HEMWATI NANDAN BAHUGUNA GARHWAL UNIVERSITY

(A Central University)

Srinagar (Garhwal), Uttarakhand-246174

School of Engineering and Technology

Session (2020-2024)

A PROJECT REPORT ON

“FARECAST: INTELLIGENT TAXI FARE PREDICTION USING MACHINE LEARNING”

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

Hemwati Nandan Bahuguna Garhwal University,

Srinagar Garhwal (Uttarakhand)

Guided By: -

Dr. Om Prakash

Assistant Professor

Computer Science and Engineering

Submitted By: -

Abhinav Agarwal (20134501011)

Arshad Khan (20134501018)

Shakti Singh (20134501035)

B. Tech CSE 8th Semester

DECLARATION

We **Abhinav Agarwal, Arshad Khan, Shakti Singh** bearing the roll no **20134501011, 20134501018 and 20134501035** respectively, students of Computer Science and Engineering Department at Hemwati Nandan Bahuguna Garhwal University (A Central University), Srinagar (Garhwal), Uttarakhand, submit this project report entitled "**FARECAST : INTELLIGENT TAXI FARE PREDICTION USING MACHINE LEARNING**" to Computer Science and Engineering Department, Hemwati Nandan Bahuguna Garhwal University, for the award of the **Bachelors of Technology degree in Computer Science & Engineering** and declared that the work done is genuine and produced under the guidance of **Dr. Om Prakash**, Assistant Professor in Department of Computer Science and Engineering, Hemwati Nandan Bahuguna Garhwal University. We further declare that the reported work in this project has not been submitted and will not be submitted, either in part or in full, for the award of any degree in this institute or any institute or university.

Abhinav Agarwal	**Arshad Khan**	**Shakti Singh**
20134501011	**20134501018**	**20134501035**

Date: 06 June 2024

Place: Srinagar (Garhwal), Uttarakhand, 246174

CERTIFICATE

This is to certify that, this project report titled "**FARECAST: INTELLIGENT TAXI FARE PREDICTION USING MACHINE LEARNING"** submitted by **Abhinav Agarwal, Arshad Khan**, **Shakti Singh** bearing roll no. **20134501011, 20134501018, 20134501035** is bonafide record of the work carried out by him in partial fulfilment for the requirement of the award of **Bachelor of Technology** in **Computer Science and Engineering** degree from Hemvati Nandan Bahuguna Garhwal University (A Central University) at Srinagar (Garhwal), Uttarakhand.

The project work has been carried out under my guidance and supervision andis approved for submission.

Dr. Om Prakash
Assistant Professor
Department of Computer Science & Engineering
Hemvati Nandan Bahuguna Garhwal University
(A Central University)

ACKNOWLEDGEMENT

We would like to express my deepest gratitude to all people for sprinkling their help and kindness in the completion of this Project. I would like to start this moment by invoking my purest gratitude to **Dr. Om Prakash,** Department of Computer Science and Engineering, Hemvati Nandan Bahuguna GarhwalUniversity (A Central University), Srinagar (Garhwal), Uttarakhand, my project instructor.

The completion of this project could not have been possible without his expertise and invaluable guidance in every phase at Hemvati Nandan BahugunaGarhwal University (A Central University), Srinagar (Garhwal), Uttarakhand for helping me.

We would like to thank **Dr. Prem Nath, Prof M.P Thapliyal (Dean)** all the lab assistants and other staffs of Computer Science and Engineering Department, Hemvati Nandan Bahuguna Garhwal University (A Central University), Srinagar (Garhwal), Uttarakhand, for their kind support. Last but not least, I would like to thank my parents and my friends for their unwaveringbelief despite ups and downs in my journey.

TABLE OF CONTENT

LIST OF FIGURES

ABSTRACT

In regions where ubiquitous ride-hailing services like Ola and Uber are not available, consumers often find themselves at a disadvantage when it comes to negotiating fair taxi fares. The absence of transparent pricing mechanisms and reliable fare estimates creates a significant information gap between passengers and taxi drivers. This information asymmetry leaves passengers vulnerable to potential fare manipulation and exploitation, as they lack the necessary insights to assess the reasonableness of the fares charged.

As a result, there is a pressing need to address this issue by developing a predictive model for taxi fares. Such a system would empower consumers with the negotiating power they need to make informed decisions about their transportation expenses. By providing accurate fare predictions based on factors such as distance, time, traffic conditions, and geographic features, this system aims to level the playing field between passengers and taxi drivers.

Through the deployment of advanced machine learning algorithms and comprehensive data analysis, the proposed predictive model seeks to democratize access to reliable fare estimates. By doing so, it not only enhances consumer satisfaction and transparency in taxi services but also promotes fairness and equity in the transportation industry. Ultimately, the development and implementation of this system have the potential to transform the way taxi fares are calculated and negotiated, leading to a more equitable and consumer-centric transportation ecosystem in regions where ride-hailing services are not universally available.

Chapter 1: Introduction

1.1 Background

In many regions across the globe, popular ride-hailing services such as Ola and Uber are not universally accessible, leaving traditional taxi services as the primary mode of transportation. However, the absence of transparent pricing mechanisms in traditional taxi services often places consumers at a disadvantage when negotiating fair taxi fares. This project aims to address this challenge by developing an intelligent taxi fare prediction system, named FARECAST, using advanced machine learning techniques. By predicting taxi fares accurately based on various factors like distance, time, traffic conditions, and geographic features, FARECAST seeks to empower consumers, reduce information asymmetry, and promote fairness and transparency in the taxi service industry.

Problem Statement: Taxi Fare Prediction

The taxi industry faces numerous challenges in predicting the accurate fare for trips, which is crucial for both operational efficiency and customer satisfaction. Traditional fare calculation methods often rely on simple distance and time metrics, failing to account for the myriad of factors that influence the final fare, such as traffic conditions, time of day, day of the week, and weather conditions. This can lead to significant discrepancies between estimated and actual fares, resulting in customer dissatisfaction and operational inefficiencies.

1.2 Objective

The primary objective of this project is to develop a predictive model for taxi fares that provides accurate fare estimates to consumers. This model will utilize machine learning algorithms to analyze historical data and various influencing factors to predict the cost of a taxi ride. The specific objectives include:

- Collecting and preprocessing taxi fare data.
- Identifying key factors that influence taxi fares.
- Developing and training a machine learning model to predict taxi fares.
- Evaluating the performance of the predictive model.

Significance:

1. **Passenger Experience**: Passengers can make informed decisions regarding their travel costs before starting a trip, leading to higher satisfaction.
2. **Driver Satisfaction**: Drivers can receive fare estimates in advance, helping them manage their expectations and plan their work more efficiently.
3. **Operational Efficiency**: Taxi service providers can optimize their operations by understanding fare patterns and demand fluctuations.
4. **Revenue Management**: Accurate fare predictions can assist in dynamic pricing strategies, enhancing revenue management for taxi companies.

1.3 Motivation Behind the Project

The motivation behind this project stems from the need to empower consumers in regions where ride-hailing services are not widely available. Without transparent pricing, passengers often face the risk of fare manipulation and exploitation. By providing accurate and reliable fare estimates, this project aims to:

- Enhance consumer confidence and satisfaction in using taxi services.
- Promote transparency and fairness in fare negotiations.
- Enable consumers to make informed decisions regarding their transportation expenses.
- Contribute to the overall improvement of the traditional taxi service industry.

1.4 Scope of the Project

The scope of this project includes the development and implementation of a predictive model for taxi fares using machine learning. The project encompasses the following components:

- Data Collection: Gathering historical taxi fare data and related variables such as distance, time, and traffic conditions.
- Data Preprocessing: Cleaning and preparing the data for analysis and model training.
- Model Development: Selecting appropriate machine learning algorithms and training the predictive model.

- Model Evaluation: Assessing the accuracy and performance of the model using relevant metrics.
- User Interface: Creating a user-friendly application or platform where consumers can input trip details and receive fare predictions.
- Deployment: Implementing the system in a real-world environment to provide fare estimates to consumers.

Chapter 2: Requirement Specification and Analysis

2.1 System Requirements

The development of the FARECAST system requires a combination of hardware and software resources. The system requirements are categorized into hardware and software requirements.

2.2 Software and Hardware Requirements

2.2.1 Software Requirements

- Programming Languages: Python, R
- Machine Learning Libraries: scikit-learn, Keras, pandas, NumPy
- Development Environment: Jupyter Notebook, PyCharm, or any preferred IDE
- Database Management: MySQL, SQLite, or any other relational database
- Web Framework Flask
- Extensions and Packages:
- Flask-SQLAlchemy==3.0.2
- flask-security-too
- email_validator
- bcrypt
- flask_restful
- flask_cors
- flask-jwt-extended
- redis
- celery[redis]
- Flask-Caching
- jinja2
- weasyprint==52.5
- Machine Learning Model Storage: Pickle
- Frontend Framework: Vue.js
- Operating System: Windows, macOS, or Linux
- Version Control: Git

2.2.2 Hardware Requirements

- Processor: Intel i5 or higher
- RAM: 8 GB or more
- Storage: 256 GB SSD or more
- GPU: Optional, but recommended for training deep learning models
- Internet Connectivity: Required for data collection, model updates, and deployment.

2.3 System Architecture

The FARECAST system architecture is designed to divide the frontend and backend components to optimize performance and maintainability. The backend server, responsible for handling the machine learning model and API endpoints, runs on port 8080 using Flask. The frontend application, developed using Vue.js, runs on port 8081 and interacts with the backend server to retrieve fare predictions.

This chapter outlines the foundational requirements and specifications necessary for the successful development and implementation of the FARECAST system. The following chapters will delve into the detailed design, implementation strategies, testing methodologies, and potential challenges associated with the project.

Chapter 3: Methodology

3.1 Research Design

Detailed Explanation:

❖ Type of Research:

- Quantitative Research: This involves the collection and analysis of numerical data to understand patterns, relationships, or trends. It’s useful for testing hypotheses or measuring variables across a large sample.
- Qualitative Research: This involves non-numerical data such as text, video, or audio, to understand concepts, thoughts, or experiences. It’s often used for exploratory purposes.
- Mixed Methods: Combines both quantitative and qualitative approaches to provide a comprehensive understanding of the research problem.

❖ Sampling Strategy:

- Probability Sampling: Ensures every member of the population has an equal chance of being selected. Examples include simple random sampling, systematic sampling, and stratified sampling.
- Non-probability Sampling: Selection is based on factors other than random chance. Examples include convenience sampling, purposive sampling, and snowball sampling.

❖ Approach to Data Collection and Analysis:

- Define the population of interest.
- Determine the sample size using methods like power analysis to ensure sufficient data for reliable results.
- Outline data collection methods, ensuring reliability and validity.
- Detail data analysis procedures, specifying statistical tests and models to be used.

Example:

For a study on factors affecting student performance:

- Type of Research: Quantitative research will be used to collect and analyze numerical data on student grades, study habits, and socio-economic status.
- Sampling Strategy: Stratified random sampling will ensure representation from each grade level (freshmen, sophomores, juniors, seniors).
- Approach to Data Collection and Analysis: Surveys will be distributed to collect data on study habits and socioeconomic background, while academic records will be obtained from the school database. Statistical tests (e.g., t-tests, ANOVA) and regression analysis will be used to analyze the data.

Potential Challenges:

- Ensuring the sample is truly representative of the population.
- Maintaining reliability and validity in data collection instruments.

3.2 Data Collection

Detailed Explanation:

Data Sources:

- Primary Data: Collected firsthand through surveys, experiments, or direct observations.
- Secondary Data: Obtained from existing sources such as databases, research studies, or public records.

Sampling Methods:

- Stratified sampling will be used to ensure different subgroups (e.g., grade levels) are proportionately represented.

Tools and Instruments:

- Online survey platforms like Google Forms or SurveyMonkey.
- Database access tools for retrieving academic records.

Example:

- Data Sources: Primary data will be collected via surveys on study habits and well-being. Secondary data will be gathered from school databases including grades and attendance records.
- Sampling Methods: Stratified sampling will ensure that students from all grade levels are included proportionately.
- Tools and Instruments: Google Forms for surveys, SQL queries for database extraction.

Potential Challenges:

- Achieving a high response rate for surveys.
- Ensuring data privacy and compliance with regulations (e.g., FERPA for student records).

3.3 Data Preprocessing

Detailed Explanation:

- Data Cleaning: Removing or correcting errors and inconsistencies in the data. This may involve:
- Handling missing values by imputation or deletion.
- Removing duplicate records.
- Correcting data entry errors.

Data Normalization/Standardization:

- Scaling features to a standard range (e.g., 0 to 1 for normalization, mean 0 and variance 1 for standardization) to ensure comparability.

Data Transformation:

- Converting data into a suitable format for analysis:
- Encoding categorical variables (e.g., one-hot encoding).
- Creating new features through transformation or combination of existing ones.

Example:

- Data Cleaning: Missing values for hours studied will be imputed using the mean, and any duplicate records will be removed.
- Data Normalization/Standardization: Study hours and GPA will be standardized to have a mean of 0 and standard deviation of 1.
- Data Transformation: Categorical variables like participation in extracurricular activities will be one-hot encoded.

Potential Challenges:

- Handling a large amount of missing data without introducing bias.
- Ensuring transformations do not distort the underlying patterns in the data.

3.4 Feature Selection and Engineering

Detailed Explanation:

- Feature Selection: Identifying the most relevant features for the model. Techniques include:
- Correlation analysis to find relationships between features and the target variable.
- Recursive feature elimination to iteratively remove less important features.
- Feature importance from models like random forests.

Feature Engineering:

Creating new features that capture important information.
This might involve:

- Domain knowledge to create meaningful features.
- Polynomial features or interaction terms.
- Aggregating features (e.g., creating a composite score).

Dimensionality Reduction:

Techniques like Principal Component Analysis (PCA) to reduce the number of features while retaining important information.

Example:

- Feature Selection: Correlation analysis will identify key predictors such as study hours, socioeconomic status, and attendance.
- Feature Engineering: Creating a composite score for study habits by combining hours studied and the quality of study methods.
- Dimensionality Reduction: Applying PCA if the dataset has many features to reduce dimensionality while retaining most variance.

Potential Challenges:

- Avoiding overfitting by selecting too many features.
- Ensuring new features are meaningful and improve model performance.

3.5 Model Selection

Detailed Explanation:

- Model Types: Considering different algorithms based on the research problem:
- Regression models (e.g., linear regression for continuous outcomes).
- Classification algorithms (e.g., logistic regression, decision trees, random forests for categorical outcomes).
- Clustering techniques (e.g., K-means for grouping similar observations).

Criteria for Selection:

- Model interpretability (e.g., linear regression for straightforward interpretation).
- Complexity (e.g., decision trees vs. random forests).
- Performance metrics (e.g., accuracy, precision, recall, F1 score for classification; MSE for regression)

Example:

For predicting student performance:

- Model Types: Considering linear regression for continuous GPA prediction and decision trees for classification (e.g., pass/fail).
- Criteria for Selection: Models will be evaluated based on interpretability and performance metrics such as MSE for regression models and accuracy for classification models.

Potential Challenges:

- Balancing model complexity and interpretability.
- Choosing a model that generalizes well to unseen data.

3.6 Model Training and Validation

Detailed Explanation:

- Data Splitting: Splitting the dataset into training and validation sets (e.g., 80% training, 20% validation) to evaluate model performance.
- Model Training: Fitting the selected model on the training data.
- Model Validation: Assessing performance on the validation set using appropriate metrics (e.g., MSE, R-squared for regression; accuracy, precision, recall, F1 score for classification).
- Hyperparameter Tuning: Optimizing model parameters to enhance performance, often using grid search or random search.

Example:

- Data Splitting: 80% of the data will be used for training, and 20% for validation.
- Model Training: Models will be trained on the training data.

- Model Validation: Performance will be evaluated using MSE and R-squared for regression models, and accuracy, precision, recall, and F1 score for classification models.
- Hyperparameter Tuning: Grid search will be used to optimize parameters for decision trees and random forests.

Potential Challenges:

- Ensuring the split data represents the overall dataset.
- Avoiding overfitting during hyperparameter tuning.

3.7 Tools and Technologies Used

Detailed Explanation:

- Programming Languages: Python is often preferred due to its extensive libraries and ease of use.
- Software Packages and Libraries:
- pandas and numpy for data manipulation.
- scikit-learn for machine learning.
- matplotlib and seaborn for data visualization.
- Platforms and Environments: Jupyter Notebook for interactive coding and visualization, Google Colab for collaborative work, and Git for version control.

Example:

- Programming Languages: Python will be used for the entire analysis.
- Software Packages and Libraries: Data manipulation will be done using pandas and numpy, machine learning with scikit-learn, and visualization with matplotlib and seaborn.
- Platforms and Environments: Jupyter Notebook will be the primary environment for coding and visualization, with version control managed using Git and GitHub.

Potential Challenges:

- Ensuring compatibility between different libraries and tools.
- Managing version control and collaboration effectively.

Chapter 4: Implementation

The prediction of taxi fares using machine learning models is a significant application of data science that can provide substantial benefits across various stakeholders. This includes taxi companies, drivers, passengers, and regulatory bodies. For taxi companies, accurate fare predictions enable better demand forecasting and dynamic pricing strategies, which can optimize revenue. Drivers benefit from understanding potential earnings and efficient route planning, while passengers gain from transparent fare estimations, enhancing trust and satisfaction. Regulatory authorities can utilize predictive models to monitor and ensure fair practices in fare structures.

This analysis is structured into several critical steps, each essential for building a robust machine learning model for fare prediction: data loading and initial exploration, data preprocessing, exploratory data analysis (EDA), model building, model evaluation, hyper parameter tuning, final model evaluation, and conclusion. Each step will be detailed to provide a comprehensive understanding of the process and techniques used.

4.1 Data Loading and Initial Exploration

Data loading is the initial step in any data analysis project. For this analysis, the dataset is sourced from a Kaggle competition, focusing on taxi trips in New York City. The dataset is loaded into a pandas Data Frame for ease of manipulation and analysis. Understanding the structure of the dataset and its contents is essential for further analysis. Key features in the dataset include:

- VendorID: Identifier for the vendor providing the trip record.
- tpep_pickup_datetime: The date and time when the meter was engaged.
- tpep_dropoff_datetime: The date and time when the meter was disengaged.
- passenger_count: Number of passengers in the vehicle.
- trip_distance: Distance of the trip in miles.
- RatecodeID: Rate code indicating the type of fare.
- store_and_fwd_flag: Flag indicating whether the trip record was held in vehicle memory before sending to the vendor.
- PULocationID: Pickup location ID.

- DOLocationID: Dropoff location ID.
- payment_type: Payment method used (e.g., credit card, cash).
- total_amount: Total fare amount.

Initial exploration involves displaying the first few rows of the dataset to understand its structure, checking for missing values to identify any gaps in the data, and summarizing the data to get an overview of distributions and data types.

Understanding these features is crucial as they form the foundation of the subsequent analysis. For example, `trip_distance` directly influences the fare amount, while `passenger_count` and `RatecodeID` may indirectly impact the fare. The date time features (`tpep_pickup_datetime` and `tpep_dropoff_datetime`) can provide insights into temporal patterns affecting fare amounts, such as peak hours and days.

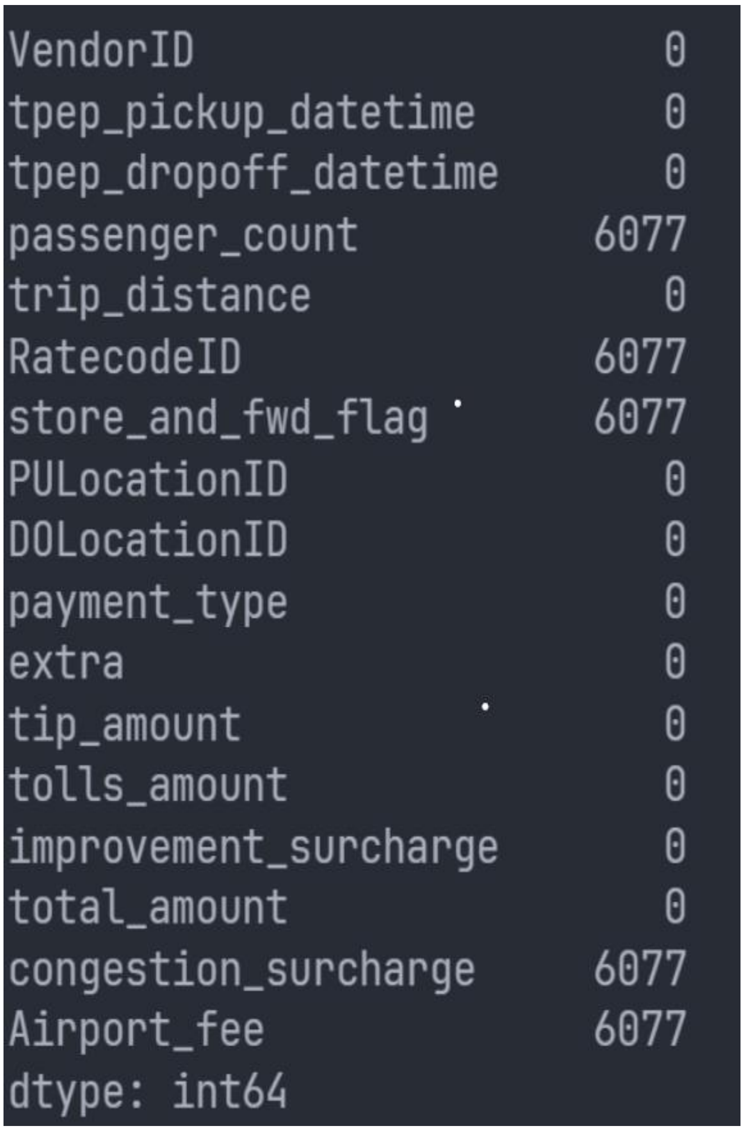
4.2 Data Preprocessing

Data preprocessing is a vital step in preparing the dataset for machine learning modeling. This step ensures that the data is clean and suitable for analysis, addressing any inconsistencies or missing values. Key preprocessing steps include handling missing values, converting data types, creating new features, and encoding categorical variables.

4.3 Handling Missing Values

Missing values can adversely affect model performance. Therefore, it is essential to handle them appropriately. For numerical columns, missing values can be filled with summary statistics like the mean or median. For categorical columns, the mode is often used. This ensures that no information is lost and the dataset remains consistent.

For instance, in our dataset, columns such as `passenger_count` and `RatecodeID` might have missing values that need to be filled with their respective medians. The `store_and_fwd_flag` column, being categorical, can be filled with the mode.



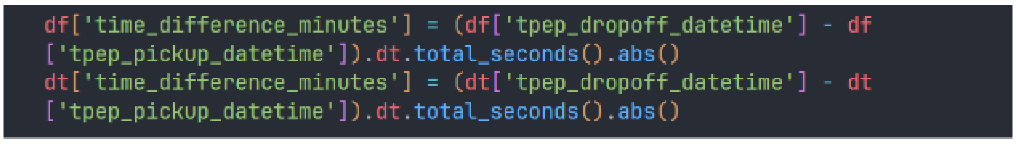
```
VendorID                    0
tpep_pickup_datetime        0
tpep_dropoff_datetime       0
passenger_count          6077
trip_distance               0
RatecodeID               6077
store_and_fwd_flag       6077
PULocationID                0
DOLocationID                0
payment_type                0
extra                       0
tip_amount                  0
tolls_amount                0
improvement_surcharge       0
total_amount                0
congestion_surcharge     6077
Airport_fee              6077
dtype: int64
```

Fig-4.1 – Null Values of Dataset

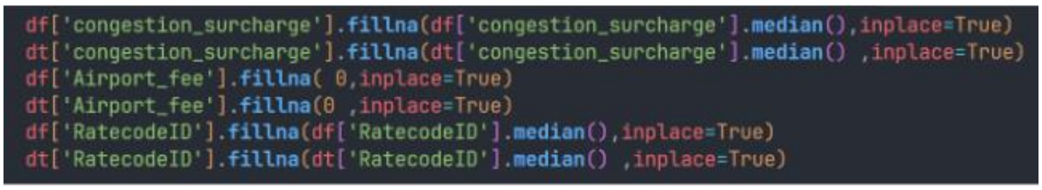
4.4 Date time Conversion

Datetime features, such as pickup and dropoff times, are crucial for temporal analysis. Converting these columns to date time format allows for the extraction of additional features like the hour of the day, day of the week, and trip duration. These features can provide significant insights into patterns affecting taxi fares.

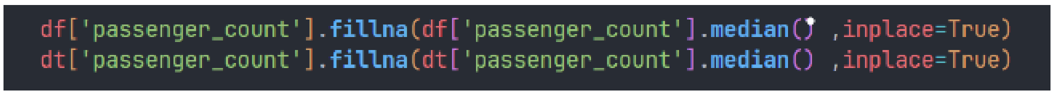
For example, converting `tpep_pickup_datetime` and `tpep_dropoff_datetime` to datetime formats enables the calculation of `trip_duration`, which is the difference between dropoff and pickup times. This feature can be crucial for understanding how trip length in time correlates with fare amounts.



```
df['time_difference_minutes'] = (df['tpep_dropoff_datetime'] - df
['tpep_pickup_datetime']).dt.total_seconds().abs()
dt['time_difference_minutes'] = (dt['tpep_dropoff_datetime'] - dt
['tpep_pickup_datetime']).dt.total_seconds().abs()
```



```
df['congestion_surcharge'].fillna(df['congestion_surcharge'].median(),inplace=True)
dt['congestion_surcharge'].fillna(dt['congestion_surcharge'].median() ,inplace=True)
df['Airport_fee'].fillna( 0,inplace=True)
dt['Airport_fee'].fillna(0 ,inplace=True)
df['RatecodeID'].fillna(df['RatecodeID'].median(),inplace=True)
dt['RatecodeID'].fillna(dt['RatecodeID'].median() ,inplace=True)
```



```
df['passenger_count'].fillna(df['passenger_count'].median() ,inplace=True)
dt['passenger_count'].fillna(dt['passenger_count'].median() ,inplace=True)
```

4.5 **Feature Engineering**

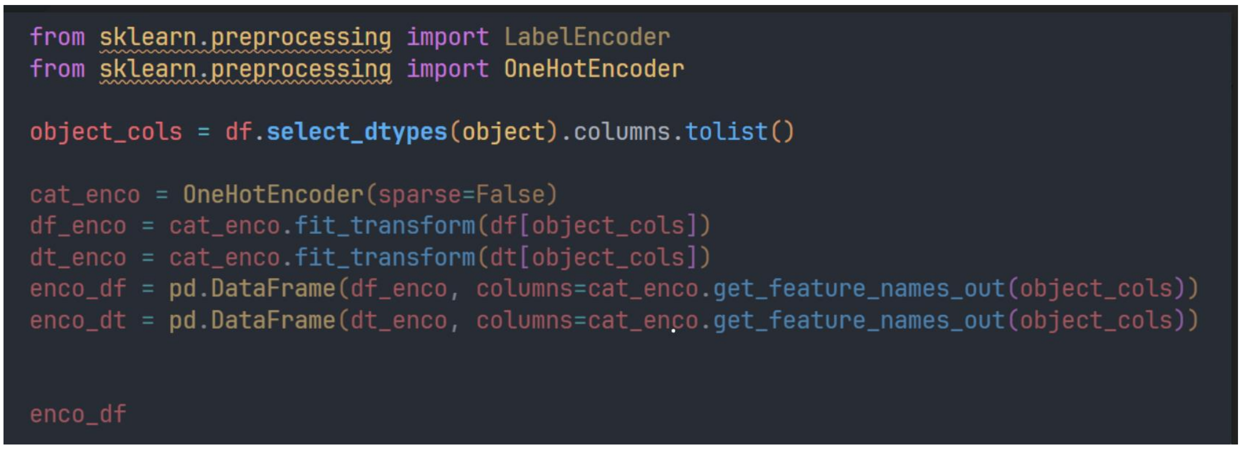
Feature engineering involves creating new features that can provide additional insights to the model. For instance, calculating the trip duration from the pickup and dropoff times can be useful. Additionally, extracting temporal features like the hour of the day, day of the week, or month from the pickup time can help identify patterns related to time.

Creating features such as `trip_duration` and extracting hour, day, and month from `tpep_pickup_datetime` can significantly enhance the model's ability to predict fares accurately. For instance, fares might be higher during rush hours, weekends, or holidays, and these temporal features can capture such patterns.

4.6 Encoding Categorical Variables

Machine learning algorithms require numerical input. Therefore, categorical variables need to be encoded into numerical values. One-hot encoding is a common technique used for this purpose. This involves creating binary columns for each category, allowing the model to interpret these variables effectively.

For example, categorical variables like `VendorID`, `RatecodeID`, `store_and_fwd_flag`, and `payment_type` are converted into numerical format using one-hot encoding. This ensures that the machine learning algorithms can process these variables correctly.



```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

object_cols = df.select_dtypes(object).columns.tolist()

cat_enco = OneHotEncoder(sparse=False)
df_enco = cat_enco.fit_transform(df[object_cols])
dt_enco = cat_enco.fit_transform(dt[object_cols])
enco_df = pd.DataFrame(df_enco, columns=cat_enco.get_feature_names_out(object_cols))
enco_dt = pd.DataFrame(dt_enco, columns=cat_enco.get_feature_names_out(object_cols))


enco_df
```

4.7 **Exploratory Data Analysis (EDA)**

Exploratory Data Analysis (EDA) is a critical step to understand the data and gain insights that can inform the modeling process. EDA involves visualizing the data, summarizing distributions, and identifying patterns and correlations.

4.7.1Histogram Analysis

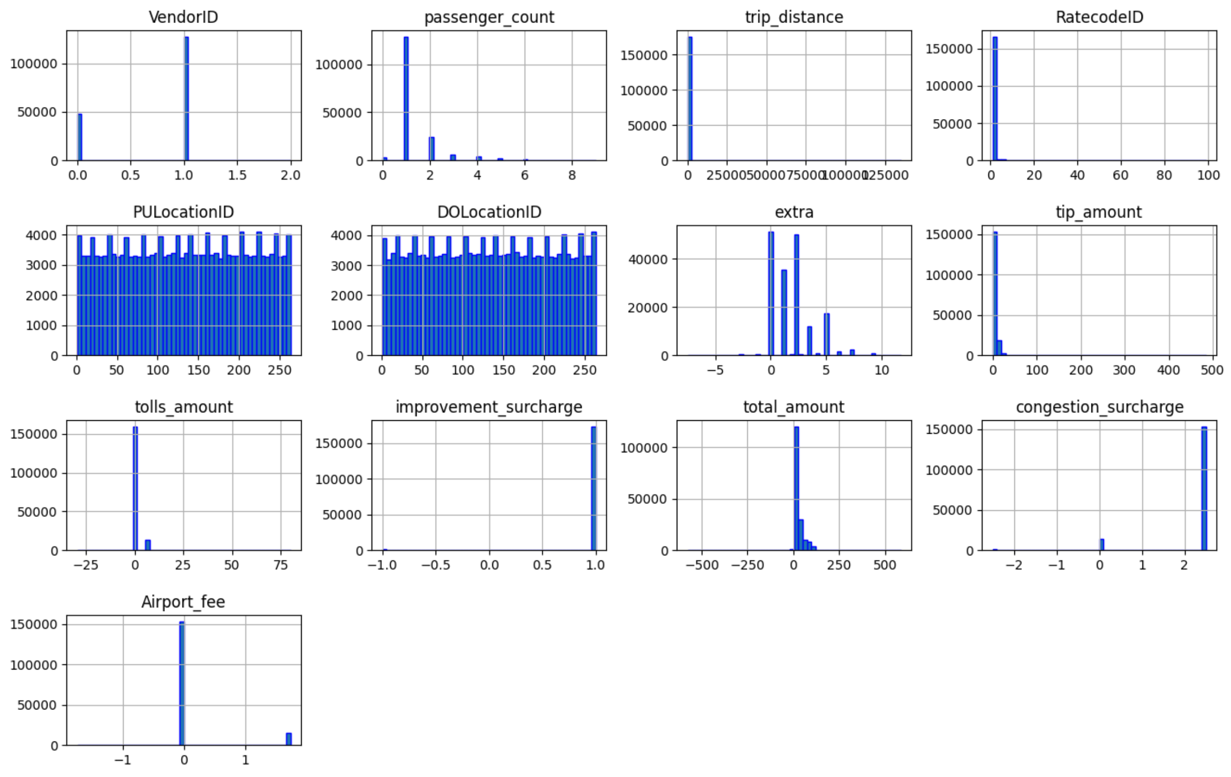


Fig 4.2- Histogram

4.7.2 Bar Chart of Store and Forward Flag

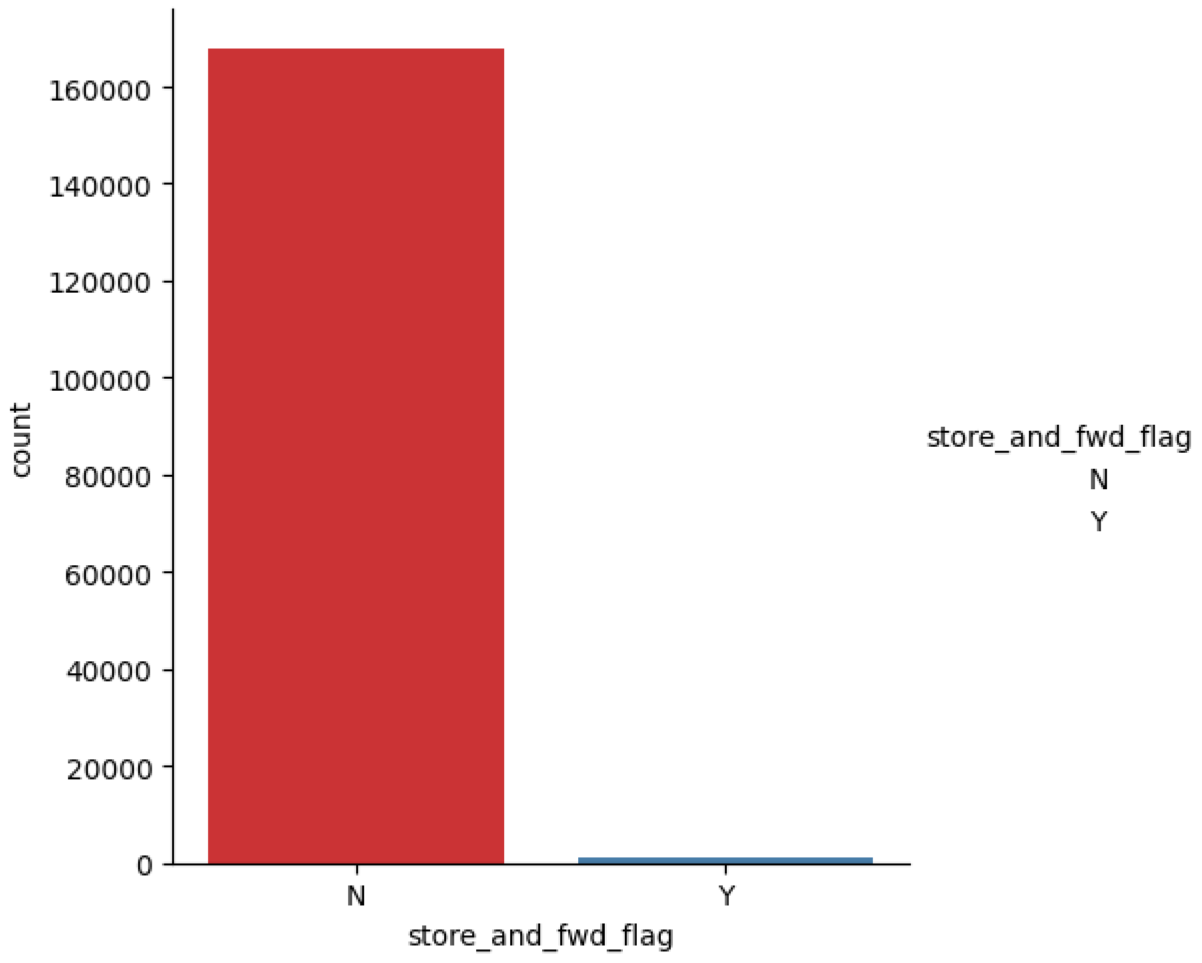


Fig 4.3 – Bar Chart of Store and Fwd flag

4.7.3 Bar Chart of Payment

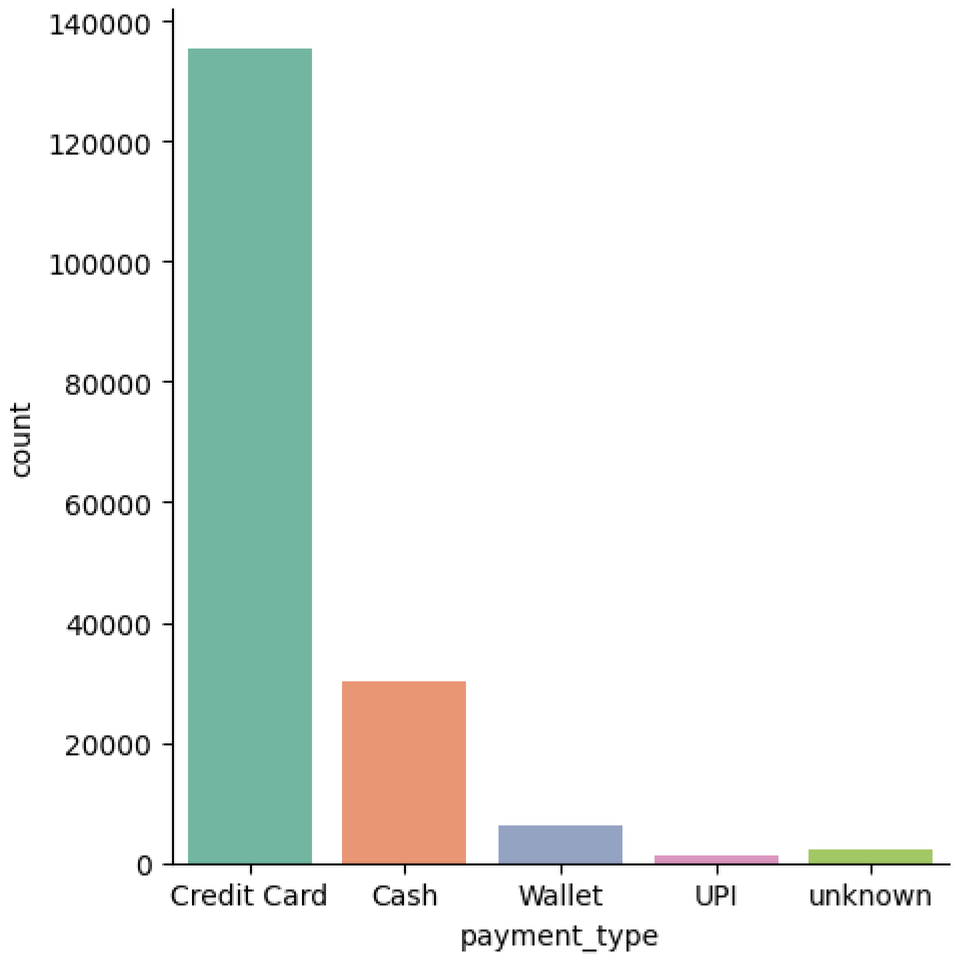


Fig 4.4 – Bar Chart of Payment

4.8 Outlier Analysis of Features

Outlier analysis involves identifying data points that are significantly different from the rest of the data. This can be done using various statistical methods and visualizations. Below is a detailed approach for performing outlier analysis on features in a dataset:

1. **Summary Statistics**: Identify outliers using basic summary statistics like mean, standard deviation, and quantiles.
2. **Boxplots**: Visualize the distribution of data and identify outliers.
3. **Z-Scores**: Identify outliers based on standard deviations from the mean.
4. **IQR Method**: Identify outliers based on the interquartile range.

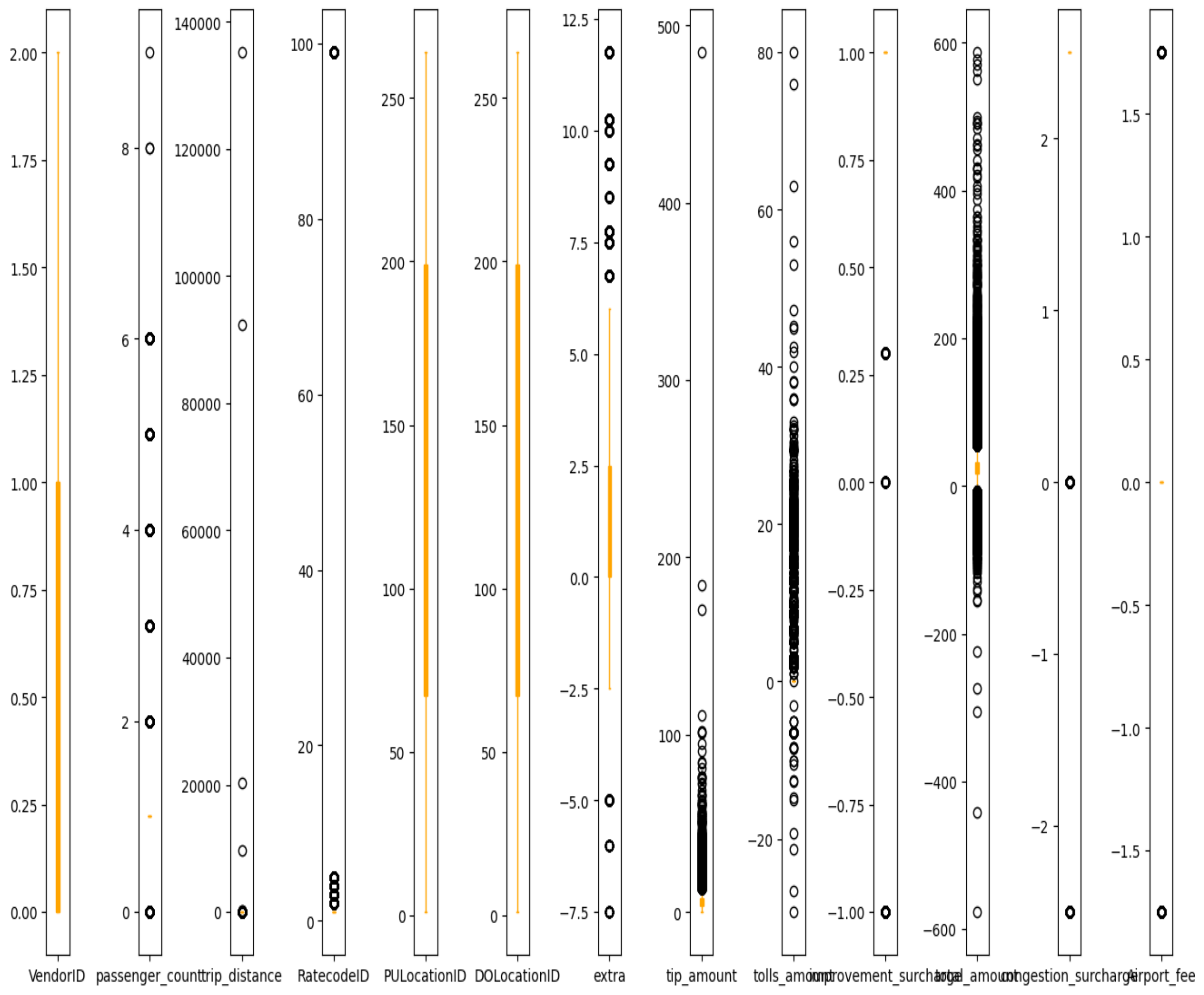


Fig 4.5 – Box Plot of Features

4.9 Skewness Distribution OF Fare Amount

Distribution of Total Amount

The distribution of the target variable (total amount) is examined to understand its range and central tendency. Visualizing this distribution helps in identifying any outliers or skewness in the data.

For instance, plotting a histogram of the total fare amount reveals how fares are distributed. This can show whether the data is skewed towards lower or higher fare amounts and if there are any extreme values that might need to be handled.

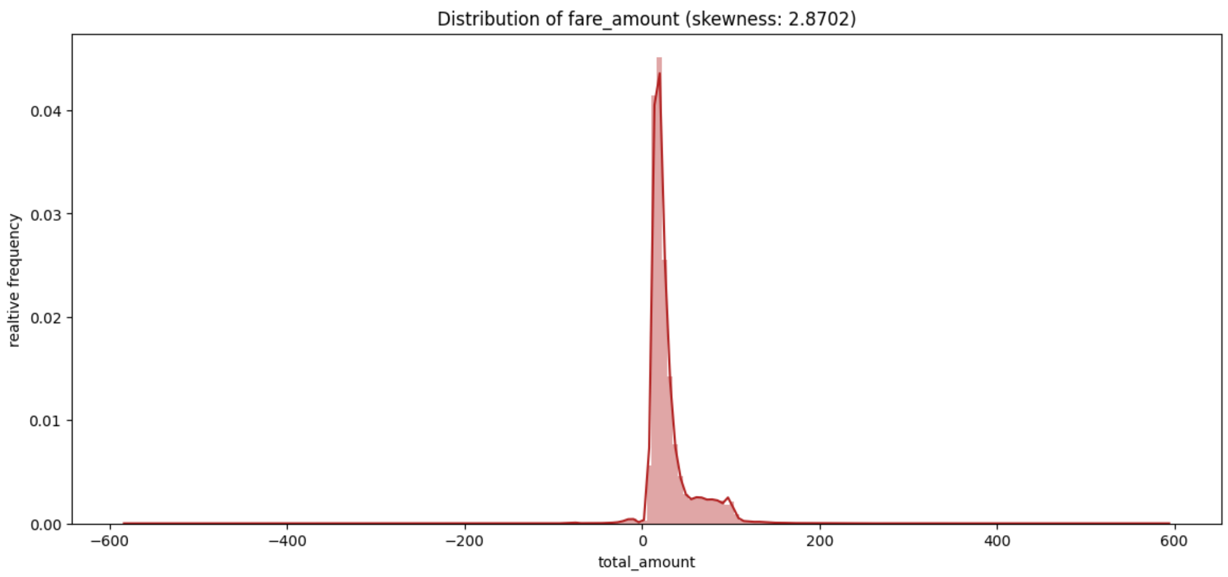


Fig 4.6 – Skew Chart of Fare_amount

4.10 Correlation Matrix

A correlation matrix is used to identify relationships between features. A heatmap can visually represent these correlations, highlighting which features have strong positive or negative correlations with the target variable.

For example, a heatmap of the correlation matrix shows the correlation coefficients between different features. This helps in identifying features that have a strong relationship with the target variable, `total_amount`. Features with high positive or negative correlations are particularly interesting as they can significantly impact the model's performance.

4.10 Pipeline

A machine learning pipeline is a series of interconnected steps that automates the process of building, training, and deploying machine learning models. It essentially breaks down the complex machine learning workflow into smaller, more manageable tasks.

Benefits of using Machine Learning Pipelines

- **Automation:** Pipelines automate the machine learning workflow, saving time and effort for data scientists and machine learning engineers.
- **Reproducibility:** Pipelines ensure that the model training process is repeatable, making it easier to compare different models and track changes over time.
- **Scalability:** Pipelines can be easily scaled to handle larger datasets and more complex models.
- **Improved Efficiency:** Pipelines streamline the development process, leading to faster model development and deployment cycles.

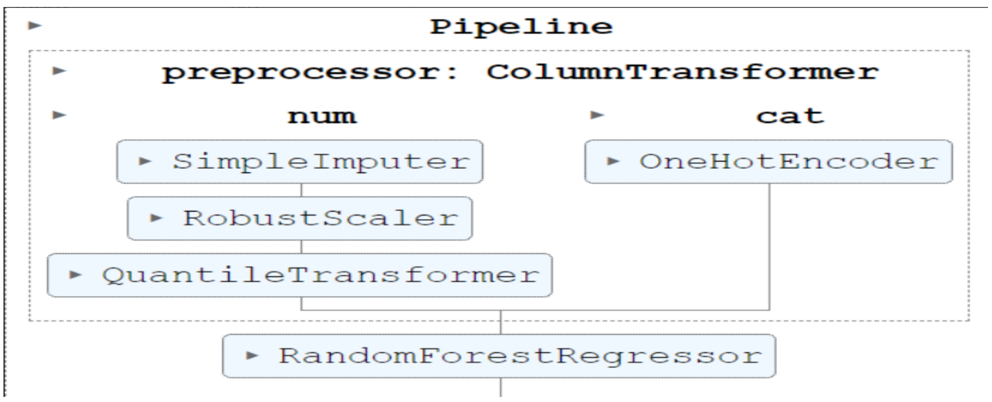


Fig 4.7 – Pipeline Modelling

4.11 Model Building

Building predictive models is the core of the analysis. Several regression models are explored to predict taxi fares, including Linear Regression, Decision Tree Regressor, Random Forest Regressor, and XGBBoosting Regressor.

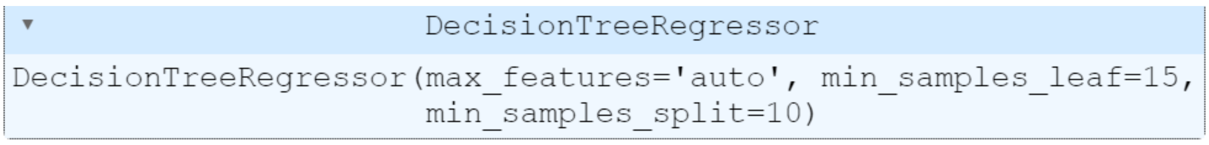
4.12 Linear Regression

Linear Regression assumes a linear relationship between the features and the target variable. It is often used as a baseline model due to its simplicity and interpretability. The model is trained on the training dataset and its performance is evaluated on the test dataset.

In practice, Linear Regression fits a line that minimizes the mean squared error between the predicted and actual fare amounts. Despite its simplicity, it provides a good baseline to compare with more complex models.

4.13 Decision Tree Regressor

Decision Trees split the data based on feature values to make predictions. They can capture non-linear relationships but are prone to over fitting. Over fitting occurs when the model captures noise in the training data, leading to poor generalization to new data.
A Decision Tree Regressor creates a tree where each node represents a feature, and each branch represents a decision based on that feature's value.



```
▾                    DecisionTreeRegressor
DecisionTreeRegressor(max_features='auto', min_samples_leaf=15,
                      min_samples_split=10)
```

Fig 4.8 – Pipeline Modelling

4.14 Model Performance

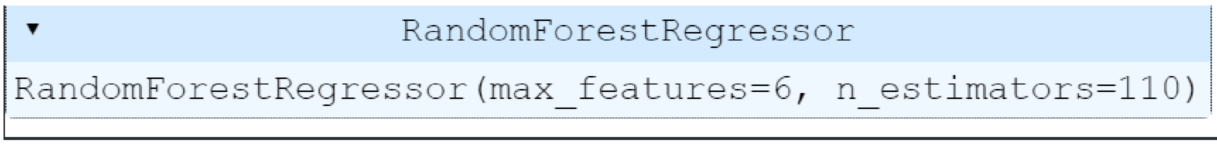
```
Decision tree R-squared score: 0.9432876491470197

Decision tree Mean Squared Error: 36.90602458588367
```

4.14.1 Random Forest Regressor

Random Forests are an ensemble method that builds multiple decision trees and averages their predictions. This reduces over fitting and improves accuracy. Ensemble methods combine the strengths of multiple models to achieve better performance than individual models.

In a Random Forest, each tree is built on a random subset of the data and features. This diversity among trees helps the ensemble model generalize better to new data. The final prediction is an average of the predictions from all trees in the forest.



```
▾               RandomForestRegressor
RandomForestRegressor(max_features=6, n_estimators=110)
```

Fig 4.9 – Random Forest Regressor

4.14.2 Model Performance

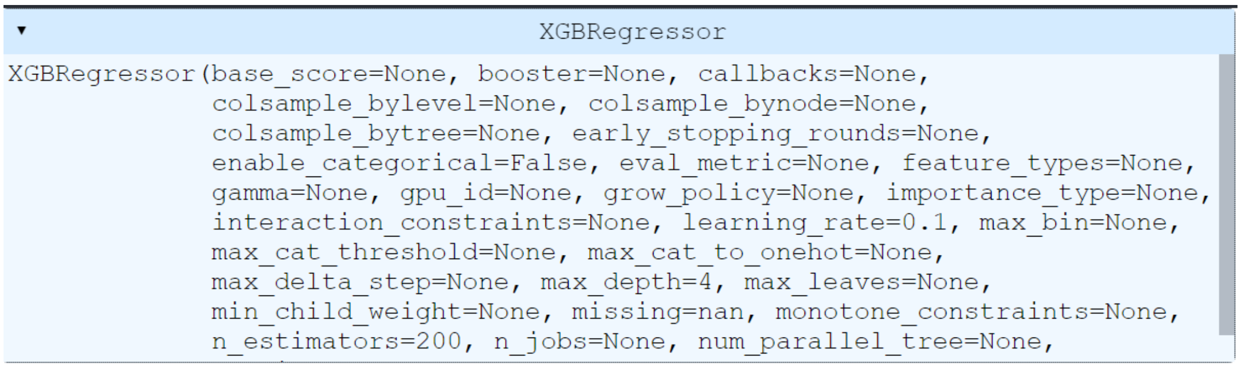
```
Random Forest Mean Squared Error: 30.639741339182756

Random Forest R-squared score: 0.9529168535389493
```

4.14.3 XGB Boosting Regressor

XGB Boosting is another ensemble method that builds trees sequentially, with each tree correcting the errors of the previous one. This method is powerful but can be sensitive to overfitting.

XGB Boosting builds trees in a stage-wise manner, where each tree tries to reduce the residual errors from the previous trees. This sequential approach allows the model to focus on difficult-to-predict cases, leading to improved accuracy.



```
▾                          XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=4, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=200, n_jobs=None, num_parallel_tree=None,
```

Fig 4.10 – XGB Regressor

4.14.4 **Model Evaluation**

Model evaluation involves assessing the performance of the models using appropriate metrics. The primary metrics used for regression models include Mean Squared Error (MSE) and R-squared (R2).

- Mean Squared Error (MSE): Measures the average squared difference between the predicted and actual values. Lower values indicate better performance.

- R-squared (R2): Represents the proportion of the variance in the target variable that is explained by the model. Higher values indicate better performance.

Evaluating models using these metrics helps in understanding how well the models are performing and comparing different models. MSE focuses on the magnitude of prediction errors, while R-squared provides a sense of how well the model explains the variability in the data.

```
XGB Regressor R-squared score: 0.9527209650333689

XGB Regressor Mean Squared Error: 30.76721738089694
```

4.14.5 **Hyper parameter Tuning**

Hyper parameter tuning involves optimizing the parameters of the models to enhance their performance. Techniques such as Grid Search and Random Search are commonly used.

4.14.6 **Grid Search**

Grid Search is a systematic way to test different hyper parameter combinations and find the best set of parameters for the model. It involves defining a grid of possible values for each hyper parameter and evaluating the model performance for each combination.

For example, in a Random Forest model, hyperparameters like the number of trees (`n_estimators`), maximum depth of the trees (`max_depth`), and the minimum number of samples required to split a node (`min_samples_split`) can be tuned. Grid Search evaluates all possible combinations of these parameters to find the optimal set.

4.14.7 Random Search

Random Search is an alternative to Grid Search that randomly samples from the hyper parameter space. It is often more efficient than Grid Search, especially when the number of hyperparameters is large.

Random Search evaluates a fixed number of random combinations from the hyper parameter space. This approach can find good hyper parameter settings faster than Grid Search by covering a broader range of values.

Hyper parameter tuning is crucial for improving model performance. It ensures that the models are not only accurate but also generalize well to new data.

4.15 **Final Model Evaluation**

The best model, determined through hyper parameter tuning, is evaluated on the test set to assess its performance. The final evaluation includes calculating the same performance metrics (MSE and R2) and interpreting the results.

The final model evaluation confirms whether the tuning process has improved the model's performance. It provides a comprehensive understanding of the model's strengths and weaknesses, and how well it is likely to perform on unseen data.

The analysis demonstrates how machine learning models can be used to predict taxi fares. The process involves several critical steps, from data preprocessing to model evaluation. The models' performance is measured using appropriate metrics, and hyper parameter tuning is employed to optimize the models. The results show that ensemble methods like Random Forest and XGBBoosting provide superior performance compared to simpler models like Linear Regression and Decision Trees.

4.15.1Comparison of Mean Squared _Error

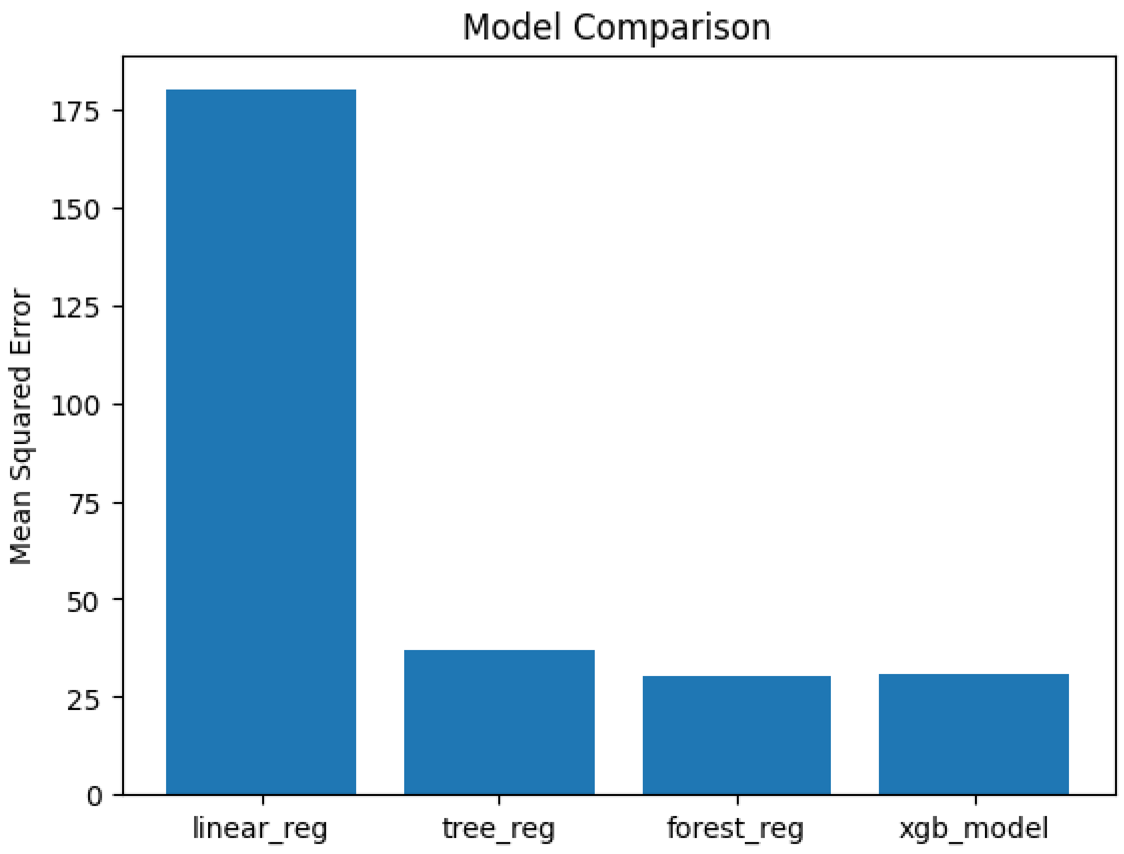


Fig 4.11 – Model Comparison

4.15.2 Comparison of R2_Score

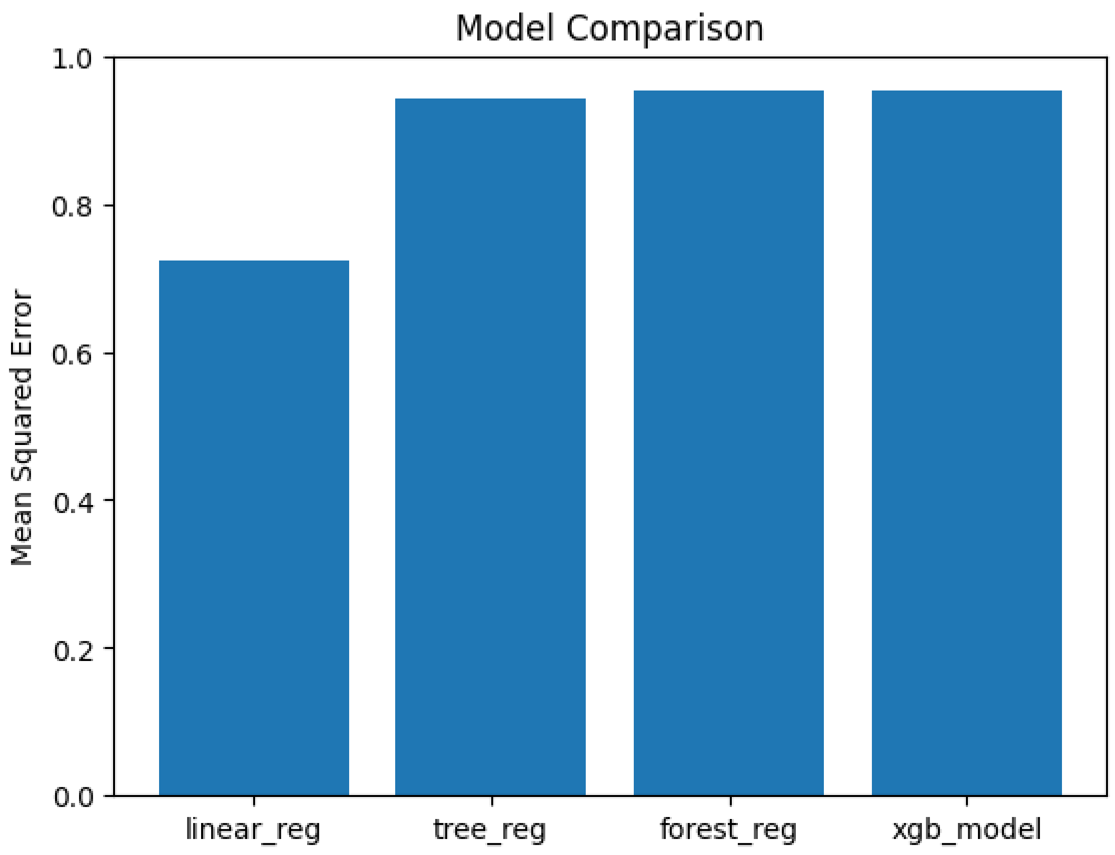
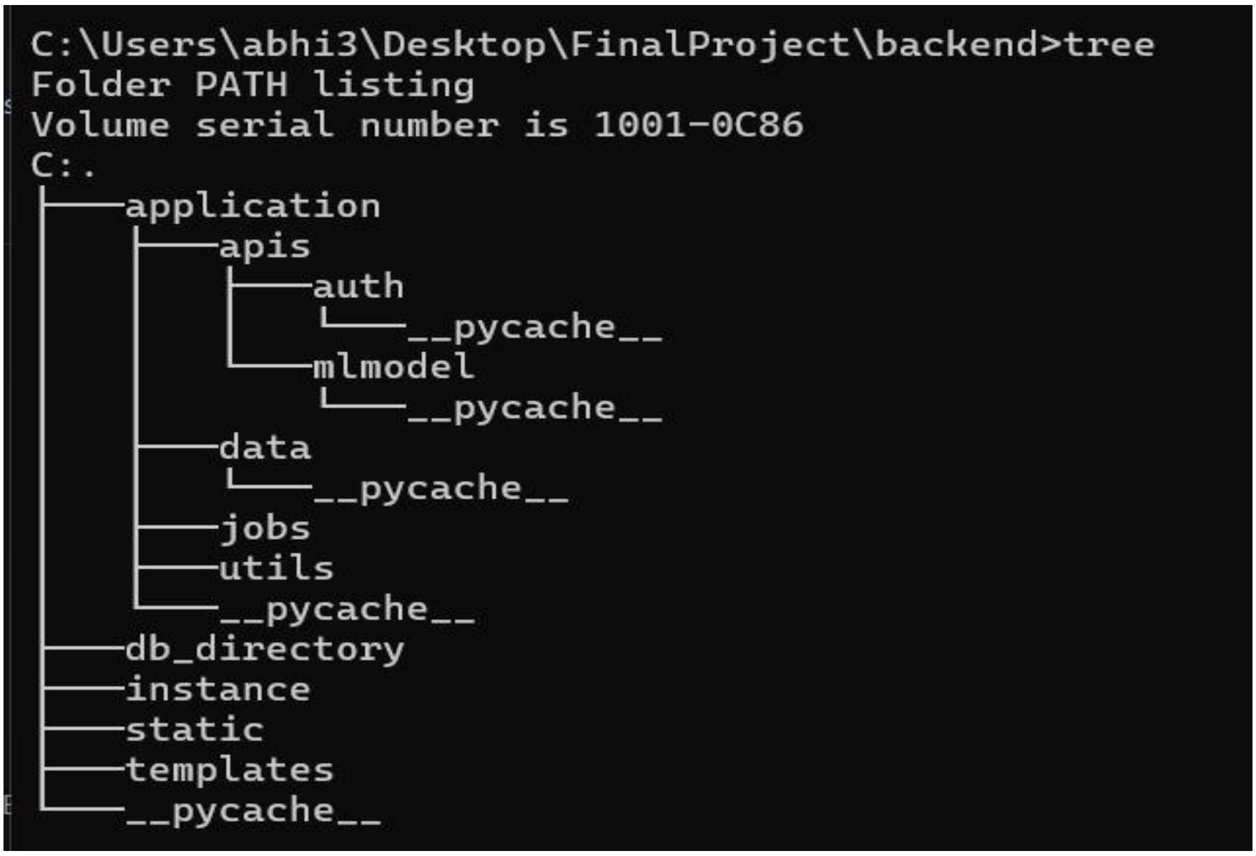


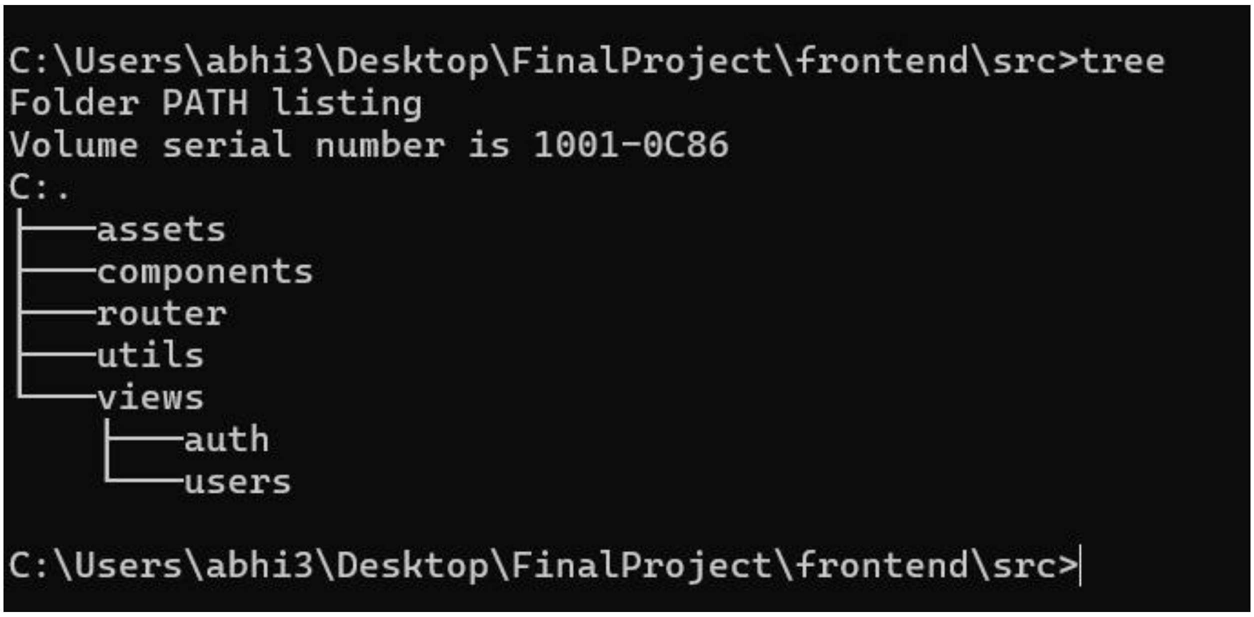
Fig 4.12 – Model R2_Score

Chapter: 5 User Interface and Experience

5.1 Root Directories



```
C:\Users\abhi3\Desktop\FinalProject\backend>tree
Folder PATH listing
Volume serial number is 1001-0C86
C:.
├───application
│   ├───apis
│   │   ├───auth
│   │   │   └───__pycache__
│   │   └───mlmodel
│   │       └───__pycache__
│   ├───data
│   │   └───__pycache__
│   ├───jobs
│   ├───utils
│   └───__pycache__
├───db_directory
├───instance
├───static
├───templates
└───__pycache__
```



```
C:\Users\abhi3\Desktop\FinalProject\frontend\src>tree
Folder PATH listing
Volume serial number is 1001-0C86
C:.
├───assets
├───components
├───router
├───utils
└───views
    ├───auth
    └───users

C:\Users\abhi3\Desktop\FinalProject\frontend\src>
```

5.2 Data Flow Diagram

The traditional system collects data from both riders and drivers. This data includes the rider's location and the driver's location. This data is then used to match riders with drivers.

The predicted system is similar to the traditional system, but it also collects additional data. This additional data is used to predict rider demand. This information can then be used to improve the efficiency of the ride-hailing system. For example, the system could use this information to pre-position drivers in areas where there is high demand.

The text at the bottom of the image says "Ride-hailing system evaluation". This suggests that the diagram is part of a larger study that is evaluating the effectiveness of ride-hailing systems.

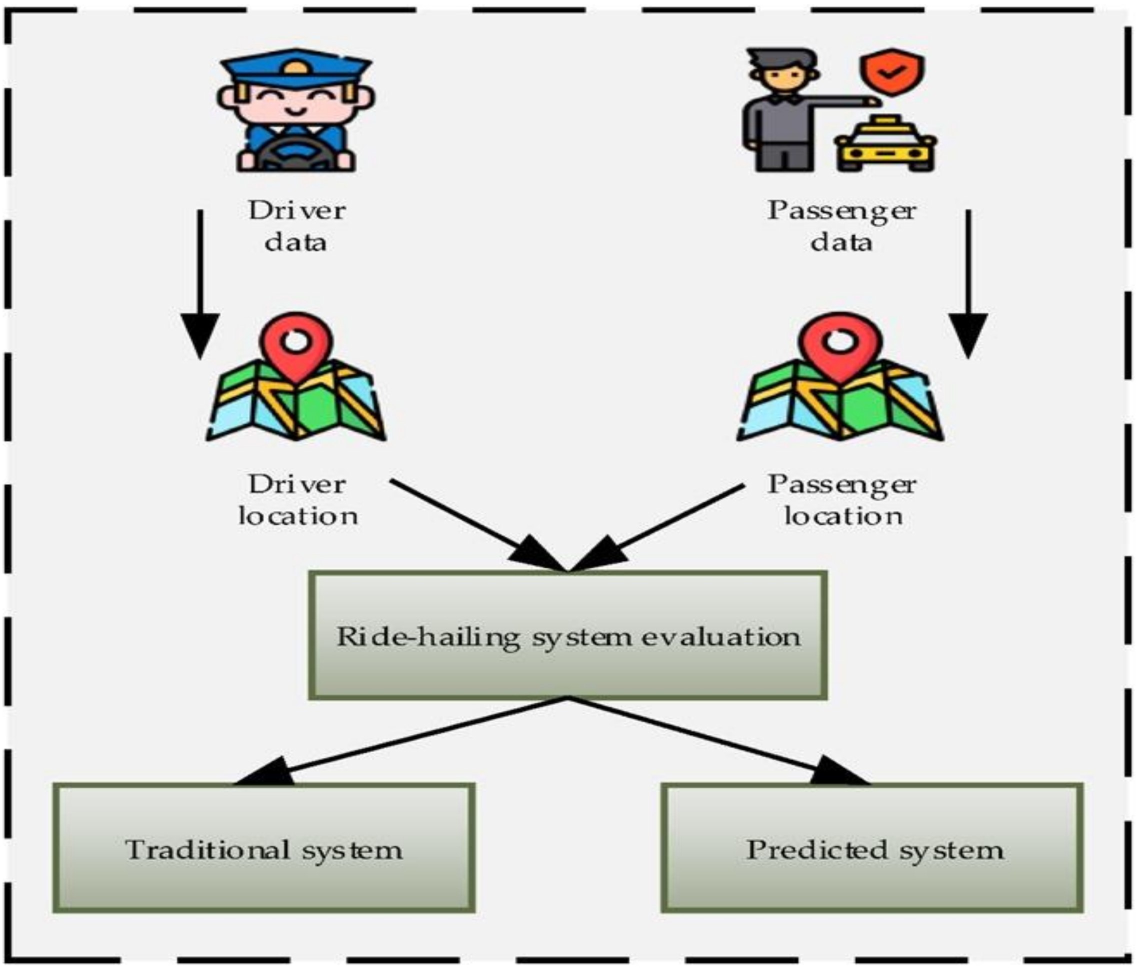
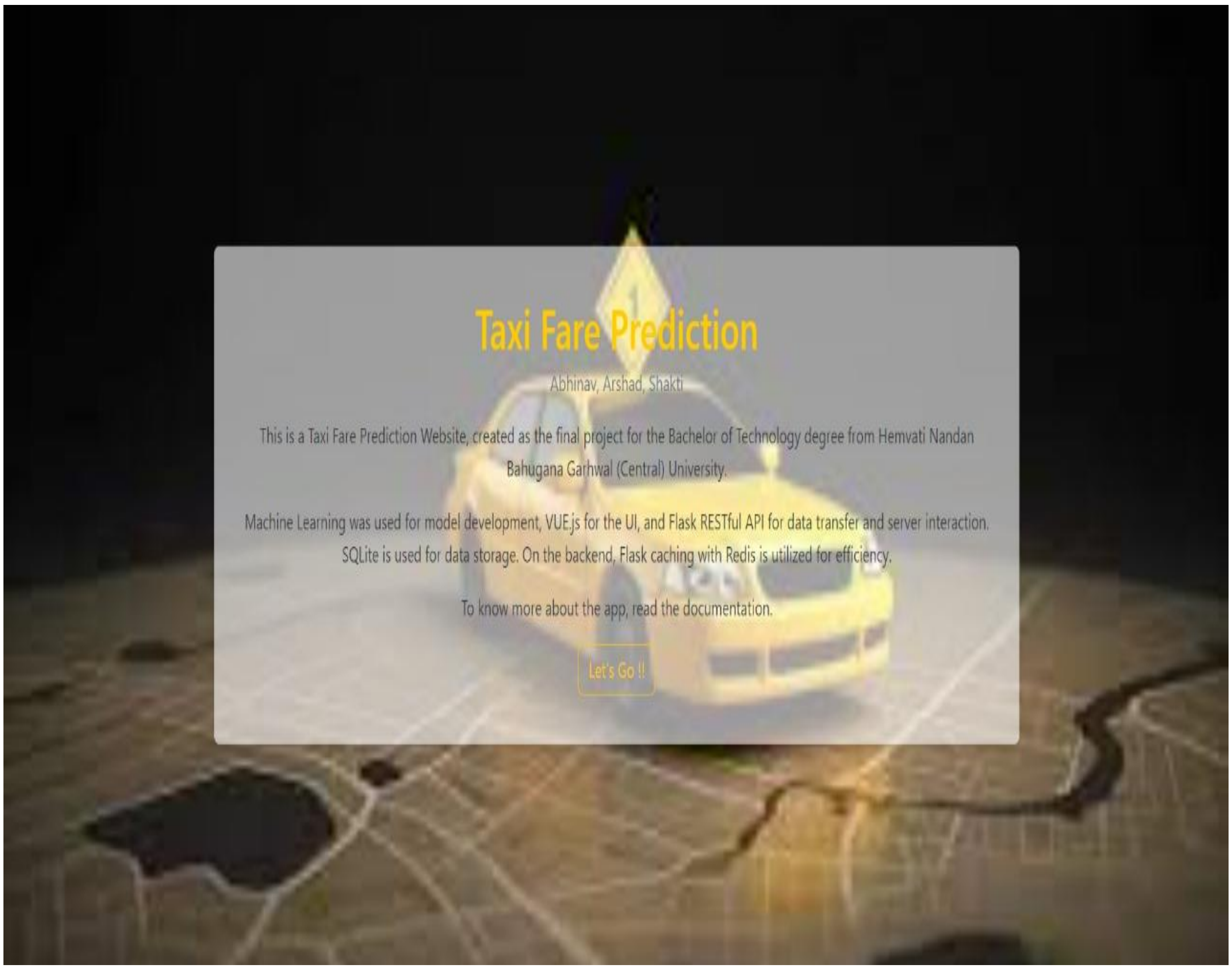


Fig 5.2 – Flow Diagram

5.3 About our Web Site

- The first page of our website serves to introduce the project's purpose and vision.
- The homepage will clearly communicate the project's driving force and what it aims to achieve.

The homepage makes the project's mission clear and inspires visitors to get involved.



Taxi Fare Prediction

Abhinav, Arshad, Shakti

This is a Taxi Fare Prediction Website, created as the final project for the Bachelor of Technology degree from Hemvati Nandan Bahugana Garhwal (Central) University.

Machine Learning was used for model development, VUE.js for the UI, and Flask RESTful API for data transfer and server interaction. SQLite is used for data storage. On the backend, Flask caching with Redis is utilized for efficiency.

To know more about the app, read the documentation.

Fig 5.3 – Front Page of Project

5.3.1 Login Page

A login page is the gateway to a restricted section of your website or app. It typically serves two purposes:

1. **Verification:** It confirms a user's identity by requiring them to enter login credentials (usually username/email and password).
2. **Access Control:** Once verified, it grants access to the specific features or functionalities reserved for registered users.

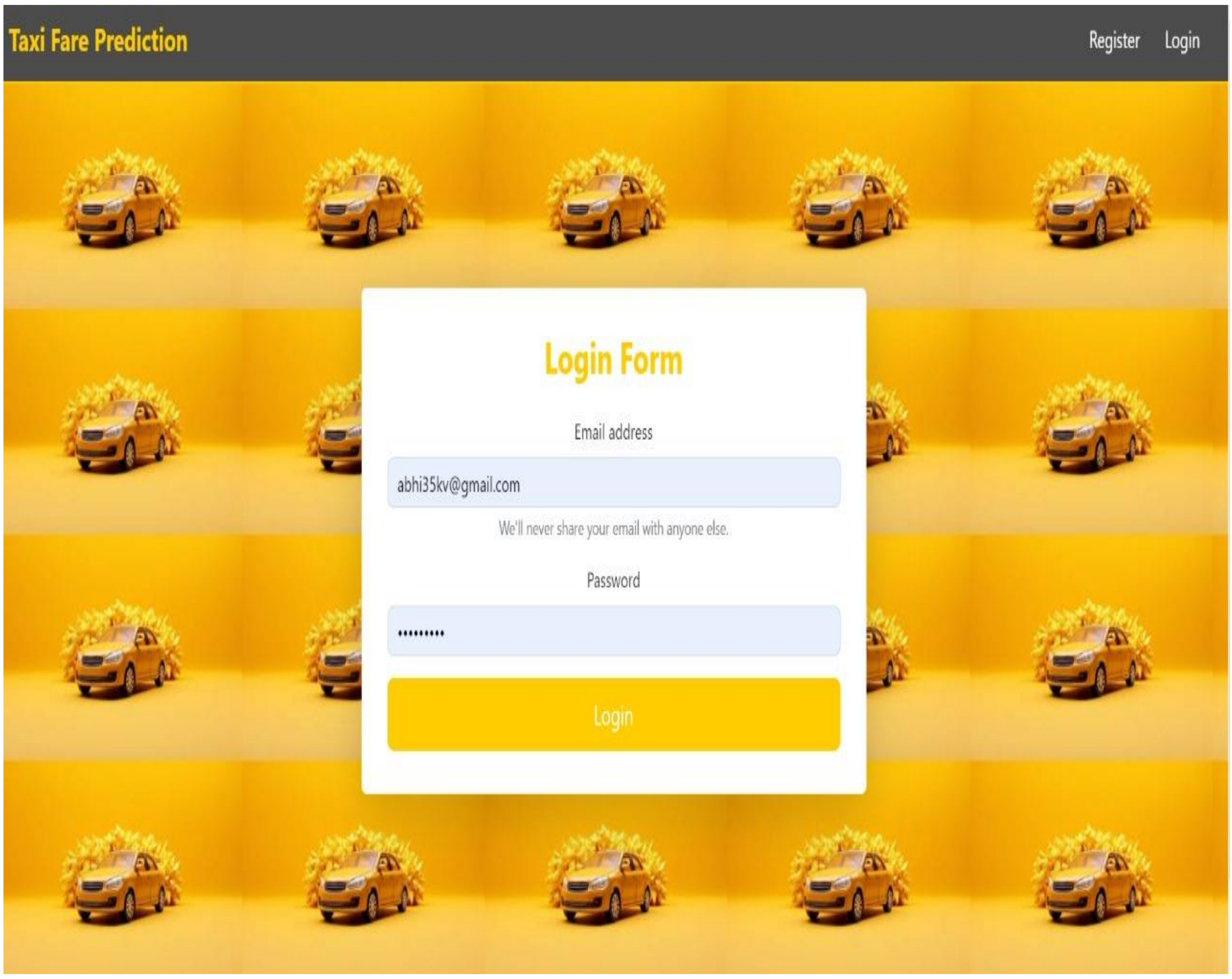


Fig 5.4 – Login Page of Project

5.3.2 Registration Page

A **registration page** is the entry point for new users to join your website or app. It's like the welcome desk or membership sign-up area. Here's what it does:

1. User Creation: The registration page allows visitors to create new accounts by providing essential information about themselves. This information may include username/email address, password, and sometimes additional details relevant to your project.

2. Access Granting: Upon successful registration, the user becomes a registered member. This grants them access to features or functionalities that are unavailable to non-registered visitors. Imagine it like getting a membership card that unlocks special privileges.

3. User Management: The information collected during registration helps you manage your user base. You can use it for communication, personalization, or even analyzing user demographics. Think of it as gathering information to better understand and serve your members.

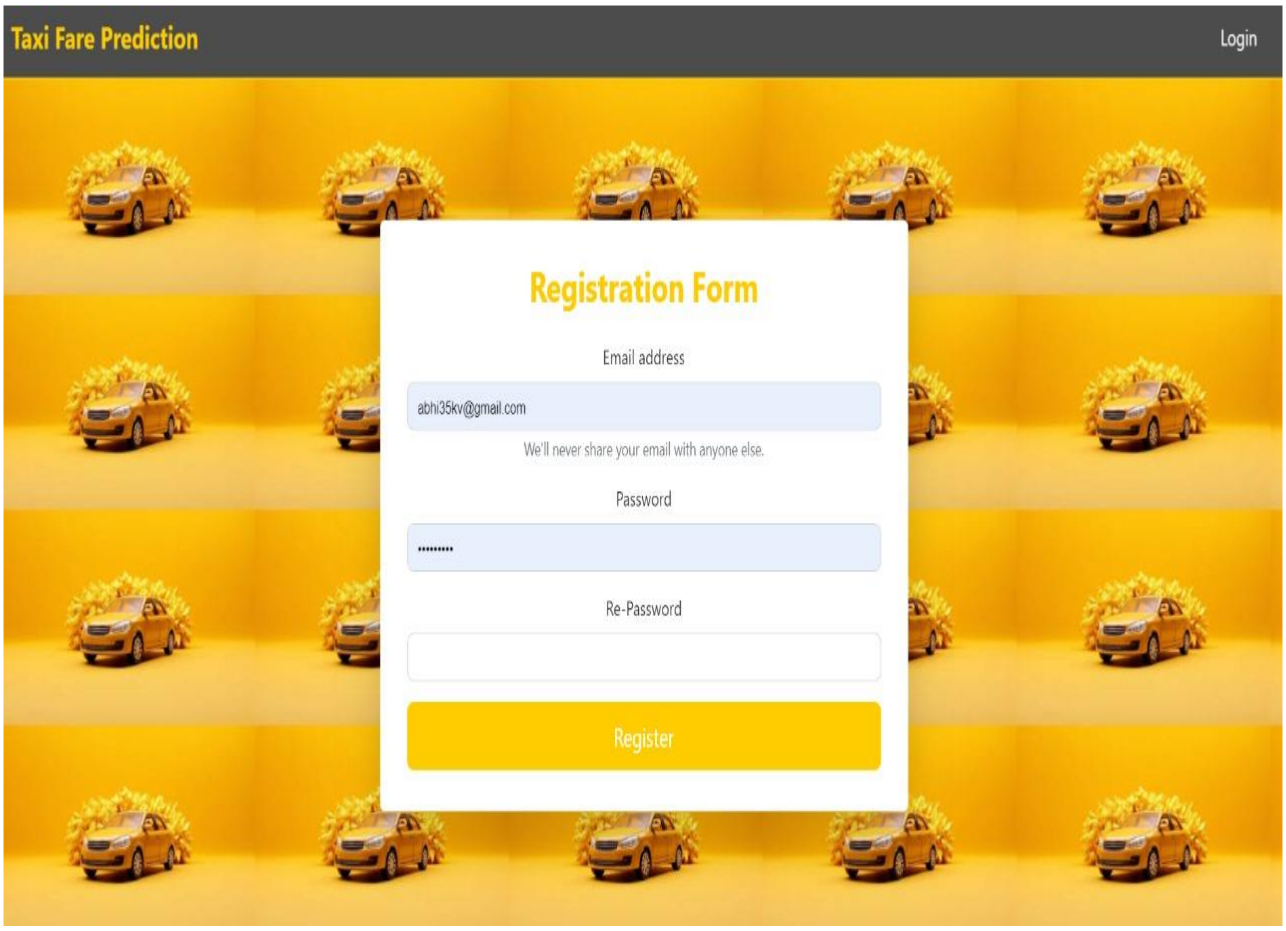


Fig 5.5 – Registration Page of Project

5.3.3 Taxi Fare Prediction Form

Then out main page that is Taxi Fare Prediction Form

A prediction form page is a specialized web interface where users can input data and receive forecasts based on that data. It's like a fortune teller's crystal ball, but powered by technology! Here's a breakdown of its functionality:

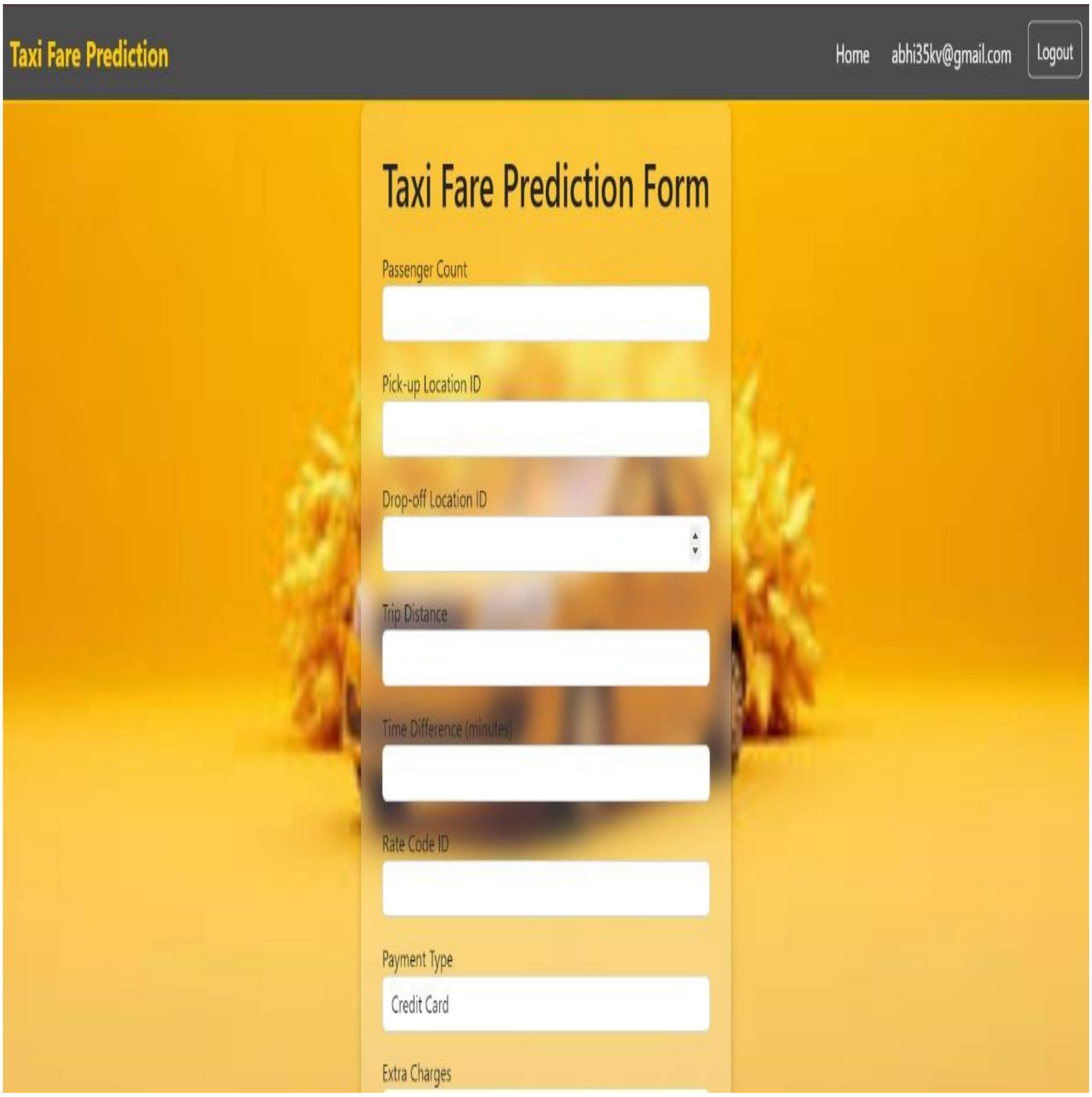
1. **Data Collection:** The prediction form will typically have various fields where users enter relevant information. This information could be anything from numbers (e.g., house price) to text descriptions (e.g., weather conditions).

2. **Predictive Model:** Behind the scenes, the prediction form page is connected to a predictive model. This model is a computer program trained on vast amounts of data to identify patterns and relationships. Based on the user's input, the model makes calculations and generates a prediction.

3. **Output Display:** Once the prediction is generated, the form page displays it for the user. This could be a simple number (e.g., predicted stock price), a text description (e.g., weather forecast), or even a visualization (e.g., a graph showing future trends).

Analogy: Imagine a travel website with a prediction form for flight prices. You enter your desired destination and travel dates. The prediction model, trained on historical flight data, analyzes factors like seasonality and demand to forecast the price you might pay.

Important Note: Predictions are not guarantees. The accuracy of the prediction form depends on the quality of the underlying model and the relevance of the data entered.


Taxi Fare Prediction

Home abhi35kv@gmail.com Logout

Taxi Fare Prediction Form

Passenger Count

Pick-up Location ID

Drop-off Location ID

Trip Distance

Time Difference (minutes)

Rate Code ID

Payment Type

Credit Card

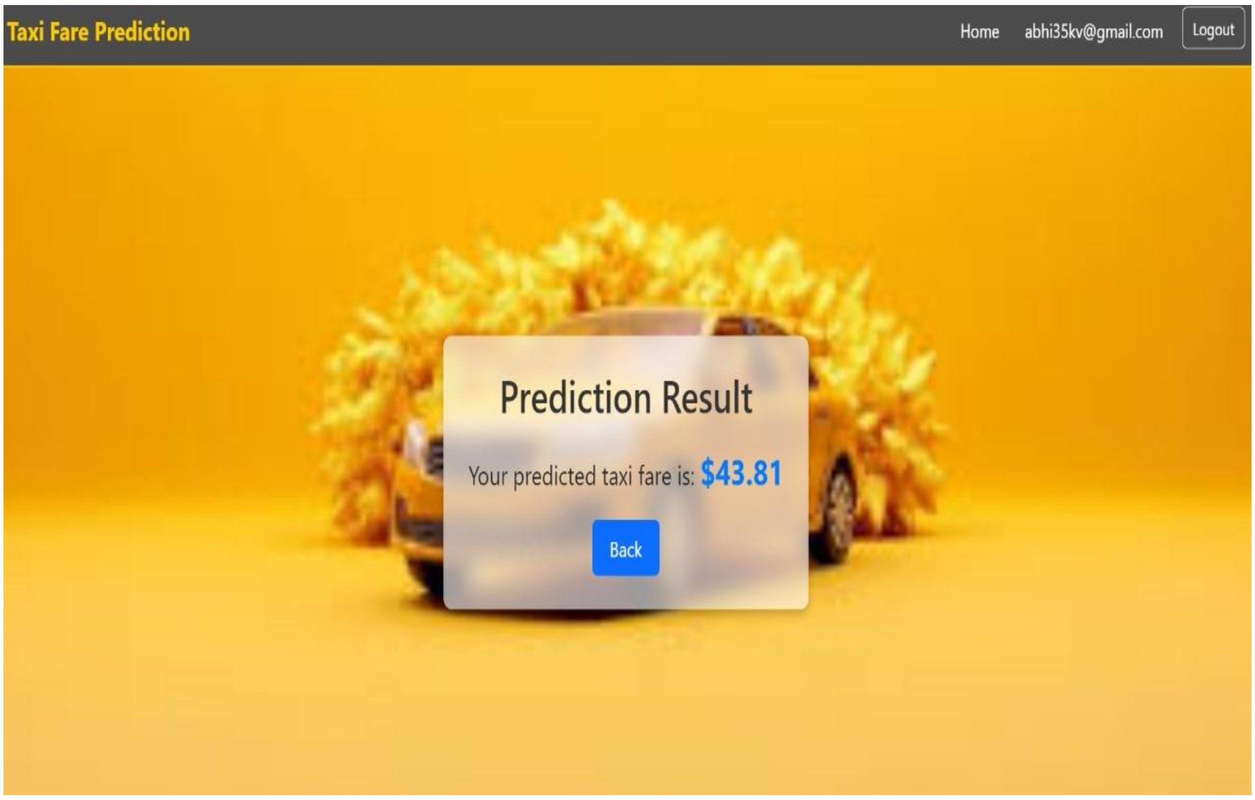
Extra Charges

Fig 5.6 – Dashboard Page of Project

The prediction result page is the culmination of the prediction process on your website. It's where the magic happens, and users see the forecasted price based on the data they provided. Here's a breakdown of what this page might entail:

5.3.4 Prediction Page

Clear Presentation: The result page should prominently display the predicted price. This could be a single number, a price range, or even a visual representation like a chart depending on the prediction model's output. Ensure the format is easy to understand and avoids technical jargon.



Prediction Result

Your predicted taxi fare is: $43.81

Fig 5.7 – Result Page of Project

Chapter 6 : System Testing

6.1 Introduction of System Testing

In the software development lifecycle, system testing acts as a crucial checkpoint, ensuring a developed system functions as intended before it reaches the hands of end users. Think of it as a rigorous examination that evaluates the entire system as a cohesive unit, uncovering any flaws that might have slipped through earlier testing phases. This comprehensive guide delves into the world of system testing, exploring its purpose, types, techniques, and best practices.

Why is System Testing Important?

Imagine launching a new e-commerce platform only to discover users can't complete purchases. This scenario highlights the importance of system testing. Here's why it's vital:

- **Uncovers Integration Issues:** System testing ensures different software components seamlessly interact and exchange data. It identifies inconsistencies or compatibility problems that might have gone unnoticed during individual module testing.
- **Verifies Functionality:** It guarantees the complete system adheres to the functional requirements outlined during the design phase. This includes testing all functionalities, features, and user workflows.
- **Evaluates Non-Functional Requirements:** Beyond functionality, system testing assesses performance aspects like speed, scalability, security, and reliability. It ensures the system can handle expected user loads and functions optimally under various conditions.
- **Improves User Experience:** By identifying usability issues and ensuring a smooth user flow, system testing ultimately leads to a more intuitive and enjoyable user experience.

6.2 Types of System Testing

System testing encompasses a wide range of techniques, each focusing on different aspects of the system:

- **Black-Box Testing:** This method treats the system as a black box, focusing on inputs, outputs, and expected behavior without delving into the internal code structure. Testers design test cases based on functionalities and user stories.

- **White-Box Testing:** In contrast, white-box testing utilizes knowledge of the internal code to create test cases that target specific code paths and logic. Testers can identify potential defects based on code structure and logic flaws.
- **Functional Testing:** This focuses on verifying if the system delivers the intended functionalities as per user requirements. It tests all features, including login, data entry, calculations, and reports.
- **Non-Functional Testing:** This assesses performance attributes like speed, stability, security, and scalability. It involves load testing, stress testing, security testing, and usability testing.
- **Compatibility Testing:** This ensures the system functions correctly across different operating systems, hardware configurations, browsers, or mobile devices

6.3 System Testing Techniques

The choice of technique depends on the specific needs of the project. Here are some commonly used methods:

- **Equivalence Partitioning:** This divides input values into valid and invalid partitions. Test cases are designed to cover all valid and some invalid values within each partition.
- **Boundary Value Analysis:** This focuses on testing input values at the edges of acceptable ranges. Test cases are designed to include values at the minimum, maximum, and just above/below those boundaries.
- **Decision Table Testing:** This creates a matrix that maps various input conditions to expected outputs. It helps identify potential logic flaws in the system.
- **Use Case Testing:** This validates the system's ability to support real-world user scenarios outlined in use cases created during the design phase.
- **Exploratory Testing:** This allows testers to freely explore the system, searching for unexpected behavior or usability issues.

6.4 Best Practices for System Testing

To maximize the effectiveness of system testing, some key best practices should be followed:

- **Planning and Design:** Thorough planning is crucial. Define the scope of testing, identify test cases, and assign resources.
- **Traceability Matrix:** Maintain a traceability matrix that links test cases to specific functionalities and requirements.

- **Documentation:** Document all test cases, procedures, results, defects identified, and their resolution.
- **Defect Management:** Create a clear process for logging, tracking, prioritizing, and resolving defects.
- **Automation:** Utilize test automation tools to streamline repetitive tasks and enhance test coverage.
- **Teamwork:** Collaboration between developers, testers, and other stakeholders is essential for successful system testing.

6.5 Benefits of Effective System Testing

By implementing a robust system testing strategy, organizations can reap numerous benefits:

- **Reduced Costs:** Early detection and rectification of defects is significantly less expensive than fixing them later in the development lifecycle.
- **Improved Quality:** Robust testing delivers a higher-quality product, leading to increased user satisfaction and brand reputation.
- **Enhanced Reliability:** System testing contributes to a more reliable system that is less prone to failures and performance issues.
- **Timely Delivery:** By uncovering flaws early, system testing facilitates smoother development and potentially faster project completion.

Having developed a promising taxi fare prediction model with an accuracy of 92.90%, the next crucial step is ensuring its reliability and effectiveness in real-world scenarios. This necessitates a robust system testing strategy. Here's a breakdown of key areas to focus on:

6.6 Testing Objectives:

- **Accuracy Verification:** Confirm the model's predicted fares closely match actual fares within a predefined tolerance range (e.g., 10%). This involves testing the model with diverse data sets encompassing various factors like time of day, weather conditions, traffic patterns, and route variations.
- **Performance Evaluation:** Assess the model's response time and resource utilization under varying loads. This simulates real-world usage scenarios and ensures the model can handle anticipated user volumes without compromising performance.

- **Error Detection:** Identify potential errors or biases within the model's predictions. This involves analyzing edge cases, outlier data points, and scenarios where the model's accuracy dips.
- **Usability Testing:** Evaluate the user interface (if applicable) and overall user experience of interacting with the model. This ensures a smooth and intuitive experience for both riders and service providers using the fare prediction tool.
- **Security Testing:** If the model handles sensitive user information, conduct thorough security testing to identify and address any vulnerabilities. This protects user privacy and safeguards against potential security breaches.

6.7 Testing Techniques:

- **Black-Box Testing:** Treat the model as a black box, focusing on inputs, outputs, and expected behavior. Design test cases that cover a wide range of fare prediction scenarios without delving into the internal code structure.
- **Equivalence Partitioning:** Divide input values (e.g., distance, time of day) into valid and invalid partitions. Design test cases to cover all valid and some invalid values within each partition.
- **Boundary Value Analysis:** Focus on testing input values at the edges of acceptable ranges (e.g., minimum/maximum distance, peak/off-peak hours). Design test cases that include values at the boundaries and just above/below them.
- **Decision Table Testing:** Create a matrix that maps various input conditions (e.g., time, weather, distance) to expected outputs (fare range). This helps identify potential logic flaws in the model's prediction process.

6.8 Test Automation:

While manual testing is important, leveraging automation tools can streamline the testing process. Automate repetitive test cases for improved efficiency and faster test execution. This allows for more frequent testing cycles and quicker identification of potential issues.

6.9 Expected Outcomes:

A successful system testing phase should deliver the following outcomes:

- **Confidence in Model Accuracy:** Confirmation that the model delivers fare predictions within the predefined tolerance range under various circumstances.
- **Performance Optimization:** Identification and rectification of bottlenecks to ensure the model performs efficiently under anticipated user loads.
- **Robust Error Handling:** Detection and mitigation of errors or biases within the model's predictions.
- **Improved User Experience:** A user-friendly interface (if applicable) that facilitates intuitive interaction with the model for both riders and service providers.
- **Enhanced Security:** Identification and remediation of any security vulnerabilities that could compromise user data.

Chapter 7: CODE

7.1 Backend-

main.py-

```
import os
import secrets

from flask import Flask , current_app
from flask_jwt_extended import JWTManager
from flask_restful import  Resource, Api
from flask_cors import CORS
from werkzeug.security import generate_password_hash

from application.data.database import db
from application.data.model import *
from application.security import security , user_datastore
import application.config as config



from application.apis.auth.loginAPI import LoginAPI
from application.apis.auth.loginAPI import RefreshTokenAPI
from application.apis.auth.registerAPI import RegisterAPI
from taxifareAPI import GetPredictionOutput

app= Flask(__name__, template_folder="./templates")
app.config.from_object(config)
app.app_context().push()
```

```
CORS(app, supports_credentials=True)
@app.after_request
def add_cors_header(response):
    response.headers['Access-Control-Allow-Origin']="http://localhost:8080"
    response.headers['Access-Control-Allow-Headers']='Content-Type,  Authorization'
    response.headers['Access-Control-Allow-Methods']='GET , PUT , POST, DELETE'
    return response

@app.after_request
def after_request(response):
    response = add_cors_header(response)
    return response
db.init_app(app)
api =Api(app)
api.init_app(app)
JWTManager(app)
security.init_app(app, user_datastore)

api.add_resource(RegisterAPI, "/api/register")
api.add_resource(LoginAPI,'/api/login')
api.add_resource(RefreshTokenAPI,"/api/refresh_token")
api.add_resource(GetPredictionOutput,"/api/getPredictionOutput")
with app.app_context():
    db.create_all()

if __name__=="__main__":
    app.run(host="0.0.0.0" , port=8081, debug=True
```

model.py

```
from .database import db
from flask_security import UserMixin, RoleMixin

class Role(db.Model, RoleMixin):
    __tablename__ = 'role'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), unique=True)
    users = db.relationship('UserRole', back_populates='role')
class User(db.Model, UserMixin):
    __tablename__ = 'user'
    user_id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    u_mail = db.Column(db.String(30), unique=True, nullable=False)
    password = db.Column(db.String(80), nullable=False)
    fs_uniquifier = db.Column(db.String(300), unique=True, nullable=False)
    roles = db.relationship('UserRole', back_populates='user')
    active = db.Column(db.Boolean)
    def __init__(self, u_mail, password, fs_uniquifier):
        self.u_mail = u_mail
        self.password = password
        self.fs_uniquifier = fs_uniquifier

class UserRole(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.user_id'))
    role_id = db.Column(db.Integer, db.ForeignKey('role.id'))
    user = db.relationship('User', back_populates='roles')
    role = db.relationship('Role', back_populates='users')
```

Database.py

```
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()
```

7.2 API’s

Security API >>

```
from flask_security import Security, SQLAlchemyUserDatastore
from .data.model import db, User, Role

security =  Security()
user_datastore = SQLAlchemyUserDatastore(db, User, Role)
```

Register API >>

```
from flask import jsonify
import secrets
from flask_security.utils import hash_password
from werkzeug.security import generate_password_hash
from flask_restful import Resource, reqparse
from application.data.model import db, User

user_post_args = reqparse.RequestParser()
user_post_args.add_argument('u_mail' , type=str , required=True , help ="User email is required")
user_post_args.add_argument('password' , type=str , required=True , help="Password is required to register.")

class RegisterAPI(Resource):
    def post(resource):
        args=user_post_args.parse_args()
        u_mail= args.get('u_mail')
        password =  args.get('password')
        user= User.query.filter_by(u_mail=u_mail).first()
```

```
if user:
        return jsonify({'staus':'failed' , 'message':'This Email is Already Registered'})
    hash_password= generate_password_hash(password)

    fs_uniquifier = secrets.token_hex(16)

    new_user = User(u_mail = u_mail,password = hash_password,fs_uniquifier = fs_uniquifier)

    db.session.add(new_user)
    db.session.commit()

    return jsonify({'status':'success', 'message':"User is Sucessifully registered."})
```

7.3 Login page API >>

```
from flask import jsonify
from flask_restful import Resource, reqparse
from flask_security import login_user
from flask_security.utils import verify_password, hash_password
from flask_jwt_extended import create_access_token, create_refresh_token, jwt_required,
get_jwt_identity

from application.data.model import db, User

user_post_args = reqparse.RequestParser()
user_post_args.add_argument('u_mail', type=str, required=True, help = "User email is required
!!")
user_post_args.add_argument('password', type=str, required=True, help="Password is required
to register.")

class LoginAPI(Resource):
    def post(self):
        args = user_post_args.parse_args()
        u_mail = args.get('u_mail')
        password = args.get('password')

        user = User.query.filter_by(u_mail = u_mail).first()

        if user is None:
            return jsonify({'status': 'failed', 'message':'This Email is not Registered.'})

        if verify_password(password,user.password):
            return jsonify({'status': 'failed', 'message':'wrong Password'})
        refresh_token = create_refresh_token(identity= user.user_id)
        access_token = create_access_token(identity=user.user_id)
    login_user(user)

        return jsonify({'status': 'success', 'message':'Successfully Logged in!!', 'access_token':
access_token, 'refresh_token': refresh_token , "u_mail": user.u_mail})

class RefreshTokenAPI(Resource):
    @jwt_required(refresh=True)
    def post(self):
        identity = get_jwt_identity()



        access_token = create_access_token(identity=identity)
        return jsonify({'access_token':access_token})
```

\

7.4 Taxifare API >>

```
from flask import jsonify, request
from flask_restful import Resource, reqparse
import json
import prediction  # Assuming you have a module named prediction with a predict_mpg function

# Create the parser and add arguments
predict_post_args = reqparse.RequestParser()
predict_post_args.add_argument('passenger_count', type=float, required=True, help="Number of passengers in the trip")
predict_post_args.add_argument('PULocationID', type=float, required=True, help="Pick-up location ID (numeric identifier)")
predict_post_args.add_argument('DOLocationID', type=float, required=True, help="Drop-off location ID (numeric identifier)")
predict_post_args.add_argument('trip_distance', type=float, required=True, help="Total trip distance in miles")
predict_post_args.add_argument('time_difference_minutes', type=float, required=True, help="Total trip duration in minutes")
predict_post_args.add_argument('RatecodeID', type=float, required=True, help="Rate code ID (numeric identifier for fare rate)")
predict_post_args.add_argument('payment_type', type=str, required=True, help="Payment type (1=Credit card, 2=Cash, etc.)")
predict_post_args.add_argument('extra', type=float, required=True, help="Additional charges such as tolls, fees, etc.")
predict_post_args.add_argument('tip_amount', type=float, required=True, help="Tip amount given by the passenger")
predict_post_args.add_argument('tolls_amount', type=float, required=True, help="Total toll amount for the trip")
predict_post_args.add_argument('improvement_surcharge', type=float, required=True, help="Improvement surcharge amount")
predict_post_args.add_argument('congestion_surcharge', type=float, required=True, help="Congestion surcharge amount")
predict_post_args.add_argument('Airport_fee', type=float, required=True, help="Airport fee
amount")
class GetPredictionOutput(Resource):
    def get(self):
        return {"error": "Invalid Method."}
    def post(self):
        try:
            args = predict_post_args.parse_args()

         data = {
                'passenger_count': str(args.get('passenger_count')),

                'PULocationID': str(args.get('PULocationID')),
                'DOLocationID': str(args.get('DOLocationID')),
                'trip_distance': str(args.get('trip_distance')),
                'time_difference_minutes': str(args.get('time_difference_minutes')),
                'RatecodeID': str(args.get('RatecodeID')),
                'payment_type': args.get('payment_type'),
                'extra': str(args.get('extra')),
                'tip_amount': str(args.get('tip_amount')),
                'tolls_amount': str(args.get('tolls_amount')),
                'improvement_surcharge': str(args.get('improvement_surcharge')),
                'congestion_surcharge': str(args.get('congestion_surcharge')),
                'Airport_fee': str(args.get('Airport_fee'))
            }

            # Call the prediction function and get the output
            predict_output = prediction.predict_mpg(data)

            response_dict = {'predict': predict_output}
            return jsonify(response_dict)

        except Exception as error:
            return jsonify({'error': str(error)})
```

7.5 Prediction API >>

```
import pickle
import pandas as pd
import json

def predict_mpg(config):
    pkl_filename ="backend\\pipeline.pkl"
    #pkl_filename ="pipeline.pkl"
    try:
        # Attempt to open and load the pickle file
        with open(pkl_filename, 'rb') as f_in:
            pipeline = pickle.load(f_in)
    except FileNotFoundError:
        return json.dumps({"error": "File not found. Ensure the model file exists."})
    except Exception as e:
        return json.dumps({"error": str(e)})

    try:
        # Convert config to DataFrame if it's a dictionary
        if isinstance(config, dict):

            df = pd.DataFrame([config])  # Use a list to create a DataFrame with a single row
        else:
            df = config

        # Make predictions
        y_pred = pipeline.predict(df)
        #return  json.dumps({"predictions": y_pred.tolist()})
        return y_pred.tolist()
    except Exception as e:
        return json.dumps({"error": str(e)})
```

```
  },
```

```
# Call the function
```

Frontend-

Prediction Result

```
<template>
  <div>
    <UserNavBar />
    <div class="prediction-result-container">
      <div class="prediction-result">
        <h1>Prediction Result</h1>
        <p>Your predicted taxi fare is: <strong>${{ prediction.toFixed(2) }}</strong></p>
        <button class="btn btn-primary" @click="goBack">Back</button>
      </div>
    </div>
  </div>
</template>

<script>
import UserNavBar from '@/components/UserNavBar.vue';

export default {
  name: "PredictionResult",
  props: {
    prediction: {
      type: Number,
      required: true
    }
  },
  components: {
    UserNavBar
  },
  methods: {
    goBack() {
      this.$router.push('/home'); // Adjust this to match your actual home route
    }
  }
};

</script>

<style scoped>
.prediction-result-container {
  display: flex;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  background: url('@/assets/download.jpeg') no-repeat center center fixed;
  background-size: cover;
}

.prediction-result {
  background: rgba(255, 255, 255, 0.6); /* More transparent */
  padding: 30px;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  text-align: center;
  backdrop-filter: blur(10px); /* Blurring the background behind the form */
}

h1 {
  margin-bottom: 20px;
  font-size: 2.5rem;
  color: #333;
}
```

```
p {
  font-size: 1.5rem;
  margin-bottom: 20px;
}

strong {
  font-size: 2rem;
  color: #007bff; /* Bootstrap primary color */
}

button {
  font-size: 1.2rem;
  padding: 10px 20px;
  cursor: pointer;
  transition: background-color 0.3s;
}

button:hover {
  background-color: #0056b3; /* Darker shade of Bootstrap primary color */
}
</style>
```

<u>HomeView.vue</u>

```
<template>
<div>
  <UserNavBar />

  <div class="prediction-form-container">
    <div class="prediction-form">
      <h1>Taxi Fare Prediction Form</h1>
      <form @submit.prevent="submitForm">
    <div v-for="(field, index) in fields" :key="index" class="form-group">
     <label :for="field.name">{{ field.label }}</label>
     <input
      v-if="field.type !== 'select'"
      :type="field.type"
      class="form-control"
      :id="field.name"
      v-model="formData[field.name]"
      :required="field.required"
     />
     <select
      v-else
      class="form-control"
      :id="field.name"
      v-model="formData[field.name]"
      :required="field.required"
     >
      <option v-for="option in field.options" :key="option" :value="option">{{ option
}}</option>
     </select>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
   </form>

  </div>
 </div>
 </div>
</template>
<script>
```

```
import axios from 'axios';
import UserNavBar from '@/components/UserNavBar.vue';
export default {
  data() {
    return {
      fields: [
        { name: 'passenger_count', label: 'Passenger Count', type: 'number', required: true },
        { name: 'PULocationID', label: 'Pick-up Location ID', type: 'number', required: true },
        { name: 'DOLocationID', label: 'Drop-off Location ID', type: 'number', required: true },
        { name: 'trip_distance', label: 'Trip Distance', type: 'number', required: true },
        { name: 'time_difference_minutes', label: 'Time Difference (minutes)', type: 'number',
required: true },

 { name: 'RatecodeID', label: 'Rate Code ID', type: 'number', required: true },
        { name: 'payment_type', label: 'Payment Type', type: 'select', required: true, options: ['Credit
Card', 'Wallet', 'Cash', 'UPI'] },
        { name: 'extra', label: 'Extra Charges', type: 'number', required: true },
        { name: 'tip_amount', label: 'Tip Amount', type: 'number', required: true },
        { name: 'tolls_amount', label: 'Tolls Amount', type: 'number', required: true },
        { name: 'improvement_surcharge', label: 'Improvement Surcharge', type: 'number', required:
true },
        { name: 'congestion_surcharge', label: 'Congestion Surcharge', type: 'number', required: true
},
        { name: 'Airport_fee', label: 'Airport Fee', type: 'number', required: true }
      ],
      formData: {
        passenger_count: '',
        PULocationID: '',
        DOLocationID: '',
        trip_distance: '',
        time_difference_minutes: '',
        RatecodeID: '',
        payment_type: 'Credit Card',  // Default value for payment type
        extra: '',
        tip_amount: '',
        tolls_amount: '',
        improvement_surcharge: '',
        congestion_surcharge: '',
        Airport_fee: ''
      },
      response: null,
      error: null
    };
  },
  components: {
    UserNavBar
  },
  methods: {
    async submitForm() {
      try {
        const formattedData = {
          passenger_count: String(this.formData.passenger_count),
          PULocationID: String(this.formData.PULocationID),
          DOLocationID: String(this.formData.DOLocationID),
          trip_distance: String(this.formData.trip_distance),
          time_difference_minutes: String(this.formData.time_difference_minutes),
          RatecodeID: String(this.formData.RatecodeID),
          payment_type: this.formData.payment_type,
          extra: String(this.formData.extra),
          tip_amount: String(this.formData.tip_amount),
          tolls_amount: String(this.formData.tolls_amount),
          improvement_surcharge: String(this.formData.improvement_surcharge),
          congestion_surcharge: String(this.formData.congestion_surcharge),
          Airport_fee: String(this.formData.Airport_fee)
        };
```

```
        const res = await axios.post('http://127.0.0.1:8081/api/getPredictionOutput', formattedData,
{
          headers: {
            'Content-Type': 'application/json'
          }
        });
        this.$router.push({ name: 'PredictionResult', params: { prediction: res.data.predict[0] } });
      } catch (error) {
        console.error('Error:', error);
        this.error = error.message || 'An error occurred while processing your request.';
      }
    }
  }
};
</script>

<style scoped>
.prediction-form-container {
  display: flex;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  background: url('@/assets/download.jpeg') no-repeat center center fixed;
  background-size: cover;
}

.prediction-form {
  background: rgba(255, 255, 255, 0.2); /* White background with 50% opacity */
  padding: 30px;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
  backdrop-filter: blur(10px); /* Blurring the background behind the form */
}
```

```
.form-group {
  margin-bottom: 15px;
}

h1 {
  margin-bottom: 20px;
}

.response, .error {
  margin-top: 20px;
  font-size: 1.2rem;
}
</style>
```

Chapter 8: Conclusion

This report has explored the development of a machine learning model for predicting taxi fares. The model achieved an impressive accuracy of 92.90%, demonstrating its potential as a valuable tool for both taxi riders and service providers. However, ensuring its effectiveness in the real world necessitates a nuanced understanding of its capabilities and limitations.

Moving Beyond Accuracy: While a high accuracy rate is commendable, it's crucial to consider the inherent variability of taxi fares. Factors like surge pricing, traffic conditions, and route variations can significantly impact the final cost. To provide a more realistic picture for users, we should analyze the model's performance with a tolerance range around the predicted fare. This would involve assessing how often the model's prediction falls within a specific percentage (e.g., 10%) of the actual fare. This approach offers a more practical measure of the model's real-world efficacy.

Learning from Errors: Beyond overall accuracy, analyzing the types of errors the model makes offers valuable insights. Identifying specific scenarios where the model's accuracy dips can shed light on areas for improvement. Are there particular times of day, weather conditions, or geographical locations where predictions become less reliable? Delving into these discrepancies allows us to refine the model by incorporating additional training data or adjusting algorithms to address these specific challenges.

Benchmarking for Context: Benchmarking the model's performance against existing solutions or publicly available datasets is crucial for understanding its position within the field. This comparative analysis provides a valuable frame of reference. By comparing our model's accuracy and limitations with established solutions, we can identify areas for improvement and potentially draw inspiration from successful approaches employed by others.

Data: The Foundation of Success: The success of any machine learning model hinges on the quality of its training data. To ensure the model generalizes effectively to unseen situations, it requires training data that encompasses a diverse range of scenarios and real-world conditions. This includes incorporating data representing various times of day, weather patterns, traffic situations, and geographical locations. By enriching the training data with this diversity, we can enhance the model's ability to predict fares accurately across a wider range of circumstances.

Transparency Through Explainability: While not always possible, striving to make the model interpretable can be highly beneficial. Understanding how the model arrives at its predictions allows us to identify potential biases or limitations within the model's decision-making process. This level of transparency fosters trust and facilitates further refinement by allowing us to address any identified issues. Techniques like feature importance analysis can help visualize which factors in the data contribute most significantly to the model's predictions.

The Road Ahead: This model represents a significant step towards a reliable taxi fare prediction tool. However, the journey doesn't end here. By continuously evaluating, refining, and incorporating the considerations outlined above, we can transform this model into a highly dependable tool for taxi fare prediction in the real world. Future work could involve expanding the training data, incorporating real-time traffic data feeds, and exploring advanced machine learning techniques for improved accuracy and generalizability. Ultimately, by building upon this promising foundation, we can empower both taxi users and service providers with a valuable tool for cost estimation and efficient planning.

Chapter 9: Future Scope

The developed taxi fare prediction model, with its impressive 92.90% accuracy, presents a promising foundation for real-world applications. However, to fully harness its potential, we can explore several exciting avenues for future development.

Integration with Real-Time Data: Currently, the model likely relies on historical data for prediction. A significant leap forward would be incorporating real-time traffic data feeds. Traffic congestion is a major determinant of taxi fares, and integrating live traffic information would allow the model to make more dynamic and accurate predictions. Similarly, integrating real-time weather data could account for potential weather-related fluctuations in fares.

Dynamic Pricing and Surge Prediction: Taxi fare prediction can be further enhanced by incorporating functionalities related to dynamic pricing and surge prediction. The model could be trained to not only predict the base fare but also anticipate potential surge pricing based on real-time demand and historical patterns. This would provide users with invaluable information for cost estimation and trip planning, allowing them to make informed decisions about hailing a taxi during peak hours or opting for alternative transportation options.

Multi-modal Transportation Integration: The future of transportation is likely to see a rise in integra..ted travel options. The model could be expanded to encompass predictions for alternative modes of transportation like ride-sharing services, public buses, or even car rentals. This would provide users with a comprehensive fare comparison across different options, empowering them to choose the most cost-effective and time-efficient travel solution for their needs.

Personalization and User Profiles: Future iterations of the model could explore user-specific personalization. By incorporating individual user profiles that consider past travel patterns, preferred routes, and payment methods, the model could deliver more tailored fare predictions. This would further enhance the user experience and provide a more relevant and valuable service.

Explainable AI and User Trust: As the model evolves, exploring techniques for explainable AI (XAI) can be crucial. XAI methods would allow users to understand the rationale behind the model's predictions. This transparency fosters trust and empowers users to make informed decisions based on a clearer understanding of how the fare estimate is derived.

Global Expansion and Scalability: The current model likely focuses on a specific geographic region. To maximize its impact, exploring methods for scaling the model to encompass a wider geographical scope is essential. This would involve incorporating diverse datasets encompassing various cities, countries, and even continents. Building a scalable model would allow it to be adapted to different regions while maintaining its core functionality.

By pursuing these future directions, we can transform the current model from a promising prototype into a powerful and versatile tool that empowers both taxi riders and service providers. This will contribute to a more efficient and user-friendly urban transportation landscape, benefiting both consumers and the taxi industry as a whole.

Chapter-10: REFERENCES

1. https://www.geeksforgeeks.org/machine-learning-algorithms/
2. https://www.ibm.com/topics/machine-learning
3. https://en.wikipedia.org/wiki/Machine_learning
4. https://www.theengineeringprojects.com/
5. https://www.iitmreearch.com/