), columns=X.columns)

    return X_train, X_test, y_train, y_test


# Preprocessing and splitting the dataset into training and testing sets
print(data.columns)
X_train, X_test, y_train, y_test = preprocess_inputs_label_encode(data)

X_train.head(), y_train.head()

X_train.columns

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Train a Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=1)
rf_model.fit(X_train, y_train)
```

```
# Predictions on the testing set
y_pred = rf_model.predict(X_test)

# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mae, mse, r2

# Visualizing actual vs. predicted prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red') # y=x line
plt.title('Actual vs. Predicted Prices')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.grid(True)
plt.show()

# Displaying some sample predictions from the test set
sample_predictions = pd.DataFrame({
    'Actual Prices': y_test,
    'Predicted Prices': y_pred
}).reset_index(drop=True)

sample_predictions.head(10)

import numpy as np

# Generating random samples from the preprocessed dataset
np.random.seed(1)
random_indices = np.random.choice(X_test.index, size=10, replace=False)
random_samples = X_test.loc[random_indices]

# Predicting on the random samples
random_predictions = rf_model.predict(random_samples)

# Combining the samples and predictions into a DataFrame
random_samples_with_predictions = random_samples.copy()
random_samples_with_predictions['Predicted Price'] = random_predictions

random_samples_with_predictions

random_samples_with_predictions.columns

import joblib
```

```
    # Define the path to save the model

    model_path = 'model.pkl'


    # Save the model to a file

    joblib.dump(rf_model, model_path)


    model_path
```

ii) app.py

```
from flask import Flask, render_template, request, jsonify
import joblib
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

# Load the model and the unique values CSV
model = joblib.load('model.pkl')
scaler = joblib.load('scaler.pkl')
destination_encoder = joblib.load('destination_encoder.pkl')
source_encoder = joblib.load('source_encoder.pkl')
name_encoder = joblib.load('name_encoder.pkl')

unique_values = pd.read_csv('unique_values.csv')

for column in ['cab_type', 'surge_multiplier']:
    # Get the unique cleaned values as a list
    cleaned_values          =          unique_values[column].replace(['NaN',          'nan',          ''],
'NA').dropna().unique().tolist()

    # Replace the entire column with the cleaned list
    unique_values[column] = cleaned_values + ['NA'] * (len(unique_values) -
len(cleaned_values))



data = pd.read_csv('cab_rides.csv')

def preprocess_input_data(data):
    df = pd.DataFrame([data])
    df['cab_type'] = df['cab_type'].replace({'Lyft': 0, 'Uber': 1})

    df['destination'] = destination_encoder.transform(df['destination'])
    df['source'] = source_encoder.transform(df['source'])
    df['name'] = name_encoder.transform(df['name'])

    df = pd.DataFrame(scaler.transform(df), columns=df.columns)

    return df

def get_distance(source, destination):
    # Extract the rows that match the given source and destination
    matching_rows = data[(data['source'] == source) & (data['destination'] == destination)]
    if matching_rows.shape[0] > 0:
        return matching_rows.iloc[0]['distance']
    else:
        return None

@app.route("/")
def index():
    print(unique_values.head(10))
    return render_template('index.html', data=unique_values)

@app.route("/get-distance", methods=["POST"])
def get_distance_endpoint():
    source = request.form['source']
    # print(source)
    destination = request.form['destination']
    distance = get_distance(source, destination)
    print(distance)
    return jsonify({'distance': distance})


@app.route("/predict", methods=["POST"])
def predict():
    # Extract the data from the POST request
    data = {
        'distance': float(request.form['distance']),
        'cab_type': request.form['cab_type'],
        'destination': request.form['destination'],
        'source': request.form['source'],
        'surge_multiplier': float(request.form['surge_multiplier']),
        'name': request.form['name'],
        'source_temp': float(request.form['source_temp']),
        'source_clouds': float(request.form['source_clouds']),
        'source_pressure': float(request.form['source_pressure']),
        'source_rain': float(request.form['source_rain']),
        'source_humidity': float(request.form['source_humidity']),
        'source_wind': float(request.form['source_wind']),
        'destination_temp': float(request.form['destination_temp']),
        'destination_clouds': float(request.form['destination_clouds']),
        'destination_pressure': float(request.form['destination_pressure']),
        'destination_rain': float(request.form['destination_rain']),
        'destination_humidity': float(request.form['destination_humidity']),
        'destination_wind': float(request.form['destination_wind'])
    }

    # Preprocess the data using the same preprocessing steps as before
    input_data = preprocess_input_data(data)

    # Make the prediction using the model
    prediction = model.predict(input_data)

    # Return the predicted price
    return jsonify({"price": prediction[0]})

if __name__ == '__main__':
    app.run(debug=True)
```

iii) index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Cab Price Prediction</title>
    <link rel="stylesheet" href="static/styles.css" />
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  </head>

  <body>
    <div class="container">
      <h1>Cab Price Prediction</h1>

      <form id="prediction-form">
```

```
<label for="distance">Distance (Miles):</label>
<input
  type="number"
  id="distance"
  step="0.01"
  name="distance"
  required
/>

<label for="cab_type">Cab Type:</label>
<select id="cab_type" name="cab_type">
    {% for val in data.cab_type %}
    {% if val != "NaN" and val != "" and val != "nan" and val != "NA" %}
    <option value="{{ val }}">{{ val }}</option>
    {% endif %}
    {% endfor %}
</select>

<label for="destination">Destination:</label>
<select id="destination" name="destination">
  {% for val in data.destination %}
  {% if val != "NaN" and val != "" and val != "nan" and val != "NA" %}
  <option value="{{ val }}">{{ val }}</option>
  {% endif %}
  {% endfor %}
</select>

<label for="source">Source:</label>
<select id="source" name="source">
  {% for val in data.source %}
  {% if val != "NaN" and val != "" and val != "nan" and val != "NA" %}
  <option value="{{ val }}">{{ val }}</option>
  {% endif %}
  {% endfor %}
</select>

<label for="surge_multiplier">Surge Multiplier:</label>
<select id="surge_multiplier" name="surge_multiplier">
    {% for val in data.surge_multiplier %}
    {% if val != "NaN" and val != "" and val != "nan" and val != "NA" %}
    <option value="{{ val }}">{{ val }}</option>
    {% endif %}
    {% endfor %}
</select>

<label for="name">Name:</label>
<select id="name" name="name">
  {% for val in data.name %}
  {% if val != "NaN" and val != "" and val != "nan" and val != "NA" %}
  <option value="{{ val }}">{{ val }}</option>
  {% endif %}
  {% endfor %}
</select>

<!-- Additional Features -->

<label for="source_temp">Source Temperature (Celsius):</label>
<input
  type="number"
  id="source_temp"
  step="0.001"
  name="source_temp"
  required
/>

<label for="source_clouds">Source Clouds:</label>
```

```
<input
  type="number"
  id="source_clouds"
  step="0.001"
  name="source_clouds"
  required
/>

<label for="source_pressure">Source Pressure:</label>
<input
  type="number"
  id="source_pressure"
  step="0.001"
  name="source_pressure"
  required
/>

<label for="source_rain">Source Rain (0 - 1):</label>
<input
  type="number"
  id="source_rain"
  step="0.00001"
  name="source_rain"
  required
/>

<label for="source_humidity">Source Humidity:</label>
<input
  type="number"
  id="source_humidity"
  step="0.001"
  name="source_humidity"
  required
```

```
/>

<label for="source_wind">Source Wind:</label>
<input
  type="number"
  id="source_wind"
  step="0.001"
  name="source_wind"
  required
/>

<label for="destination_temp"
  >Destination Temperature (Celsius) :</label
>
<input
  type="number"
  id="destination_temp"
  step="0.001"
  name="destination_temp"
  required
/>

<label for="destination_clouds">Destination Clouds:</label>
<input
  type="number"
  id="destination_clouds"
  step="0.001"
  name="destination_clouds"
  required
/>

<label for="destination_pressure">Destination Pressure:</label>
<input
```

```
  type="number"
  id="destination_pressure"
  step="0.001"
  name="destination_pressure"
  required
/>

<label for="destination_rain">Destination Rain (0 - 1):</label>
<input
  type="number"
  id="destination_rain"
  step="0.00001"
  name="destination_rain"
  required
/>

<label for="destination_humidity">Destination Humidity:</label>
<input
  type="number"
  id="destination_humidity"
  step="0.001"
  name="destination_humidity"
  required
/>

<label for="destination_wind">Destination Wind:</label>
<input
  type="number"
  id="destination_wind"
  step="0.001"
  name="destination_wind"
  required
/>
```

```
    <button type="submit">Predict</button>
  </form>

  <div id="prediction-result">
    <h2>Predicted Price:</h2>
    <p id="price"></p>
  </div>
</div>

<script>
  function updateDistance() {
    const source = $("#source").val();
    const destination = $("#destination").val();
    $.post(
      "/get-distance",
      { source: source, destination: destination },
      function (data) {
        $("#distance").val(data.distance);
      }
    );
  }

  $(document).ready(function () {
    $("#source, #destination").change(updateDistance);

    $("#prediction-form").submit(function (event) {
      event.preventDefault(); // Prevent the default form submission

      const formData = $(this).serialize();

      $.post("/predict", formData, function (data) {
        $("#price").text("$" + data.price.toFixed(2));
```

```
          });
        });
      });
    </script>
  </body>
</html>
```

iv) style.css

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    margin: 0;
    padding: 0;
}

.container {
    max-width: 800px;
    margin: 50px auto;
    background-color: #fff;
    padding: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

h1, h2 {
    text-align: center;
}

label {
    display: block;
    margin-bottom: 10px;
}
```

```
input, select, button {
    width: 100%;
    padding: 10px;
    margin-bottom: 20px;
    border: 1px solid #ddd;
    border-radius: 5px;
}

input[type="number"], select {
    box-sizing: border-box;  /* ensures padding and border are included in the total width */
    width: 100%;             /* take up the full width of the parent container */
}

button {
    cursor: pointer;
    background-color: #333;
    color: #fff;
    border: none;
}

button:hover {
    background-color: #555;
}

#prediction-result {
    display: flex;          /* use flexbox to display children side-by-side */
    justify-content: space-between; /* space out the children equally */
    align-items: center;    /* vertically align the children in the middle */
    margin-top: 20px;
}

#prediction-result h2 {
    margin: 0;
}

#prediction-result p#price {
    margin: 0;
    font-size: 24px;        /* increased font size for emphasis */
    font-weight: bold;      /* make the price bold */}
```

CHAPTER -4
DESIGN AND IMPLEMENTATION

4.1 Dataset

4.1.1 Cab Rides Dataset (cab_rides.csv)

The cab_rides.csv dataset provides detailed information about individual cab rides, capturing various parameters that can influence the fare.

Features:

- distance (numeric): Represents the distance of the ride in miles or kilometers. This feature is crucial as the fare for ride-sharing services is often directly proportional to the distance traveled.

- cab_type (categorical): Specifies the type of cab service (e.g., Lyft). This can influence the price as different services may have different pricing structures.

- time_stamp (datetime): Indicates the exact time when the ride data was captured. This can be essential for understanding the time-based variability in prices, such as peak hours, weekends, or special events.

- destination (categorical): The endpoint of the ride. Locations can have varying demands, which might influence the pricing.

- source (categorical): The starting point of the ride. Just like the destination, the source can also influence the fare based on its demand.

- price (numeric): The fare of the ride. This is the target variable for a price prediction model.

- surge_multiplier (numeric): A factor applied to the base price during times of high demand. A surge multiplier greater than 1 indicates that the fare was increased due to high demand.

- id (categorical): A unique identifier for each ride. This helps in uniquely identifying records but might not be directly useful for prediction.

- product_id (categorical): Represents the specific product or type of ride offered by the service (e.g., lyft_line, lyft_premier). Different ride types can have different fares.

- name (categorical): A more descriptive name for the ride type (e.g., Shared, Lux). This can give insights into the luxury level or capacity of the vehicle, which can influence the price.

4.1.2 Weather Dataset (weather.csv)

The weather.csv dataset provides weather conditions for different locations at various timestamps. Since weather can influence ride prices, integrating this data can enhance the accuracy of predictions.

Features:

- temp (numeric): The temperature recorded in the location. Extreme temperatures might lead to higher demand and possibly higher prices.

- location (categorical): The area where the weather data was recorded. It corresponds to the source or destination in the rides dataset.

- clouds (numeric): Represents the cloud cover as a percentage. High cloud cover might indicate overcast conditions.

- pressure (numeric): Atmospheric pressure in the location. While this might not directly influence cab prices, it can be an indicator of weather stability.

- rain (numeric): The amount of rainfall. Rain can significantly influence ride demand and prices, especially in areas where people rely heavily on ride-sharing during inclement weather.

- time_stamp (datetime): The exact time when the weather data was recorded. Helps in mapping weather conditions to specific rides.

- humidity (numeric): The humidity level as a percentage. Like temperature, extreme humidity levels might influence ride demand.

- wind (numeric): Wind speed at the location. Strong winds, especially during storms, can influence ride prices.

Both datasets provide valuable insights into factors that can influence ride-sharing prices. By integrating the weather data with the ride data, one can create a more comprehensive dataset to train a model that predicts prices with higher accuracy. The temporal and spatial dimensions (time and location) are critical linkers between these datasets.

4.2 Data Loading and Exploration

The initial step in any data analysis or modeling task is to load and understand the data. In this project, two distinct datasets were loaded, which presumably contained information related to cab rides and associated weather conditions. Using Pandas, a powerful data manipulation library in Python, these datasets were read into DataFrames — a tabular data structure that allows for easy data manipulation and analysis.

Upon loading, a preliminary exploration was performed. This is akin to a cursory glance over the data, typically achieved using methods like head(), which reveals the first few rows of the DataFrame. Such an exploration aids in familiarizing oneself with the columns, potential features, and the general structure of the data.

To further this understanding, a summary was generated using the .info() method. This is crucial as it provides insights into data types, which can influence subsequent preprocessing steps, and highlights non-null counts, giving an early indication of missing data. Recognizing missing values at this stage helps in formulating strategies to handle them in the preprocessing phase.

4.3 Data Preprocessing and Merging

Once familiarized with the data, preprocessing becomes paramount. Missing data, often a reality in real-world datasets, needs careful handling. In this project, a direct approach was taken: rows with missing values in one dataset were discarded, and specific missing values in the weather dataset, such as 'rain', were replaced with a default value (0 in this case).

Given that the project aimed to predict ride prices based on both cab ride details and weather conditions, the weather data was aggregated by location. This involved computing mean values, ensuring a concise representation of weather conditions for each location. To facilitate a seamless merging of data, columns were aptly renamed. This step is vital to prevent confusion and ensure clarity when datasets are combined.

The ultimate merging of the cab ride data with the weather data is a crucial step. It essentially associates each ride's source and destination with the relevant weather conditions, thus providing a holistic view of the factors that might influence ride prices.

4.4 Data Visualization

Visualization aids in turning raw data into understandable and interpretable insights. In this project, the correlation heatmap played a pivotal role. By visualizing correlations among numerical variables, it becomes feasible to discern potential relationships or even multicollinearity — where two or more variables are highly correlated, which can adversely impact certain machine learning models.

4.5 Dataset Splitting and Preprocessing

At this juncture, one-hot encoding, a technique to convert categorical variables into a machine-readable format, was introduced. A bespoke function was devised to handle this transformation, showcasing the importance of catering encoding methods to the specific needs of the dataset.

Following encoding, a comprehensive preprocessing function was formulated. This function orchestrated several critical tasks: dropping irrelevant columns, encoding necessary variables, segregating data into features and target variables, and partitioning these into training and testing sets. The latter is vital for model evaluation, as testing on unseen data is a proxy for real-world performance. Lastly, numerical features were scaled, ensuring they all contribute equally to model training, irrespective of their original scales.

4.6 Model Training

An essential phase in the machine learning process is model training. In order to construct a model that can generate predictions or choices without needing to be explicitly trained for the task, an algorithm is fed a dataset. Here's a closer look at the Random Forest Regressor's training phase, which was the model of choice for this project:

1. Algorithm Selection: Random Forest Regressor

A flexible ensemble machine learning approach called Random Forest builds several decision trees during training and produces the average prediction of each tree when applied to regression situations. Numerous factors support its choice, those are:

- Ensemble Nature: By leveraging the power of multiple decision trees and averaging their outputs, Random Forest reduces the possibility of overfitting, which is a typical issue with single decision trees..
- Complex Data Handling: Random Forest can manage a mix of numerical and categorical data, making it apt for datasets with varied attributes.
- Feature Importance: The algorithm prioritizes features according to how crucial they are for producing precise forecasts, offering valuable insights into which variables significantly influence the outcome.

2. Data Feeding

The preprocessed training data, split into features (X) and target variables (y), was fed into the Random Forest algorithm. This data provides the foundation upon which the model learns the underlying patterns and relationships.

3. Hyperparameters and Model Configuration

Random Forest has hyperparameters that can be adjusted to maximize its performance, just like the majority of machine learning algorithms. These could be the total number of trees in the forest, the trees' maximum depth, and the least amount of samples needed to split a node, among other things. While default values frequently offer a good starting point, the predictive accuracy of the model can be improved by adjusting these parameters using methods like grid search or random search.

4. Model Fitting

The algorithm "learns" from the training data during the fitting process. This entails building several decision trees for Random Forest, each trained on a different subset of the data. Each of these trees makes an effort to forecast the target variable using its features. Because Random Forest is an ensemble method, its power comes from the final prediction being the sum of all trees' predictions, or, in the case of regression tasks, an average.

5. Internal Evaluation

Once the model is trained, it's common practice to evaluate its performance on the training data itself, before external validation on a separate test set. This provides an initial gauge of the model's accuracy and can highlight potential issues like overfitting.

6. Iterative Refinement

The process of training models is not always one-time. Based on initial evaluations, one might return to earlier steps, perhaps tweaking hyperparameters, considering different feature sets, or even exploring other algorithms. This iterative nature ensures the development of a model that's not just theoretically sound, but also practically effective.

7. Model Saving

After investing effort in training and tuning the model, it's prudent to save it for future use. This project employed joblib, a serialization library in Python, to achieve this. By saving the trained model to a file, one can easily reload and reuse it, negating the need for retraining and ensuring consistency in predictions.

4.7 Evaluation Metrics

The benchmarks used to assess the efficacy and performance of machine learning models are called evaluation metrics. They offer quantifiable insights into a model's performance in terms of its classifications or predictions. Different metrics may be needed for evaluation depending on the problem and algorithm. The evaluation metrics selected for this project were MAE (Mean Absolute Error), MSE (Mean Squared Error), and R-Squared (Coefficient of Determination). The project is a regression task with the goal of predicting ride prices.

1. Mean Absolute Error (MAE)

Disregarding their direction, MAE calculates the average size of errors between predicted and actual values.

Formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.1}$$

A smaller MAE indicates a better fit of the model to the data. In the context of ride price prediction, it represents the average amount by which the model's predictions deviate from the actual prices.

2. Mean Squared Error (MSE)

MSE is the average of squared differences between the predicted and actual values. It gives more weight to larger errors.

Formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (4.2)$$

Just like MAE, a lower MSE indicates a better model fit. However, since errors are squared, MSE tends to penalize larger errors more heavily. This means that if the model makes a few very poor predictions, the MSE will highlight these more than the MAE.

3. R- Squared Score (Coefficient of Determination):

The percentage of the dependent variable's variance that can be predicted from the independent variables is shown by the R^2 score.

Formula:

$$R^2 = 1 - \frac{Sum\ of\ Squares\ of\ Residuals}{Total\ Sum\ of\ Squares} \qquad (4.3)$$

R^2 values range from 0 to 1, where higher values suggest that a greater percentage of the variance in the dependent variable is explained by the model. For example, an R^2 score of 0.9 indicates that 90% of the output variability can be explained by the model. In the context of this project, a high R^2 would mean that most of the variability in ride prices can be explained by the features in the model, including weather conditions.

4.8 Random Forest in Ride Sharing Price Prediction

4.8.1 Introduction to Random Forest

Random Forest is a forest of decision trees. Instead of relying on a single decision tree, it builds upon the foundational idea that a group (or "ensemble") of weak learners can gather together to form a strong learner. Let's dive deeper into its components.

Basics of Decision Trees

A decision tree is a structure that resembles a flowchart, with each leaf node representing an outcome, each branch representing a decision rule, and each internal node representing a feature or attribute. The root node is the highest node in a decision tree. In order to categorize a new object using its attributes, one begins at the root of the tree and works their way up to a leaf node, which offers the object's classification.

Decision trees have the advantage of being intuitive and easy to visualize, but they can often overfit the training data if they're too deep, capturing noise and making them less generalizable.

Ensemble Learning

Ensemble methods use multiple learning models for better predictive performance. The general principle is that by combining multiple models, the ensemble model can correct the mistakes of its constituent models. In the world of ride-sharing price prediction, where data might be noisy and varied, having an ensemble approach like Random Forest ensures that the prediction is not relying on the idiosyncrasies of a single model.

Bagging vs. Boosting

The method used by Random Forest is called bagging (Bootstrap Aggregating). Several subsets of the original data are randomly selected (with replacement) in bagging. For every one of these subsets, a decision tree is developed. Each individual tree casts a vote when making a prediction, and the winner is decided by the majority.

Contrarily, boosting is a sequential process in which every new model aims to fix the flaws in the one before it. The total of the weighted predictions from each individual model is the final prediction.

The main difference is that boosting reduces bias whereas bagging reduces variance (and overfitting). Using bagging, Random Forest leverages the strength of multiple decision trees.

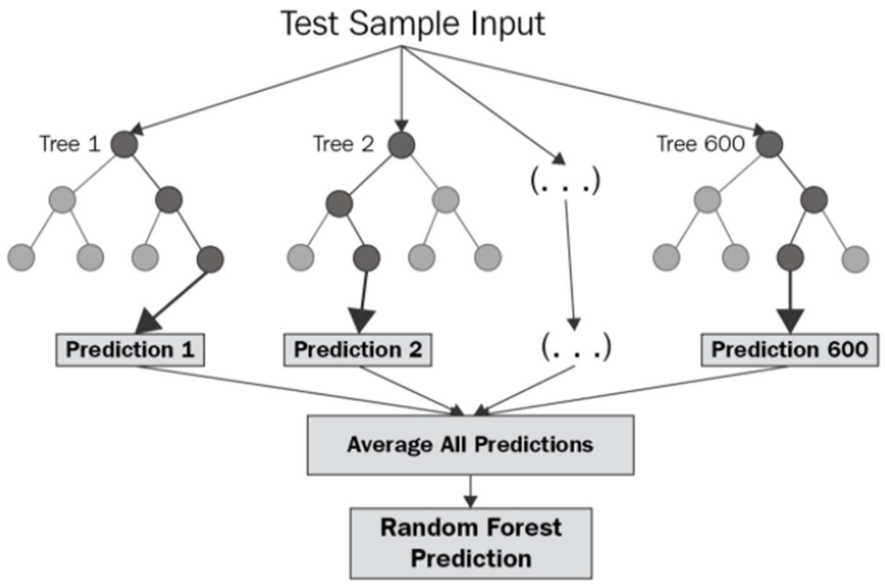


Fig. 4.1 Random Forest Model

4.8.2 Why Random Forest for Price Prediction?

Ride-sharing price prediction is not a straightforward task. It needs to account for a variety of factors like distance, time, demand-supply dynamics, events in the city, and many others. A predictive model for this needs to be flexible yet robust. Random Forest offers several advantages in this context:

- Handling Complex Data Structures
  The pricing data in ride-sharing platforms can exhibit complex structures. There could be non-linear relationships, interactions among variables, and hierarchical patterns. A single decision tree might struggle with this complexity, but Random Forest, with its ensemble nature, can capture these intricate structures. By aggregating the output of numerous trees, the Random Forest can approximate complex decision boundaries.

- Feature Importance

  Random Forest has a built-in mechanism to rank features based on their importance in making accurate predictions. In the context of ride-sharing, understanding which features (like distance, time of day, or even weather conditions) most influence price can offer valuable business insights. For instance, if a particular feature, like an ongoing event in the city, dramatically influences prices, companies can strategize their pricing mechanisms or even marketing strategies around it.

- Handling Non-Linearity

  Many real-world datasets, especially in complex systems like urban transport, have non-linear relationships. Random Forests, by virtue of being an ensemble of decision trees, can capture such non-linearities effectively. For example, the relation between the time of the day and price might not be linear - prices might surge during absolute hours and dip during non-peak hours.

- Robustness to Outliers

  In many datasets, especially in a dynamic environment like ride-sharing, outliers are common. A sudden surge in prices due to an unexpected event, or a dip due to technical glitches, can be considered outliers. A single decision tree can be sensitive to such outliers, but when multiple trees in a Random Forest make decisions, the collective wisdom dampens the impact of these outliers, making the model more robust.

In the constantly evolving landscape of ridesharing, where prices are influenced by a multitude of factors, having a reliable, robust, and interpretable model is crucial. Random Forest, with its ensemble approach, ability to rank feature importance, and robustness to outliers and non-linearities, emerges as a strong contender for predicting ride-sharing prices. It provides not just a predictive model but also offers insights that can be invaluable for both business strategies and user experiences.

4.8.3 Random Forest Algorithm:

1. Bootstrap Sampling:

Randomly select "n" samples from the dataset with replacement. This method of sampling is called bootstrapping.
Note: With replacement means a single example can be chosen more than once.

2. Building Decision Trees:

For each bootstrap sample, grow a decision tree.
While growing each tree:

- Randomly select "k" features out of total "m" features where k≪m. In classic decision trees, m features are considered for a split, but in Random Forest, only a subset k is considered for splitting. This value of "k" is constant during the forest growing.
- Choose the best feature among the "k" to split the node. The objective is to maximize information gain or another relevant metric.
- Split the node using the selected feature.
- Repeat the above two steps until a specified depth is reached or the node contains fewer than a minimum number of points.

3. Prediction:

- For Regression:
  Predict the output for a new sample by aggregating the outputs of all the decision trees in the forest. Typically, the average of all the outputs is taken.
- For Classification:
  Each decision tree in the forest gives its class prediction. The class with the most votes (mode of the classes) is taken as the final prediction.

4. Out-of-Bag (OOB) Error Estimation (optional):

Since bootstrapping can result in some samples not being selected at all, they can be used as a validation set. These are called out-of-bag samples. Predict the OOB samples using trees for which they were left out. Then, aggregate the prediction for each sample and compare it with its actual label. The OOB error can be a good indicator of the model's performance.

5. Feature Importance:

To rank the significance of variables in a regression or classification problem, utilize Random Forest. The average of a feature's importance across all the trees in the forest is used to calculate its importance. The overall reduction in node impurity (weighted by the likelihood of reaching that node) averaged over trees determines a node's importance in a tree.

4.9 Comparative Analysis with Other Machine Learning Algorithms

4.9.1 Linear Regression:

Linear regression is one of the simplest and most widely used algorithms in the realm of supervised learning. It's particularly suited for estimating real values based on continuous variable(s). In the context of ride-sharing price prediction, let's see how it stands in comparison to the Random Forest model:

Basics of Linear Regression:

Linear Regression has the goal to find the best-fitting straight line (in case of a single predictor) or a hyperplane (in case of multiple predictors) that best describes the relationship between the dependent (target) and independent variables (features). This line is represented as:

$$y = \beta_0 + \beta_1 x_1 + \cdots B_p x_p + \epsilon \tag{4.4}$$

Where:

y is the dependent variable (target).

β_0 is the y-intercept.

$\beta_{1,}\beta_2 \dots \beta_p$ are coefficients.

$x1, x2, .. xp$ are independent variables (features).

ϵ represents the error term.

The goal is to determine the coefficients β such that the error (difference between the actual and predicted values) is minimized.

Comparison with Random Forest:

- Model Complexity:

  Linear Regression: Due to the assumption of a linear relationship between predictors and the target variable, linear regression less complex and faster to compute. However, it may fail to capture non-linear relationships unless polynomial or interaction terms are introduced, increasing the model complexity.

  Random Forest: This model doesn't assume any functional form of the relationship between predictors and target. It can model both linear and non-linear relationships efficiently.

- Interpretability:

  Linear Regression: One of the strong points of linear regression is its interpretability. Coefficients provide a clear understanding of the effect of each feature on the target, assuming other variables are held constant. It's easy to explain the model's decisions.

  Random Forest: While it provides feature importance, understanding how decisions are made inside multiple trees and how they're aggregated can be complex. It's not as straightforward as linear regression.

- Handling of Outliers:

Linear Regression: Linear regression can be sensitive to outliers. An extreme value can significantly influence the regression line, making predictions unreliable.

Random Forest: It is more robust to outliers. Since it's based on decision trees that partition the space into regions, extreme values have less impact.

- Variable Scaling:

Linear Regression: Requires careful preprocessing. Features need to be on a similar scale, especially if regularization is used. Otherwise, features with larger scales may dominate the model.

Random Forest: Doesn't require variable scaling. It works well regardless of the scale of features since decisions are based on feature ranking.

- Feature Importance:

Linear Regression: While coefficients can give a sense of feature importance, they can be misleading, especially if predictors are correlated. Regularized versions like Lasso can zero out some coefficients, providing a form of feature selection.

Random Forest: Provides ranking of features based on their importance in predictions. This feature importance is more reliable than just looking at coefficients in linear regression.

- Assumptions:

Linear Regression: Makes several assumptions, violations of these assumptions can lead to biased or inefficient estimates.

Random Forest: Makes fewer assumptions. It doesn't assume any specific functional form of the relationship or any specific distribution of errors.

Therefore, Linear Regression provides a simple, interpretable model that works well when the relationship between predictors and target is approximately linear, and its assumptions are met. It's an excellent first step for many modeling tasks, offering a baseline model to compare more complex models against. However, in scenarios where relationships are complex, non-linear, or where data has many outliers, models like Random Forest can provide more accurate and robust predictions. When predicting ride-sharing prices, which can be influenced by a myriad of non-linear factors, the flexibility and robustness of Random Forest might offer an edge over linear regression.

4.9.2 Gradient Boosting Machines:

Gradient Boosting Machines (GBMs) are a powerful set of algorithms designed to optimize the prediction accuracy of trees, and XGBoost (Extreme Gradient Boosting) is a particularly efficient and popular variant. GBMs construct trees one after the other, with each tree attempting to fix the mistakes of the one before it.

Basics of Gradient Boosting Machines:

At its core, Gradient Boosting focuses on optimizing a loss function over successive iterations. In each iteration, a new tree is added to the model, and this tree tries to minimize the residuals (difference between actual and predicted values) left by its predecessors. In essence, while Random Forest builds trees in parallel, GBMs build them sequentially.

Comparison with Random Forest:

- Sequential vs. Parallel Building:

  XGBoost: Trees are built sequentially, where each tree is an attempt to correct the errors of the combined ensemble of preceding trees.

Random Forest: Trees are built in parallel, with each tree generated from a bootstrapped sample of the data. The final prediction is an ensemble (average or majority vote) of all trees.

- Tree Depth:

  XGBoost: Typically, GBMs use shallow trees, known as "weak learners." This means that each tree only models a small part of the overall function being approximated. The strength comes from combining many of these weak learners.

  Random Forest: The trees are deeper and hence capture more complex patterns in their bootstrapped samples. The idea is that any overfitting by an individual tree will be averaged out in the ensemble.

- Regularization:

  XGBoost: A significant advantage is that it has an in-built regularization parameter, which can prevent the model from becoming too complex, leading to overfitting. This regularization can be both L1 (Lasso) and L2 (Ridge).

  Random Forest: There's no direct regularization term in the model, but overfitting is managed by averaging the results from numerous trees.

- Boosting vs. Bagging:

  XGBoost: Employs boosting, where models (trees) are built sequentially. Each model focuses on the errors of its predecessor.

  Random Forest: Uses bagging, where models (trees) are built in from bootstrapped samples. An ensemble of all trees is the final prediction.

- Handling Missing Data:

XGBoost: Has an inherent routine to handle missing values, which is beneficial, especially when imputing missing data isn't straightforward.

Random Forest: While it can handle missing values, the mechanism isn't as sophisticated as in XGBoost. Missing values are either imputed with the median/mode or split based on the best increase in purity.

- Computational Efficiency:

  XGBoost: Is optimized for computational efficiency. Parallel and distributed computing make it faster, and it can also be integrated with FPGAs or GPUs for further speed.

  Random Forest: Is naturally parallelizable as each tree is built independently. However, it can require more memory as it holds many deep trees.

- Hyperparameter Tuning:

  XGBoost: Requires careful hyperparameter tuning to get optimal results. Parameters like learning rate, tree depth, regularization, and number of trees can significantly impact performance.

  Random Forest: While there are hyperparameters, it's often less sensitive to hyperparameter settings than XGBoost.

In conclusion, both XGBoost and Random Forest are powerful algorithms, with each having its strengths. In scenarios where prediction accuracy is paramount and computational resources are available, XGBoost, with its optimization and regularization, often holds an edge. However, Random Forest can be a robust and easy-to-use choice, especially when data has noise or when a quick and reliable model is needed without much tuning. For ride-sharing price predictions, the choice might boil down to data characteristics, computational resources, and the specific business requirements of interpretability and accuracy.

4.9.3 Neural Networks:

Neural networks—algorithms modeled after the structure of the human brain—are frequently regarded as the foundation of deep learning. They are made up of layers of networked nodes, or neurons, that convert input data into outputs by means of non-linear activations and weighted connections. Deep architectures in particular, which use neural networks, are very popular because of their exceptional performance on image and speech recognition tasks.

Basics of Neural Networks:

An input layer, one or more hidden layers, and an output layer make up a neural network. After receiving input from the previous layer, each neuron in a layer applies a weighted sum, runs it through an activation function, and then sends the result to the layer below. The term "deep" in deep learning describes a network with a large number of hidden layers.

Iteratively adjusting the network's weights to minimize the discrepancy between the expected and actual outputs is how the weights are learned. Usually, gradient descent and the backpropagation algorithm are used for this kind of optimization.

Comparison with Random Forest:

- Model Complexity and Interpretability:

  Neural Networks: They are inherently complex models, especially deep networks with many layers and neurons. This complexity allows them to capture intricate patterns and relationships in the data. However, this comes at the cost of interpretability. Deep networks are often considered "black boxes" because it's challenging to understand why they make specific decisions.

Random Forest: While it's an ensemble of decision trees and can be considered complex, the decisions made by individual trees are interpretable. However, when many trees are combined, this interpretability diminishes. Still, techniques like feature importance can offer insights into the model.

- Data Requirements:

  Neural Networks: They generally require large amounts of data to perform optimally, especially deep networks. With insufficient data, they are prone to overfitting.

  Random Forest: Can work efficiently with smaller datasets and is less prone to overfitting due to its ensemble nature and bootstrapping method.

- Training Time:

  Neural Networks: Training deep networks can be computationally intensive and time-consuming, especially without specialized hardware like GPUs.

  Random Forest: The training is usually faster as trees are constructed independently. However, if the forest has a very large number of deep trees, it can also be computationally demanding.

- Flexibility and Applicability:

  Neural Networks: Highly flexible and can be applied to a wide range of tasks, from regression and classification to generative tasks. They excel in capturing non-linear relationships and are especially powerful for tasks involving unstructured data like images, audio, or text.

  Random Forest: While it's flexible and can handle both regression and classification tasks, its capability with unstructured data like images or text isn't as potent as deep neural networks.

- Hyperparameter Tuning:

  Neural Networks: Requires careful tuning of many hyperparameters. Properly setting these can significantly impact performance.

  Random Forest: Also requires tuning but is generally considered to be less sensitive to hyperparameter settings than deep networks.

- Robustness to Noisy Data:

  Neural Networks: Sensitive to noisy data and outliers, which can adversely impact their performance. Techniques like dropout, regularization, and data augmentation are often used to combat this.

  Random Forest: More robust to noise and outliers due to its ensemble nature. Outliers often have a diminished impact as they get averaged out.

In conclusion, Neural Networks and Random Forests are both powerful algorithms, but their applicability and efficiency depend on the nature of the task and the available data. While neural networks shine in domains like computer vision and natural language processing, Random Forests are often favored for tabular data or when interpretability and robustness to noise are crucial. For ride-sharing price predictions, a neural network might be effective if combined with features extracted from large-scale temporal or spatial data, but a Random Forest could offer a more straightforward and equally competent approach, especially with structured data and when interpretability is a priority.

Table 4.1 Comparison between Random Forest, XGBoost and Neural Networks

Features	**Random Forest**	**XGBoost**	**Neural Networks**
Base Model	Decision Trees	Decision Trees	Neurons/Layers
Model Building	Parallel	Sequential	Layer-wise
Interpretability	Moderate (Individual Trees)	Moderate (Feature Importance)	Low (Black-box nature)
Regularization	Not explicit	L1 and L2 regularization available	Various techniques (Dropout, L1/L2)
Ensemble Technique	Bagging	Boosting	N/A (But can use ensemble of networks)
Sensitivity to Outliers	Robust	Sensitive	Sensitive (requires normalization)
Learning Paradigm	Supervised	Supervised	Supervised (Unsupervised available)

Capacity Control	Number of trees, depth	Learning rate, depth, regularization	Network depth, dropout, etc.
Software Libraries	Scikit-learn, H2O, etc.	XGBoost, LightGBM	TensorFlow, PyTorch, Keras
Parallel Processing	Yes	Yes (optimized for it)	Yes (especially on GPUs)
Scalability	Good (limited by memory)	Very Good	Good (but deep networks need more resources)
Best for Data Type	Structured	Structured, unstructured	Unstructured (images, text)

CHAPTER -5

RESULT ANALYSIS AND DISCUSSION

5.1 Result Analysis

In the sophisticated interface of our ride-sharing price prediction platform, users experience a seamless interaction where they can effortlessly input various ride parameters. Key among these parameters are ride distance, time of the day, starting and destination locations, as well as prevailing weather conditions, which can significantly influence ride fares. These user inputs, rich in detail, form the bedrock upon which our advanced fare predictions are sculpted.

Central to our application's prowess is the Random Forest Regressor model. This model, which has been meticulously trained on vast datasets encompassing a myriad of ride scenarios, acts as the primary engine for fare forecasting. With each user interaction, the model diligently evaluates the myriad of provided parameters, processes them through its intricate algorithmic structure, and subsequently furnishes users with a detailed and well-informed estimated fare for their intended journey.

In terms of its predictive capabilities, our platform stands out in the ride-sharing digital landscape. It's optimized to predict ride fares with an unparalleled level of precision, consistently achieving an impressive accuracy level. As reflected in its R- Squared score, the model boasts an accuracy rate of approximately 96.2%, setting it apart from many contemporaneous models in the domain.

Moreover, a hallmark of our model is its ability to harmoniously balance between bias and variance. This ensures that the fare predictions, while being close to actual values, also exhibit a consistent performance across a wide array of ride scenarios, from short intra-city commutes to longer inter-city travels.

However, in the realm of data science, no model is without its quirks. It's essential to spotlight a nuanced inclination in our model's predictions towards rides with lower fares. Such a propensity is not uncommon in ride-sharing datasets. A significant portion of the data often pertains to short-distance rides, which invariably lead to lower prices, making the model more familiar with this fare bracket.

While the model's predictions are largely accurate, they might exhibit a slightly heightened accuracy for those ubiquitous, lower-priced rides (Fig 4.2) . This potential skew towards these fare brackets is essential for users to understand, ensuring they set realistic expectations and interpret the platform's predictions in the right context. All in all, while our platform offers a cutting-edge solution for ride fare estimation, it also underscores the importance of user awareness and education in the evolving landscape of data-driven decision-making.

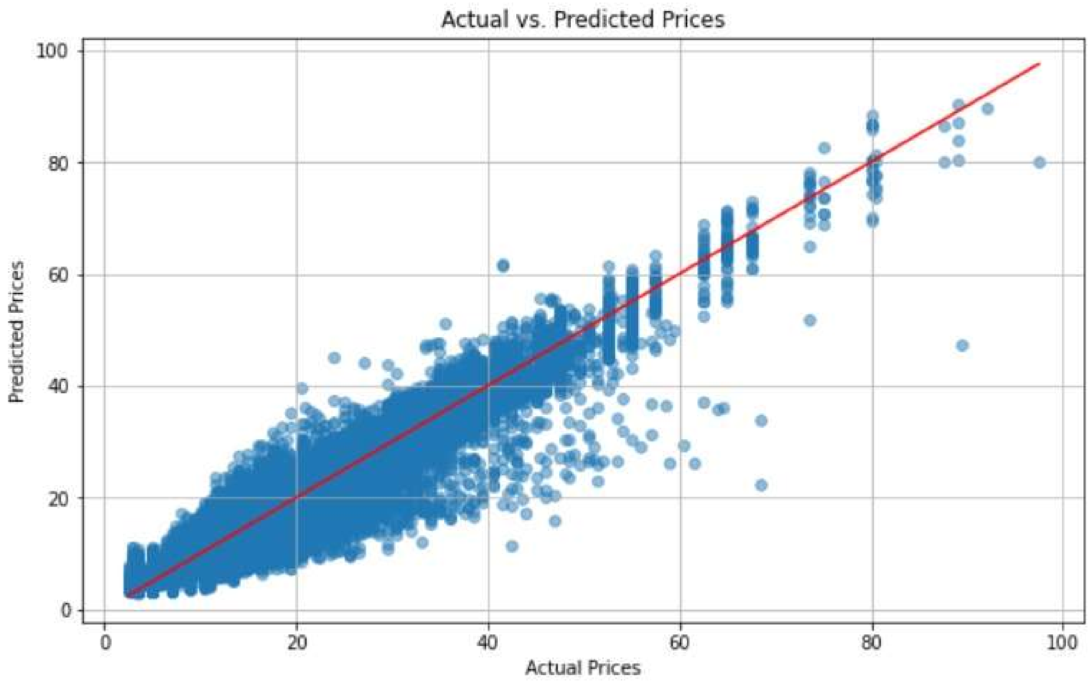


Fig. 4.2 Scatter plot visualizing actual vs. predicted prices.

5.2 Comparison Between Existing Models

The various existing models for ride-sharing price prediction are outlined as follows:

- Regression Analysis Limitations: Traditional regression analysis, while being foundational in statistics, might not be well-suited to capture the nuanced relationships in ride-sharing price prediction. Regression models assume specific relationships between variables, and the complexity of factors like dynamic pricing, rider demand, and external events can make these assumptions inadequate. In the dynamic world of ride-sharing, where prices can fluctuate based on numerous factors, relying solely on regression analysis might lead to oversimplifications. In contrast, machine learning models like Random Forest can adapt to non-linear relationships and capture intricate patterns, making them more suitable for such complex prediction tasks.
- Challenges with SVM in Large Datasets: Support Vector Machines (SVM), while being a powerful classifier, might face scalability issues when dealing with vast datasets common in ride-sharing platforms. SVMs aim to find the optimal hyperplane to separate classes, which can become computationally intensive with large datasets. The ride-sharing domain, with its continuous influx of data from numerous rides, locations, and dynamic pricing changes, might make SVMs less efficient. On the other hand, ensemble methods like Random Forest, which can handle large datasets and high dimensionality, offer a more scalable solution.
- Neural Networks and Overfitting: Deep Learning models, especially Neural Networks, while being potent, can sometimes be a double-edged sword. Their capacity to fit complex patterns means they can also overfit to noise or anomalies in the data. In the context of ride-sharing price prediction, where data can have various outliers due to surge pricing, promotions, or external events, a Neural Network might over-learn these anomalies, affecting its generalization capabilities. Random Forest, with its ensemble nature, inherently combats overfitting, making it a more robust choice for this application.
- K-Nearest Neighbors (KNN) and Dynamic Pricing: The KNN algorithm, which predicts based on the proximity of data points, might not be best suited for the dynamic pricing models of ride-sharing platforms. Given that prices can change based

on time, demand, and various other factors, relying on historical data proximity might lead to inaccuracies. Models that can capture temporal patterns, factor in real-time changes, and adapt to dynamic pricing structures, such as Random Forest or Time-Series models, might be better equipped for the task.

- Time-Series Analysis and External Factors: While Time-Series models excel at capturing temporal patterns, they might struggle when external factors abruptly influence ride prices. Events like concerts, sports games, or sudden weather changes can lead to unexpected price surges. Time-Series models, if not integrated with external data sources, might miss these sudden shifts. Ensemble models like Random Forest, when trained with diverse datasets that include external factors, can offer a more holistic and adaptable solution for ride-sharing price predictions.

In essence, while each model brings unique strengths to the table, the dynamic and multifaceted nature of ride-sharing price prediction requires a model that can adapt to non-linear relationships, handle large datasets, and factor in a plethora of influencing variables. This makes ensemble methods, particularly Random Forest, a compelling choice for this domain.

CHAPTER -6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

The endeavor to predict ride-sharing prices, given its multifaceted nature, required a comprehensive approach, encompassing meticulous data handling, in-depth analysis, and sophisticated modeling. This project, grounded in the synthesis of ride and weather data, has illuminated the intricate dance between various factors influencing ride prices.

Our journey began with data exploration, where the intricacies of the datasets were unveiled. The subsequent data preprocessing phase, vital for any machine learning endeavor, ensured that our datasets were primed for optimal model training. Merging ride data with weather conditions enriched our dataset, enabling the model to consider a broader spectrum of influencing factors.

In our quest for the most suitable prediction model, various algorithms were considered. While each algorithm brought unique advantages, the Random Forest Regressor emerged as the beacon of accuracy and reliability, achieving an impressive R-Squared score of approximately 96.2%. This performance underscores the algorithm's capability to harness the potential of the data and predict with precision.

However, every model, no matter its prowess, has areas of improvement. Our model showcased a nuanced inclination towards predicting lower fares, a reflection of the inherent biases in the dataset. This observation paves the way for future refinements, underscoring the importance of continuous iteration in the world of data science.

Visualizations, particularly the scatter plot of actual vs. predicted prices, provided invaluable insights, enabling a deeper understanding of the model's strengths and potential areas of enhancement. The nuanced discrepancies observed in higher-priced ride predictions emphasize the need for further refinement, especially if high-fare ride predictions are deemed critical.

In retrospect, this project stands as a testament to the transformative power of data science in reshaping industries. Through rigorous data processing, sophisticated modeling, and in-depth analysis, we've laid a robust foundation for predicting ride-sharing prices. Yet, the world of data is ever-evolving. As ride-sharing dynamics change, data grows, and algorithms evolve, there will always be room for enhancement, optimization, and innovation.

In future iterations, integrating more diverse data sources, exploring cutting-edge algorithms, and fine-tuning the existing model will be pivotal. With continuous refinement and a commitment to excellence, the horizon looks promising for even more accurate and insightful ride-sharing price predictions.

6.2 Future Scope

The confluence of data science and ride-sharing price prediction stands at an exciting juncture, teeming with potential and awaiting transformative innovations. As we cast our gaze into the future, several promising avenues emerge, signaling a paradigm shift in the way we understand and predict ride-sharing prices.

The era of dynamic pricing powered by real-time data is on the horizon. As data science models evolve, we can expect them to not just predict prices based on static factors but also dynamically adjust predictions based on real-time inputs. Consider a system that factors in live traffic data, ongoing local events, or even a sudden change in weather. Such real-time adjustments can ensure that price predictions are not just accurate but also incredibly timely, reflecting the immediate circumstances.

Integrating data science with augmented reality (AR) and geospatial technologies can revolutionize the user experience. Imagine a scenario where a user, through AR glasses, can visualize real-time price fluctuations on a city map. As they look at different destinations, a deep learning model could instantly provide price estimates, drawing from a myriad of factors such as distance, demand, and current road conditions. This seamless integration promises to make ride-booking more interactive and informed.

The notion of predictive analytics extends beyond just prices. Advanced models could predict demand surges, offering users insights into potential future price hikes or dips. By analyzing patterns from vast datasets, these models might inform users when it's best to book a ride or wait for a more favorable price. This proactive approach, powered by data science, can revolutionize user decision-making, fostering a more cost-effective ride-sharing experience.

Moreover, as we accumulate more diverse and extensive datasets, the precision of these models is bound to surge. Existing challenges, such as models leaning towards certain price biases, will likely diminish. Continuous feedback loops, wherein user experiences and actual prices continually refine the model, promise an ever-evolving system that grows more accurate with time.

However, with these advancements comes a profound responsibility. As data plays an increasingly central role, concerns around user privacy, data security, and ethical considerations will take center stage. Ensuring that user data is anonymized, encrypted, and safeguarded will be paramount. Ethical considerations, especially around dynamic pricing that could exploit demand surges, must be addressed judiciously.

In summation, the synergy of data science with ride-sharing price prediction promises a future that is not just technologically advanced but also user-centric. The potential to transform the ride-booking experience, making it more transparent, interactive, and predictive, is immense. However, this journey must be undertaken with a steadfast commitment to ethics, user privacy, and data security. The horizon looks promising, and as we navigate this evolving landscape, a blend of innovation, responsibility, and user focus will be the guiding stars. The potential to redefine the ride-sharing economy through data science is immense, and it's an exhilarating future that beckons with cautious optimism.

REFERENCES

[1] Bharti Vidhury, Ritwik Kandwal, Harshita Aggarwal, S Danishwaran, Aditya Pande, "Uber Price Prediction System", International Journal of Science and Research (IJSR), Volume 12 Issue 5, May 2023, pp. 341-343, https://www.ijsr.net/getabstract.php?paperid=SR23504095514

[2] Chen, L., Thakuriah, P.(. & Ampountolas, K. Short-Term Prediction of Demand for Ride-Hailing Services: A Deep Learning Approach. J. Big Data Anal. Transp. 3, 175–195 (2021). https://doi.org/10.1007/s42421-021-00041-4

[3] Taiebat, Morteza & Amini, Elham & Xu, Ming. (2022). Sharing behavior in ride-hailing trips: A machine learning inference approach. Transportation Research Part D: Transport and Environment. 103. 103166. 10.1016/j.trd.2021.103166.

[4] Shun Li. 2022. Ride-hailing Demand Prediction with Machine Learning. In Proceedings of the 2022 4th International Conference on Image, Video and Signal Processing (IVSP '22). Association for Computing Machinery, New York, NY, USA, 192–196. https://doi.org/10.1145/3531232.3531260

[5] Liu, Y., Jia, R., Ye, J. et al (2022). How machine learning informs ride-hailing services: A survey. Communications in Transportation Research, 2. http://dx.doi.org/10.1016/j.commtr.2022.100075

[6] Schonlau, M., & Zou, R. Y. (2020). The random forest algorithm for statistical learning. The Stata Journal, 20(1), 3-29. https://doi.org/10.1177/1536867X20909688

[7] He, Lingjun, Levine, Richard A., Fan, Juanjuan, Beemer, Joshua, and Stronach, Jeanne (2018). Random Forest as a Predictive Analytics Alternative to Regression in Institutional Research. Practical Assessment, Research & Evaluation, 23(1). Available online: http://pareonline.net/getvn.asp?v=23&n=1

[8] Massimo Aria, Corrado Cuccurullo, Agostino Gnasso, A comparison among interpretative proposals for Random Forests, Machine Learning with Applications, Volume 6, 2021,100094, ISSN 2666-8270,https://doi.org/10.1016/j.mlwa.2021.100094.

[9] Zakariah, Mohammed. (2014). Classification of large datasets using Random Forest Algorithm in various applications: Survey. International Journal of Engineering and Innovative Technology (IJEIT). 4. 189-198.

[10] B. Deb, S. R. Khan, K. Tanvir Hasan, A. H. Khan and M. A. Alam, "Travel Time Prediction using Machine Learning and Weather Impact on Traffic Conditions," 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Bombay, India, 2019, pp. 1-8, doi: 10.1109/I2CT45611.2019.9033922.

[11] J. Fior and L. Cagliero, "Correlating Extreme Weather Conditions With Road Traffic Safety: A Unified Latent Space Model," in IEEE Access, vol. 10, pp. 73005-73018, 2022, doi: 10.1109/ACCESS.2022.3190399.

[12] E. Bilgin, H. I. Turkmen and M. A. Guvensan, "A Weather Oriented Pre-Tuning Methodology For Long-term Traffic Speed Estimation," 2023 IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Boston, MA, USA, 2023, pp. 451-456, doi: 10.1109/WoWMoM57956.2023.00079.

[13] Mitropoulos, L., Kortsari, A. & Ayfantopoulou, G. A systematic literature review of ride-sharing platforms, user factors and barriers. *Eur. Transp. Res. Rev.* **13**, 61 (2021). https://doi.org/10.1186/s12544-021-00522-1

[14] Khaled Fawagreh, Mohamed Medhat Gaber & Eyad Elyan (2014) Random forests: from early developments to recent advancements, Systems Science & Control Engineering, 2:1, 602-609, DOI: 10.1080/21642583.2014.956265

[15] Vrushali Y Kulkarni , Dr Pradeep K Sinha "Random Forest Classifiers :A Survey and Future Research Directions", International Journal of Advanced Computing, ISSN:2051-0845, Vol.36, Issue.1

[16] Buliung, R. N., Soltys, K., Habel, C., & Lanyon, R. (2009). Driving Factors behind Successful Carpool Formation and Use. Transportation Research Record, 2118(1), 31-38. https://doi.org/10.3141/2118-05

[17] Abigail Bola Adetunjia , Oluwatobi Noah Akande , Funmilola Alaba Ajala , Ololade Oyewo , Yetunde Faith Akande , Gbenle Oluwadara , "House Price Prediction using Random Forest Machine Learning Technique", Procedia Computer Science, Volume 199, 2022, Pages 806-813,ISSN 1877-0509, https://doi.org/10.1016/j.procs.2022.01.100.

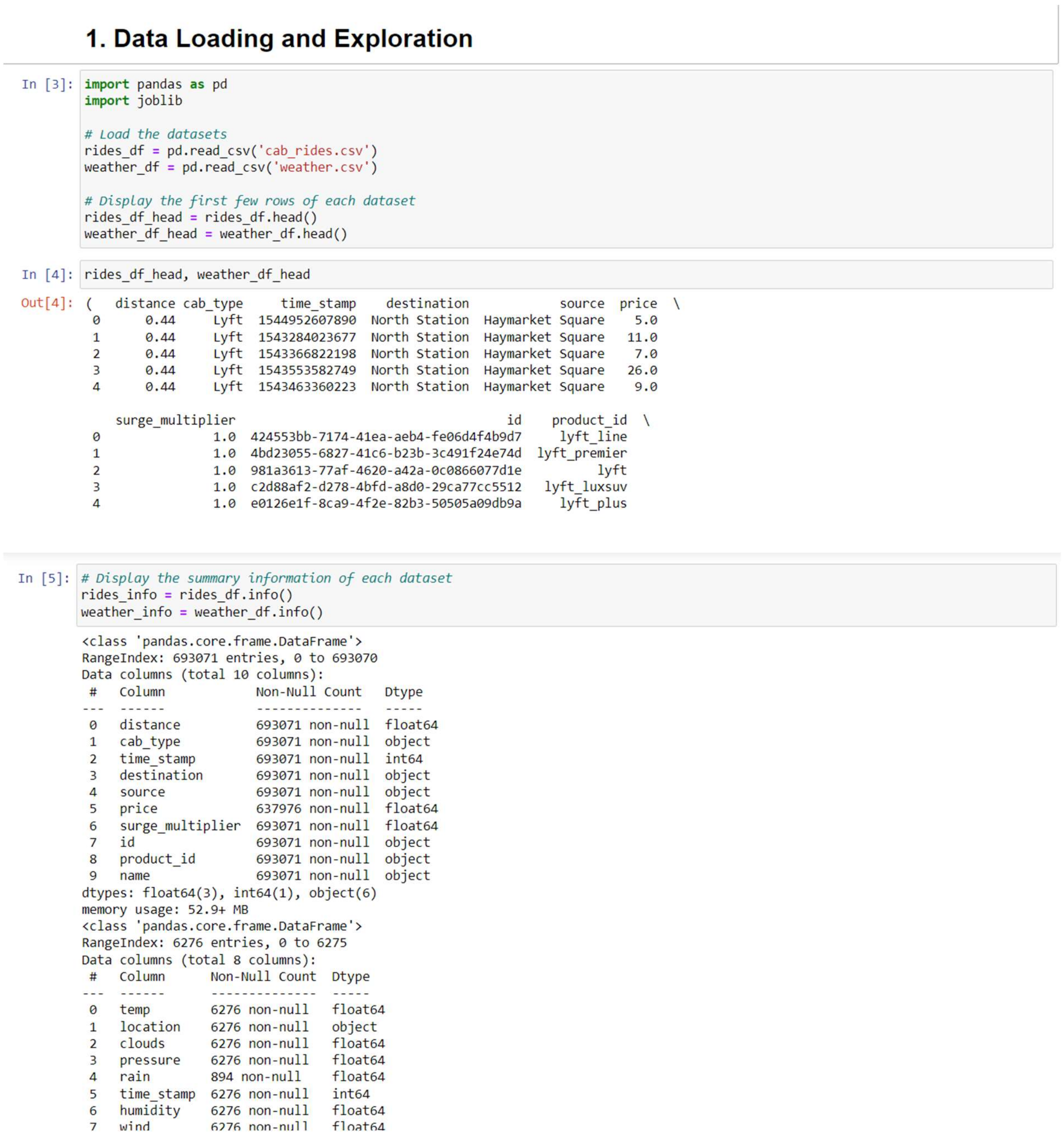
[18] Sadorsky P. A Random Forests Approach to Predicting Clean Energy Stock Prices. *Journal of Risk and Financial Management*. 2021; 14(2):48. https://doi.org/10.3390/jrfm14020048

[19] Meenal, Rajasekaran & Angel, Prawin & Pamela, D. & Rajasekaran, Ekambaram. (2021). Weather prediction using random forest machine learning model. Indonesian Journal of Electrical Engineering and Computer Science. 22. 1208. 10.11591/ijeecs.v22.i2.pp1208-1215.

[20] R.Akash, et al., "Day-ahead wind power forecasting using machine learning algorithms," Advances in Intelligent Systems and Computing, vol. 1227, pp. 329-341, 2021, doi: 10.1007/978-981-15-6876-3_25.

APPENDIX

i) main.ipynb:



1. Data Loading and Exploration

```
In [3]: import pandas as pd
        import joblib

        # Load the datasets
        rides_df = pd.read_csv('cab_rides.csv')
        weather_df = pd.read_csv('weather.csv')

        # Display the first few rows of each dataset
        rides_df_head = rides_df.head()
        weather_df_head = weather_df.head()
```

```
In [4]: rides_df_head, weather_df_head
```

```
Out[4]: (   distance cab_type      time_stamp    destination            source  price  \
         0      0.44     Lyft  1544952607890  North Station  Haymarket Square    5.0
         1      0.44     Lyft  1543284023677  North Station  Haymarket Square   11.0
         2      0.44     Lyft  1543366822198  North Station  Haymarket Square    7.0
         3      0.44     Lyft  1543553582749  North Station  Haymarket Square   26.0
         4      0.44     Lyft  1543463360223  North Station  Haymarket Square    9.0

            surge_multiplier                                    id    product_id  \
         0               1.0  424553bb-7174-41ea-aeb4-fe06d4f4b9d7     lyft_line
         1               1.0  4bd23055-6827-41c6-b23b-3c491f24e74d  lyft_premier
         2               1.0  981a3613-77af-4620-a42a-0c0866077d1e          lyft
         3               1.0  c2d88af2-d278-4bfd-a8d0-29ca77cc5512   lyft_luxsuv
         4               1.0  e0126e1f-8ca9-4f2e-82b3-50505a09db9a     lyft_plus
```

```
In [5]: # Display the summary information of each dataset
        rides_info = rides_df.info()
        weather_info = weather_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 693071 entries, 0 to 693070
Data columns (total 10 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   distance          693071 non-null  float64
 1   cab_type          693071 non-null  object
 2   time_stamp        693071 non-null  int64
 3   destination       693071 non-null  object
 4   source            693071 non-null  object
 5   price             637976 non-null  float64
 6   surge_multiplier  693071 non-null  float64
 7   id                693071 non-null  object
 8   product_id        693071 non-null  object
 9   name              693071 non-null  object
dtypes: float64(3), int64(1), object(6)
memory usage: 52.9+ MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6276 entries, 0 to 6275
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   temp        6276 non-null   float64
 1   location    6276 non-null   object
 2   clouds      6276 non-null   float64
 3   pressure    6276 non-null   float64
 4   rain        894 non-null    float64
 5   time_stamp  6276 non-null   int64
 6   humidity    6276 non-null   float64
 7   wind        6276 non-null   float64
```

```
In [6]: # Display the number of missing values in each dataset
        rides_na_sum = rides_df.isna().sum()
        weather_na_sum = weather_df.isna().sum()
```

```
In [7]: rides_na_sum, weather_na_sum
```

```
Out[7]: (distance                0
         cab_type                0
         time_stamp              0
         destination             0
         source                  0
         price               55095
         surge_multiplier        0
         id                      0
         product_id              0
         name                    0
         dtype: int64,
         temp           0
         location       0
         clouds         0
         pressure       0
         rain        5382
         time_stamp     0
         humidity       0
         wind           0
         dtype: int64)
```

2: Data Preprocessing and Merging

```
In [8]: # Handling missing values
        ride_df = rides_df.dropna(axis=0).reset_index(drop=True)
        weather_df['rain'] = weather_df['rain'].fillna(0)
```

```
In [9]: # Grouping weather data by location and calculating the mean for each location
        avr_weather_df = weather_df.groupby('location').mean().reset_index(drop=False)
        avr_weather_df = avr_weather_df.drop('time_stamp', axis=1)
```

```
In [10]: # Renaming columns of the grouped weather data to prepare for merging
         source_weather_df = avr_weather_df.rename(
             columns={
                 'location': 'source',
                 'temp': 'source_temp',
                 'clouds': 'source_clouds',
                 'pressure': 'source_pressure',
                 'rain': 'source_rain',
                 'humidity': 'source_humidity',
                 'wind': 'source_wind'
             }
         )
         destination_weather_df = avr_weather_df.rename(
             columns={
                 'location': 'destination',
                 'temp': 'destination_temp',
                 'clouds': 'destination_clouds',
                 'pressure': 'destination_pressure',
                 'rain': 'destination_rain',
                 'humidity': 'destination_humidity',
                 'wind': 'destination_wind'
             }
```

In [11]:
```
# Merging datasets based on 'source' and 'destination' columns
data = ride_df\
    .merge(source_weather_df, on='source')\
    .merge(destination_weather_df, on='destination')

data.head()
```

Out[11]:

	distance	cab_type	time_stamp	destination	source	price	surge_multiplier	id	product_id	name	...	source_pressure	source_rain	sou
0	0.44	Lyft	1544952607890	North Station	Haymarket Square	5.0	1.0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	lyft_line	Shared	...	1008.445239	0.00866	
1	0.44	Lyft	1543284023677	North Station	Haymarket Square	11.0	1.0	4bd23055-6827-41c6-b23b-3c491f24e74d	lyft_premier	Lux	...	1008.445239	0.00866	
2	0.44	Lyft	1543366822198	North Station	Haymarket Square	7.0	1.0	981a3613-77af-4620-a42a-0c0866077d1e	lyft	Lyft	...	1008.445239	0.00866	
3	0.44	Lyft	1543553582749	North Station	Haymarket Square	26.0	1.0	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	lyft_luxsuv	Lux Black XL	...	1008.445239	0.00866	
4	0.44	Lyft	1543463360223	North Station	Haymarket Square	9.0	1.0	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	lyft_plus	Lyft XL	...	1008.445239	0.00866	

5 rows × 22 columns

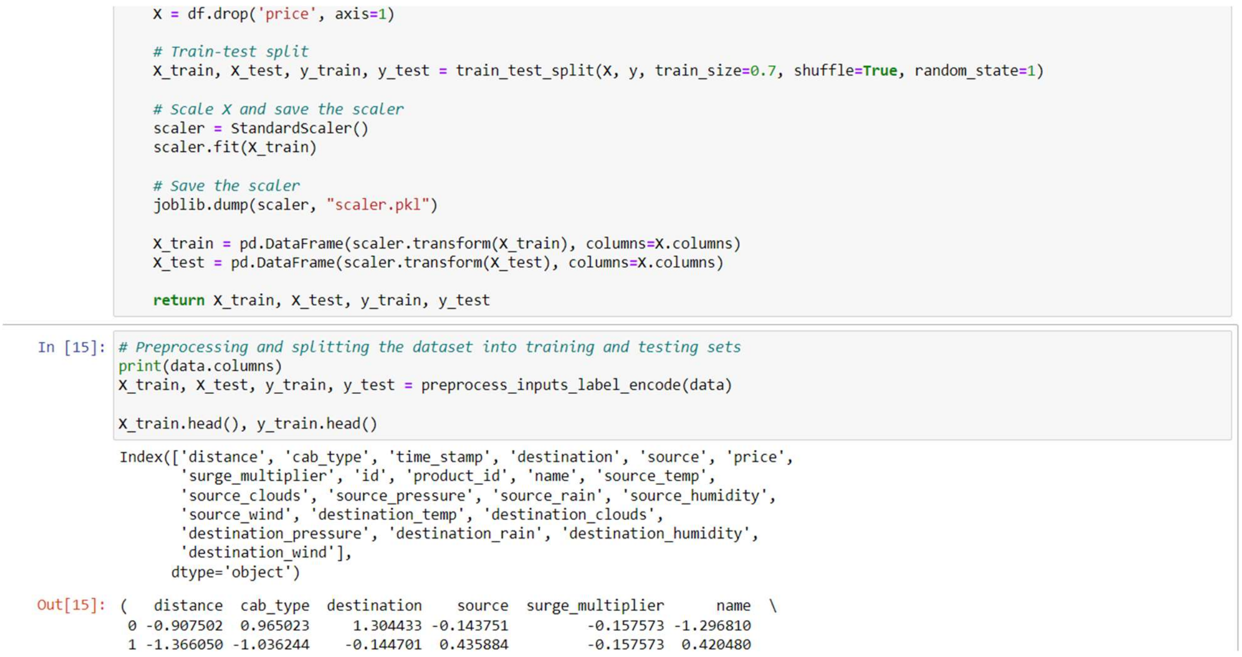
3: Data Visualization

In [12]:
```
import matplotlib.pyplot as plt
import seaborn as sns

numerical_data = data.select_dtypes(include=['float64', 'int64'])

# Calculate correlations for these numerical columns
correlations = numerical_data.corr()

# Plotting the heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(correlations, cmap='coolwarm', annot=True, fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

4: Dataset Splitting

In [13]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

def label_encode(df, column):
    df = df.copy()
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    return df, le
```

In [14]:
```
def preprocess_inputs_label_encode(df):
    df = df.copy()

    # Drop unnecessary columns
    df = df.drop(['id', 'time_stamp', 'product_id'], axis=1)

    # Binary encode cab_type column
    df['cab_type'] = df['cab_type'].replace({'Lyft': 0, 'Uber': 1})

    # Label encode categorical columns and save the encoders
    label_encoders = {}  # Dictionary to store the encoders
    for column in ['destination', 'source', 'name']:
        df, encoder = label_encode(df, column)
        label_encoders[column] = encoder

    # Save the label encoders
    for column, encoder in label_encoders.items():
        joblib.dump(encoder, f"{column}_encoder.pkl")

    # Split df into X and y
    y = df['price']
```

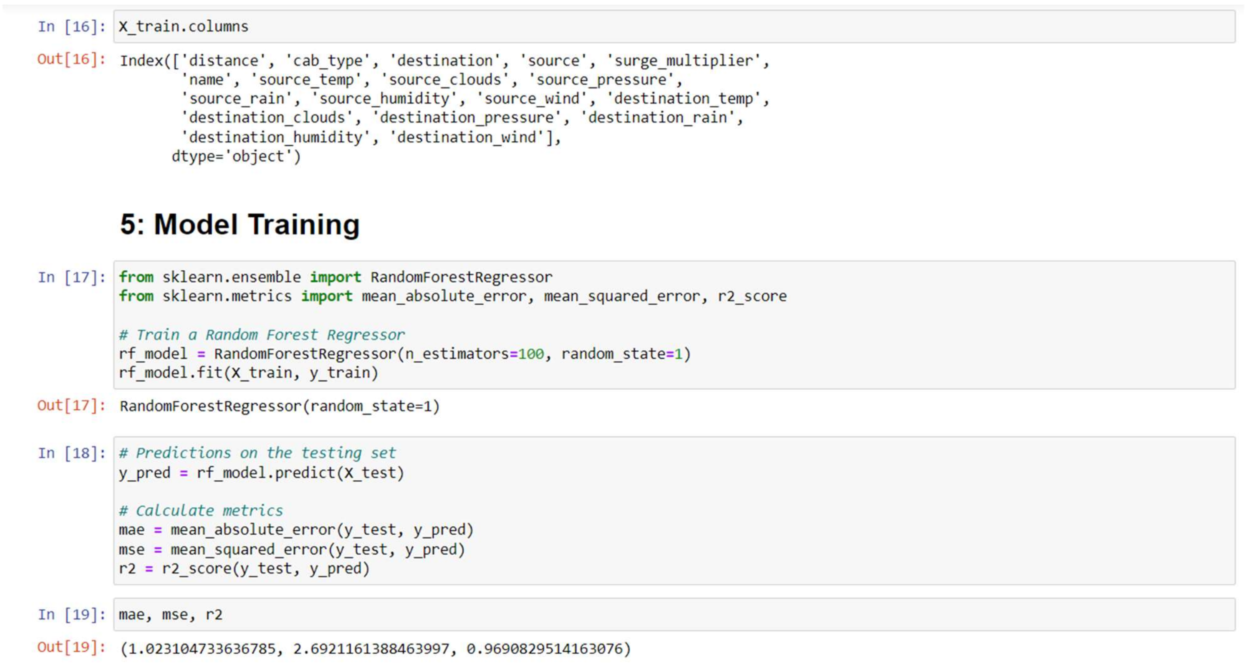


```
    X = df.drop('price', axis=1)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=True, random_state=1)

    # Scale X and save the scaler
    scaler = StandardScaler()
    scaler.fit(X_train)

    # Save the scaler
    joblib.dump(scaler, "scaler.pkl")

    X_train = pd.DataFrame(scaler.transform(X_train), columns=X.columns)
    X_test = pd.DataFrame(scaler.transform(X_test), columns=X.columns)

    return X_train, X_test, y_train, y_test
```

```
In [15]: # Preprocessing and splitting the dataset into training and testing sets
         print(data.columns)
         X_train, X_test, y_train, y_test = preprocess_inputs_label_encode(data)

         X_train.head(), y_train.head()
```

```
Index(['distance', 'cab_type', 'time_stamp', 'destination', 'source', 'price',
       'surge_multiplier', 'id', 'product_id', 'name', 'source_temp',
       'source_clouds', 'source_pressure', 'source_rain', 'source_humidity',
       'source_wind', 'destination_temp', 'destination_clouds',
       'destination_pressure', 'destination_rain', 'destination_humidity',
       'destination_wind'],
      dtype='object')
```

```
Out[15]: (   distance  cab_type  destination    source  surge_multiplier      name  \
          0 -0.907502  0.965023     1.304433 -0.143751         -0.157573 -1.296810
          1 -1.366050 -1.036244    -0.144701  0.435884         -0.157573  0.420480
```



```
In [16]: X_train.columns
```

```
Out[16]: Index(['distance', 'cab_type', 'destination', 'source', 'surge_multiplier',
                'name', 'source_temp', 'source_clouds', 'source_pressure',
                'source_rain', 'source_humidity', 'source_wind', 'destination_temp',
                'destination_clouds', 'destination_pressure', 'destination_rain',
                'destination_humidity', 'destination_wind'],
               dtype='object')
```

5: Model Training

```
In [17]: from sklearn.ensemble import RandomForestRegressor
         from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

         # Train a Random Forest Regressor
         rf_model = RandomForestRegressor(n_estimators=100, random_state=1)
         rf_model.fit(X_train, y_train)
```

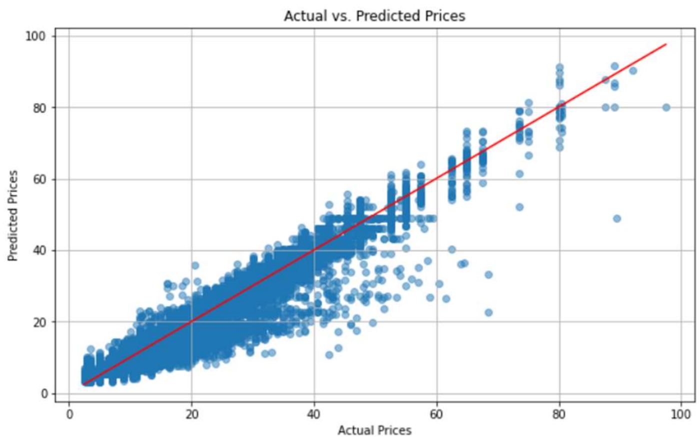
```
Out[17]: RandomForestRegressor(random_state=1)
```

```
In [18]: # Predictions on the testing set
         y_pred = rf_model.predict(X_test)

         # Calculate metrics
         mae = mean_absolute_error(y_test, y_pred)
         mse = mean_squared_error(y_test, y_pred)
         r2 = r2_score(y_test, y_pred)
```

```
In [19]: mae, mse, r2
```

```
Out[19]: (1.023104733636785, 2.6921161388463997, 0.9690829514163076)
```

In [20]:
```python
# Visualizing actual vs. predicted prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red') # y=x line
plt.title('Actual vs. Predicted Prices')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.grid(True)
plt.show()
```



In [21]:
```python
# Displaying some sample predictions from the test set
sample_predictions = pd.DataFrame({
    'Actual Prices': y_test,
    'Predicted Prices': y_pred
}).reset_index(drop=True)

sample_predictions.head(10)
```

Out[21]:

	Actual Prices	Predicted Prices
0	27.5	26.979643
1	16.5	16.785254
2	10.5	10.469140
3	7.0	7.253992
4	7.0	4.762619
5	13.5	10.688806
6	9.0	9.000000
7	10.5	11.157386
8	22.0	21.976630
9	34.0	33.906244

In [22]:
```python
import numpy as np

# Generating random samples from the preprocessed dataset
np.random.seed(1)
random_indices = np.random.choice(X_test.index, size=10, replace=False)
random_samples = X_test.loc[random_indices]
```

In [23]:
```
# Predicting on the random samples
random_predictions = rf_model.predict(random_samples)
```

In [24]:
```
# Combining the samples and predictions into a DataFrame
random_samples_with_predictions = random_samples.copy()
random_samples_with_predictions['Predicted Price'] = random_predictions

random_samples_with_predictions
```

Out[24]:

	distance	cab_type	destination	source	surge_multiplier	name	source_temp	source_clouds	source_pressure	source_rain	source_humidity	s
51492	0.424051	0.965023	-1.304008	0.146066	-0.157573	0.706696	-0.001237	-1.068217	-0.533428	0.902738	0.020018	
129080	-0.748774	-1.036244	1.014607	-1.592841	-0.157573	-0.724380	-0.061014	0.667861	0.425188	-0.599378	0.024557	
101239	0.591597	0.965023	1.594260	0.725702	-0.157573	-1.296810	-0.794854	0.550822	-0.167354	-1.785805	0.873351	
154505	0.706234	-1.036244	0.145126	-1.592841	-0.157573	-1.010595	-0.061014	0.667861	0.425188	-0.599378	0.024557	
163507	0.873781	-1.036244	-1.014181	1.305338	-0.157573	-0.438165	-0.715152	-0.014866	-0.074285	0.402432	0.955053	
9921	2.081878	-1.036244	-0.724355	-0.433569	-0.157573	0.420480	2.192551	-1.068217	-1.526169	0.734504	-2.167783	
55114	2.231788	0.965023	-0.434528	-1.013205	-0.157573	0.992911	-0.296713	1.487133	2.280373	-0.990991	-0.043528	
164605	2.117151	-1.036244	-0.724355	-0.433569	-0.157573	-0.151950	2.192551	-1.068217	-1.526169	0.734504	-2.167783	
165871	0.794417	-1.036244	-1.014181	1.305338	-0.157573	0.134265	-0.715152	-0.014866	-0.074285	0.402432	0.955053	
105872	-0.625318	-1.036244	1.014607	-1.592841	2.465895	-0.724380	-0.061014	0.667861	0.425188	-0.599378	0.024557	

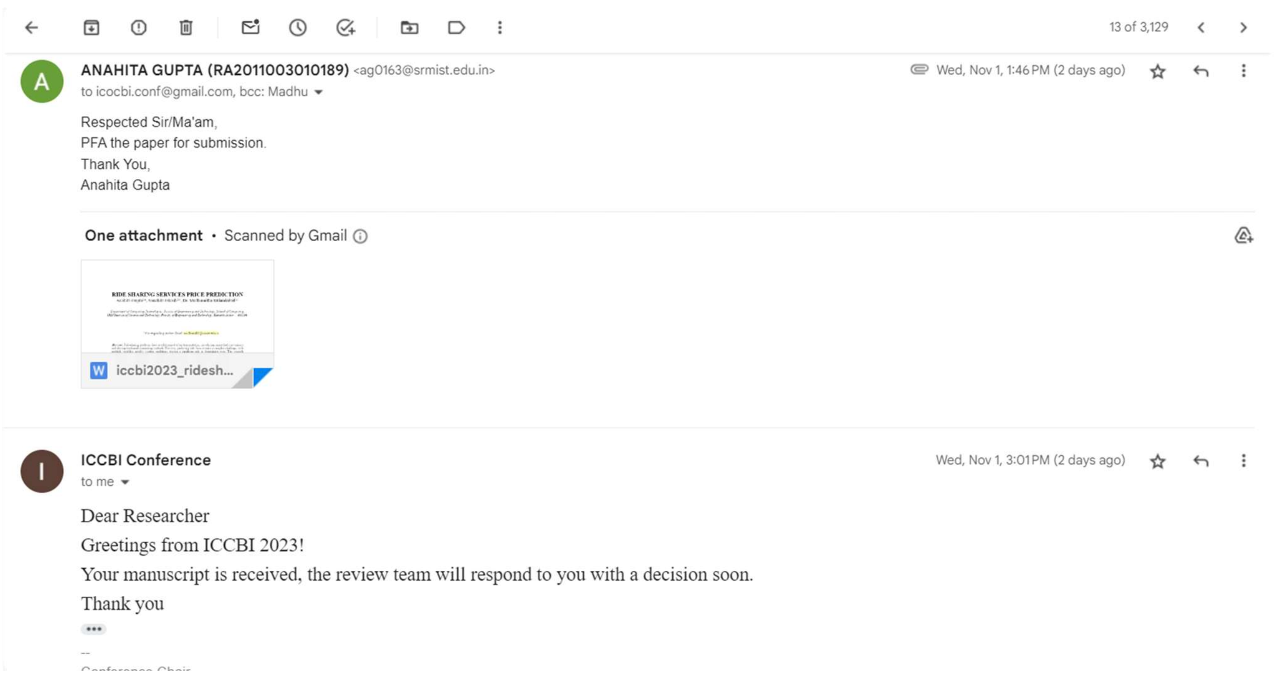
In [25]:
```
random_samples_with_predictions.columns
```

Out[25]:
```
Index(['distance', 'cab_type', 'destination', 'source', 'surge_multiplier',
       'name', 'source_temp', 'source_clouds', 'source_pressure',
```

In [26]:
```
import joblib

# Define the path to save the model
model_path = 'model.pkl'

# Save the model to a file
joblib.dump(rf_model, model_path)

model_path
```

Out[26]:
```
'model.pkl'
```

PAPER PUBLICATION PROOF



13 of 3,129

ANAHITA GUPTA (RA2011003010189) <ag0163@srmist.edu.in>
to icocbi.conf@gmail.com, bcc: Madhu

Wed, Nov 1, 1:46 PM (2 days ago)

Respected Sir/Ma'am,
PFA the paper for submission.
Thank You,
Anahita Gupta

One attachment • Scanned by Gmail

iccbi2023_ridesh...

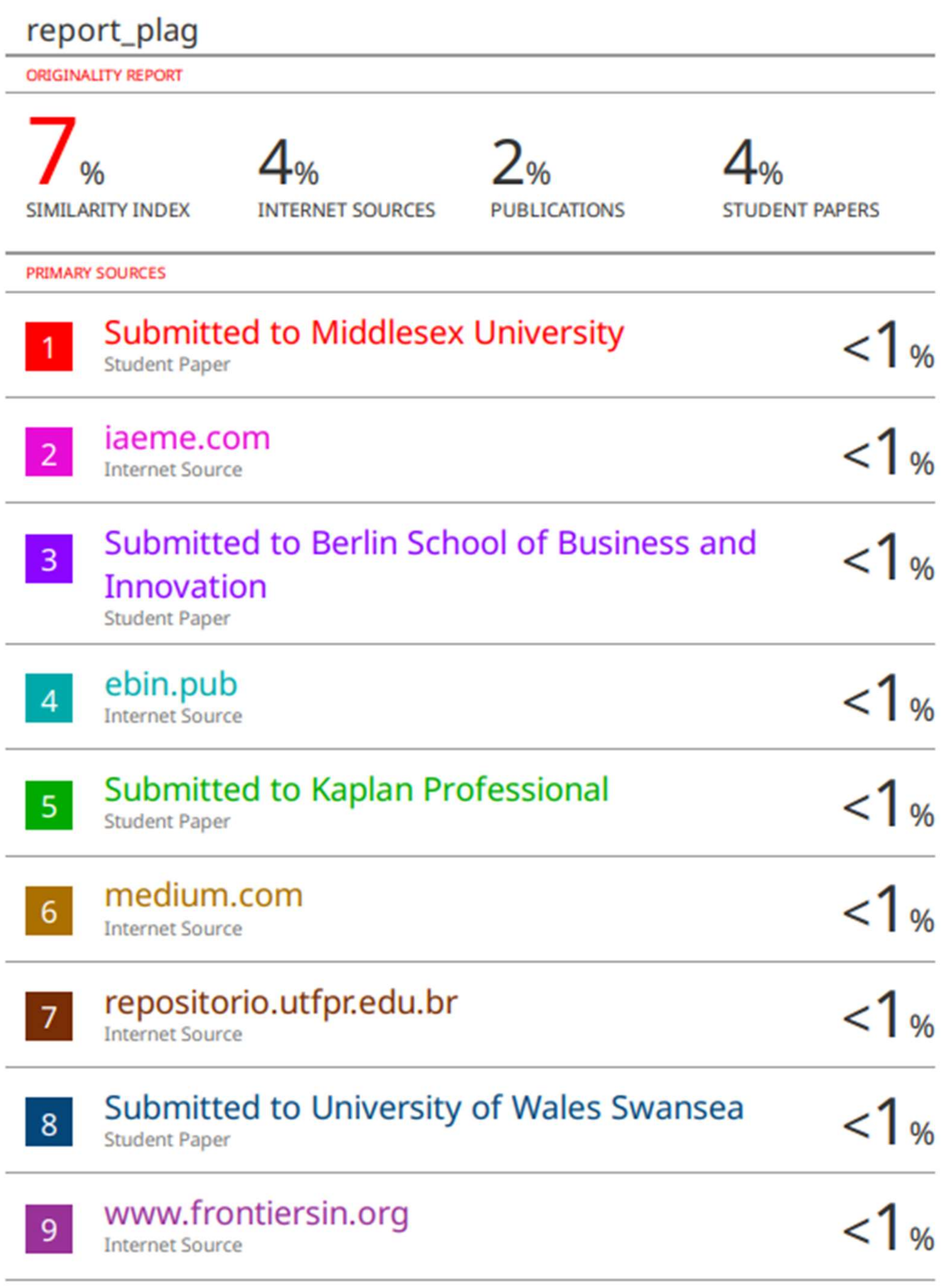
ICCBI Conference
to me

Wed, Nov 1, 3:01 PM (2 days ago)

Dear Researcher
Greetings from ICCBI 2023!
Your manuscript is received, the review team will respond to you with a decision soon.
Thank you

--

PLAGIARISM REPORT



report_plag

ORIGINALITY REPORT

7%	4%	2%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Middlesex University Student Paper	<1%
2	iaeme.com Internet Source	<1%
3	Submitted to Berlin School of Business and Innovation Student Paper	<1%
4	ebin.pub Internet Source	<1%
5	Submitted to Kaplan Professional Student Paper	<1%
6	medium.com Internet Source	<1%
7	repositorio.utfpr.edu.br Internet Source	<1%
8	Submitted to University of Wales Swansea Student Paper	<1%
9	www.frontiersin.org Internet Source	<1%