PREMIER UNIVERSITY, CHATTOGRAM

Department of Computer Science & Engineering

Report

On

”Food Delivery Time Prediction Using Machine Learning Algorithm”

SUBMITTED BY

Name: Sohela Showrin

ID: 2104010202199

Name: Rahin Toshmi Ohee

ID: 2104010202204

In partial fulfillment for the degree of

Bachelor of Science in Computer Science & Engineering under

the Supervision of

Adiba Ibnat Hossain

Lecturer

Department of Computer Science & Engineering

Premier University, Chattogram

10 September, 2024

Table of Contents

List of Figures

1 Introduction

In today's world, online food delivery services are becoming increasingly popular. Customers want their food delivered quickly and accurately. However, many food delivery companies struggle to provide precise delivery times due to various unpredictable factors such as traffic, weather, and the time it takes for restaurants to prepare food. Currently, most food delivery platforms use simple methods to estimate delivery times, such as historical data and average delivery times. These methods do not consider real-time changes and can often lead to inaccurate predictions, resulting in unhappy customers and inefficiencies in the delivery process.The primary goal of this project is to develop a predictive model for estimating the delivery time of food orders using Python. By analyzing historical data on delivery times along with various factors such as distance, the age of the delivery person, and delivery person ratings, the model aims to provide accurate predictions that can help in optimizing delivery operations.

1.1 Purpose of the Work

In the competitive landscape of food delivery services, timely deliveries are crucial for customer satisfaction and retention. Delays can lead to negative reviews and loss of business. Predicting delivery times accurately allows service providers to better manage their resources, allocate delivery personnel more efficiently, and enhance overall service quality. This predictive capability is particularly valuable in dynamically changing environments where traffic, weather, and order volumes can vary significantly.

1.2 Background of the Work

Predictive modeling in logistics and delivery services is now widely used because of improvements in machine learning and data analysis. Traditional methods of estimating delivery times often rely on simplistic and static rules, which fail to account for the complexities and variabilities of real-world conditions. Leveraging machine learning techniques enables the creation of more sophisticated models that can learn from past data and make informed predictions.

1.3 Motivation

In the competitive landscape of food delivery services, timely deliveries are crucial for customer satisfaction and retention. Delays can lead to negative reviews and loss of business. Predicting delivery times accurately allows service providers to better manage their resources, allocate delivery personnel more efficiently, and enhance overall service quality. This predictive capability is particularly valuable in dynamically changing environments where traffic, weather, and order volumes can vary significantly. In previous time customer are not aware of the delivery time of their ordered food. The motivation for this project stems from the need to enhance operational efficiency, improve customer satisfaction, and maintain a competitive edge in the rapidly growing food delivery market.

1.4 Optimal Perfomance And Selected Opinion

Among the models evaluated, Random Forests emerged as the best performing due to their ability to handle complex, non-linear relationships and interactions between the input features and the target variable (delivery time). Random Forests also provide robustness against overfitting and are capable of handling large datasets with high dimensionality, making them well-suited for this application.

The choice to use Random Forests was based on comprehensive model evaluation and comparison. Metrics such as Mean Squared Error and R-squared indicated that Random Forests provided superior predictive accuracy compared to Linear Regression and K-Nearest Neighbors. Additionally, the interpretability and feature importance insights offered by Random Forests facilitated better understanding and optimization of delivery processes.

1.5 Objectives

The primary objectives of this project are:

- To design a model that can predict food delivery time accurately.
- To reduce customer waiting times by providing accurate delivery time estimates.
- To see the impact of delivery time predictions on customer satisfaction.
- To create a simple and easy-to-use platform or model where both restaurants and customers can check estimated delivery times.
- To analyze and utilize historical delivery data to improve prediction accuracy.

Predicting food delivery times with machine learning can improve customer satisfaction and how well the delivery service runs. By using past data and smart prediction models, food delivery companies can give more accurate delivery times. This helps make customers happier. Real-time data will make these predictions even more accurate.

2 Related works

There are few contributions regarding accurate food delivery time prediction systems. Among other systems, it has become necessary to find a robust solution that can help reduce delivery time variability, provide real-time predictions, and enhance the efficiency of the entire delivery process. This system should accurately account for various factors such as traffic, weather, and restaurant preparation times, ultimately leading to improved customer satisfaction and operational efficiency for food delivery services.

Hirschberg, C., A. Rajko, and M. W. Thomas Schumacher (2016) implements the changing market for food delivery. They has investigated two types of online platforms have risen to fill that void. The first type is the “aggregators,” which emerged roughly 15 years ago; the second is the “new delivery” players, which appeared in 2013. Both allow consumers to compare menus, scan and post reviews, and place orders from a variety of restaurants with a single click. The aggregators, which are part of the traditional-delivery category, simply take orders from customers and route them to restaurants, which handle the delivery themselves. In contrast, the new-delivery players build their own logistics networks, providing delivery for restaurants that don’t have their own drivers [1]

Luhtala, R. (2023). Quantifying the effects of external predictors on food delivery platform order volumes (Master’s thesis).[2]

Coelho et al. (2016) describe a solution to optimize lunch breaks in furniture delivery tours but take the duration of the time windows for assembly as a fixed and predetermined value[3].

In the field of logistics, the planning of service times is often only a part of the larger vehicle routing problem (VRP), which is based on the traveling salesman problem (TSP).There exist many different variants of this problem when it comes to how services or deliveries are executed at geographically separated locations. The classic problem of reaching a number of nodes by the shortest possible way was well described by Laporte (1992)[4].
A variation of this basic version is the consideration of time windows which specify when the traveler or the vehicle has to arrive at a specific node (Ahn and Shin 1991) [5] . This is related to the situation discussed in this paper, as the vehicle routing problem with time windows (VRPTW) considers the duration that a vehicle has to stand at a stop. Calculating the duration of this stop time is generally considered a different problem. Other noticeable variants of the vehicle routing problem include the consideration of multi-vehicle problems [6], mixed pickup and delivery [7] or nodes which need to be visited in specific orders [8].
The problem of having variable-timed stops at each node was described by Vo¨ssing (2017) in an attempt to show why many companies still rely on human dispatcher knowledge for tour planning and time estimations [9] .
The variations in the required time are complex and require a case-by-case analysis of the underlying reasons and data. Extrapolation from one problem domain to another is often not possible, because the problems have been simplified too much to be able to implement them in the real world [10].
Related works on food delivery time prediction have limitations and significant scope for development. Most existing systems use basic algorithms and historical data, but they do not adequately address real-time factors such as traffic, weather, and restaurant preparation times. Additionally, while machine learning has been applied to delivery time predictions, there is still a lack of comprehensive solutions that integrate all relevant data sources effectively. Handling the dynamic nature of real-time data, such as sudden traffic changes or unexpected delays in food preparation, remains a challenging task. Developing a system that can accurately process and analyze this data in real-time to provide precise delivery time predictions is essential for improving customer satisfaction and operational

efficiency in food delivery services. There is a growing interest in food delivery time prediction.In the past years,customers and restaurants people do not accurately precdict the delivery time.,but an ai based model can easily predict the time by using previous famous dataset.There are some algorithm that can help the model to learn data sets and then it will give some result basen on the training set that the model learn before.
To predict the food delivery time in real time, we need to calculate the distance between the food preparation point and the point of food consumption. After finding the distance between the restaurant and the delivery locations, we need to find relationships between the time taken by delivery partners to deliver the food in the past for the same distance [11].
There are some key factors that influence delivery time:

1. Real-time traffic data significantly impacts delivery time predictions. Models often incorporate traffic data to adjust predictions dynamically [12].

2. Weather can affect delivery times by influencing both the speed of delivery and the volume of orders [12].

3. The number of orders and their distribution within a delivery area are critical factors. Higher order volumes can lead to delays due to longer preparation times and increased delivery stops [13].

4. The variability in preparation times across different restaurants is a significant factor. Some models integrate historical preparation time data to enhance prediction accuracy.[14]

5. The experience and efficiency of delivery drivers, along with optimal routing algorithms, are important considerations [3].

Machine learning has proven to be a valuable tool in predicting food delivery times, offering substantial improvements over traditional methods. Ongoing advancements in ML algorithms and the integration of real-time data sources are expected to further enhance the accuracy and reliability of delivery time predictions [15].

3 Methodology

3.1 Overview of the Workflow

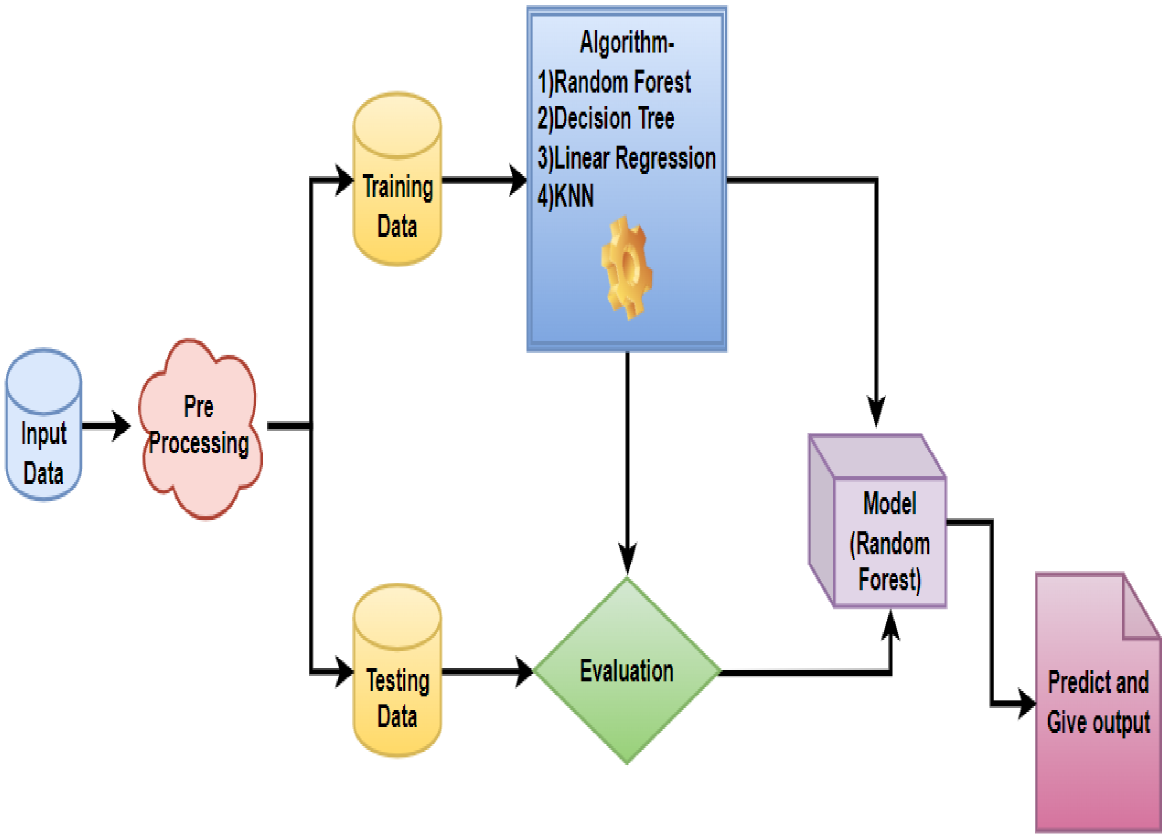


Figure 3.1: Workflow Diagram

3.2 Figure Insight and Analysis

The diagram shows the steps involved in a machine learning project aimed at predicting food delivery times. Here's a simple breakdown of each part of the process:

1. Input Data

- This is the initial data collected, which includes various features that may affect delivery times, such as order time, delivery location, restaurant location, traffic conditions, weather, etc.
- The raw data that will be used to train and test the machine learning models.

2. Pre Processing

- This step involves cleaning and transforming the raw input data to make it suitable for training and testing machine learning models. It can include tasks such as handling missing values, encoding categorical variables, normalizing/standardizing data, and feature engineering.
- Ensure that the data is in the correct format and of high quality for model training.

3. Training Data

- This is the subset of the pre-processed data used to train the machine learning models. Typically, the data is split into training and testing sets.
- The training data is used to teach the models to learn the relationships between the features and the target variable (delivery time).

4. Testing Data

- This is the subset of the pre-processed data used to evaluate the performance of the trained models.
- The testing data is used to test the model's predictions and assess its performance on unseen data.

5. Algorithm

- Different machine learning algorithms are used to build predictive models. In this case, the algorithms used are:
  - **Random Forest:** An ensemble learning method based on decision trees.
  - **Decision Tree:** A model that predicts the target by learning simple decision rules inferred from the data features.
  - **Linear Regression:** A linear approach to modeling the relationship between the target and one or more features.
  - **K-Nearest Neighbors (KNN):** A non-parametric method used for classification and regression by comparing the target to the closest training examples in the feature space.
- These algorithms are trained on the training data to create predictive models.

6. Evaluation

- This step involves assessing the performance of the models using various metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or R-squared. The evaluation helps in understanding how well the models are performing.
- Evaluate and compare the performance of the different models to select the best one for predicting food delivery times.

7. Model (Random Forest)

- After evaluation, the Random Forest model is selected as the final model based on its performance.
- The trained Random Forest model is used for making predictions on new, unseen data.

8. Predict and Give Output

- This is the final step where the selected model (Random Forest) is used to predict food delivery times for new orders.
- Provide the predicted delivery times to end-users or integrate them into the food delivery system to optimize delivery processes.

3.3 Algorithms Analysis

1.K-Nearest Neighbors (KNN):

- **Description:** KNN is a non-parametric, instance-based learning algorithm. It predicts the target variable (delivery time) based on the average of the k-nearest data points in the feature space.
- **Process:** For each new data point, the algorithm finds the k-nearest neighbors in the training dataset based on a distance metric (e.g., Euclidean distance) and assigns the average value of the target variable of these neighbors as the prediction.
- **Strengths:** Captures non-linear relationships and patterns without assuming any specific functional form.
- **Limitations:** Computationally intensive for large datasets and sensitive to the choice of k.

2. Linear Regression:

- **Description:** Linear Regression is a parametric algorithm that models the relationship between the independent variables (features) and the dependent variable (delivery time) by fitting a linear equation to the observed data.
- **Process:** The algorithm calculates the coefficients that minimize the sum of squared residuals between the observed and predicted values.
- **Strengths:** Simple, interpretable, and efficient for linear relationships.

- **Limitations:** Assumes a linear relationship, which may not capture the complexities of the data.

3.Decision Tree:

- **Description:** A Decision Tree is a non-parametric supervised learning algorithm used for classification and regression tasks. It splits the data into branches based on feature values, leading to a tree-like structure.
- **Process:** The algorithm recursively splits the data into subsets based on the feature that results in the highest information gain or lowest Gini impurity, depending on the criteria used.
- **Strengths:** Captures non-linear relationships and interactions between features.
- **Limitations:** Prone to overfitting and can create complex models that are hard to interpret.

4.Random Forest:

- **Description:** Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the average prediction of the individual trees.
- **Process:** The algorithm creates several decision trees using different subsets of the training data and features, and then averages their predictions to improve accuracy and robustness.
- **Strengths:** Reduces overfitting, handles large datasets, and captures complex interactions.
- **Limitations:** Less interpretable compared to a single decision tree.

4 Implementation

4.1 System Requirements

- **Operating System:** Windows 10
- **Python Version:** Python 3.9
- **IDE/Text Editor:** Google Colab
- **Python Libraries:**
  - **Core Libraries:** Pandas (pd), NumPy (np), Plotly Express (px)
  - **Machine Learning Libraries:** scikit-learn (sklearn)
- **Package Manager:** pip

4.2 Library Module

1. Pandas (pd)

- **Purpose:** Pandas is a widely used data manipulation and analysis library in. Python. It provides data structures and functions to work with structured data, such as tables or spreadsheets.

- **Usage in Code:** In the code, Pandas is used for tasks like reading a CSV file, data exploration (with data.head() and data.info()), and creating dataframes to manage and process data.

2. NumPy (np)

- **Purpose:** NumPy is a fundamental library for numerical operations in Python. It provides support for arrays and mathematical functions.
- **Usage in Code:** NumPy is used for mathematical operations, such as converting degrees to radians and calculating distances using the Haversine formula.

3. Plotly Express (px)

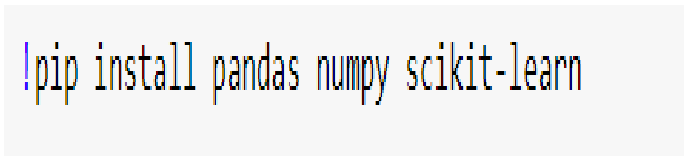
- **Purpose:** Plotly Express is a data visualization library for creating interactive and visually appealing plots and charts.
- **Usage in Code:** Plotly Express generates various plots like scatter and box plots to visualize relationships between variables such as distance, time, age, and ratings.

4. scikit-learn (sklearn)

- **Purpose:** Scikit-learn is a machine learning library that provides a wide range of tools for tasks such as data splitting, model training, model evaluation, and feature selection.
- **Usage in Code:** The code imports train_test_split from scikit-learn to split the dataset into training and testing sets, which is a common step in machine learning model development [15].

4.3 Library Module Setup & Implementation

4.3.1 Library installation

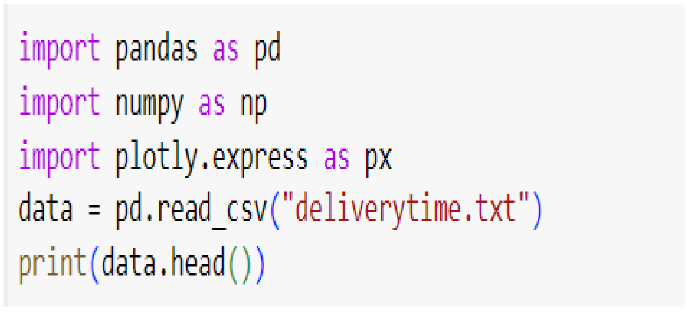


```
!pip install pandas numpy scikit-learn
```

Figure 4.1: Installing Essential Modules

Here, we have installed necessary library for the food delivery time prediction.

4.3.2 Loading dataset and importing libraries

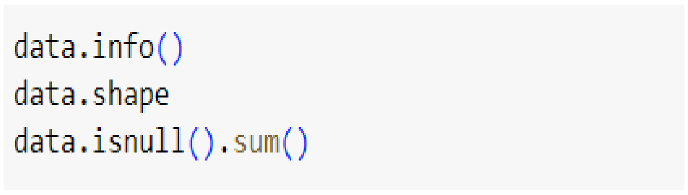


```
import pandas as pd
import numpy as np
import plotly.express as px
data = pd.read_csv("deliverytime.txt")
print(data.head())
```

Figure 4.2: Dataset Viewing

Here, we have imported necessary libraries and viewing the dataset which is named as ”deliverytime.txt”.It then prints the first few rows of this data to give us a quick look at what’s inside the file.

4.3.3 Structure and Quality of data

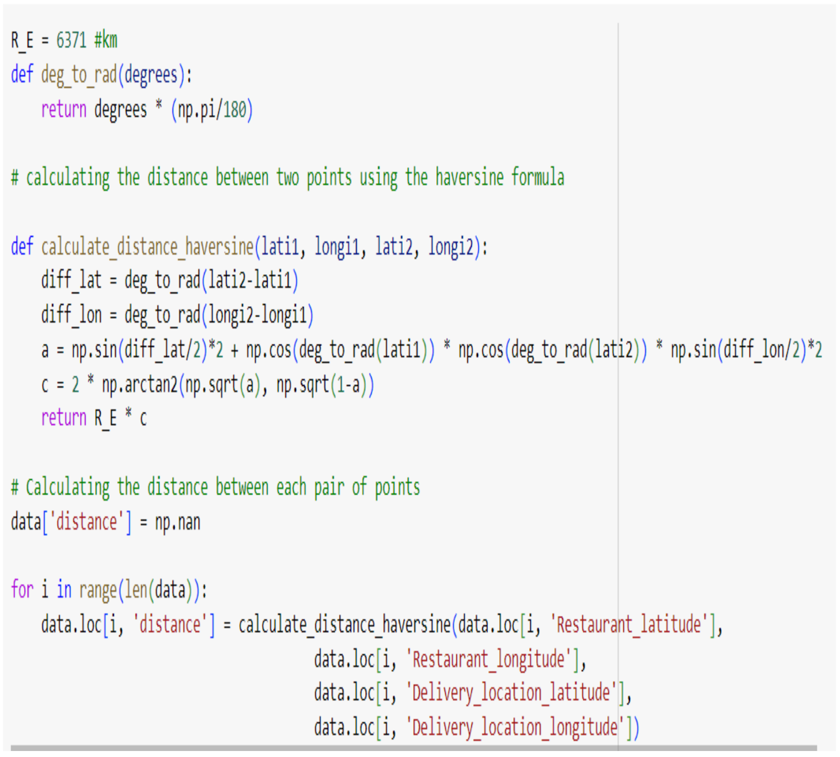


```
data.info()
data.shape
data.isnull().sum()
```

Figure 4.3: Data Schema and Quality Check

”data.info()” shows details about the columns, their data types, and non-null counts. ”data.shape” gives the number of rows and columns in the dataset. ”data.isnull().sum()” checks for any missing values in each column and shows how many are present. This helps in understanding the structure and quality of the data.

4.3.4 Distance Calculation

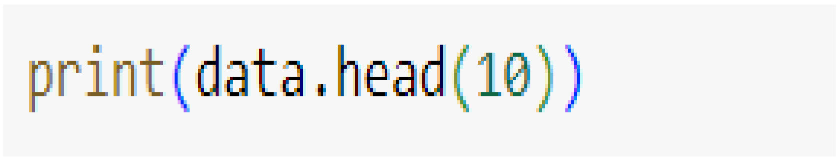


```python
R_E = 6371 #km
def deg_to_rad(degrees):
    return degrees * (np.pi/180)

# calculating the distance between two points using the haversine formula

def calculate_distance_haversine(lati1, longi1, lati2, longi2):
    diff_lat = deg_to_rad(lati2-lati1)
    diff_lon = deg_to_rad(longi2-longi1)
    a = np.sin(diff_lat/2)**2 + np.cos(deg_to_rad(lati1)) * np.cos(deg_to_rad(lati2)) * np.sin(diff_lon/2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    return R_E * c

# Calculating the distance between each pair of points
data['distance'] = np.nan

for i in range(len(data)):
    data.loc[i, 'distance'] = calculate_distance_haversine(data.loc[i, 'Restaurant_latitude'],
                                        data.loc[i, 'Restaurant_longitude'],
                                        data.loc[i, 'Delivery_location_latitude'],
                                        data.loc[i, 'Delivery_location_longitude'])
```

Figure 4.4: Distance Analysis for Locations

This part of the code calculates the distance between two locations using the haversine formula, which accounts for Earth’s curvature. First, it converts degrees to radians for latitude and longitude values. Then, the ”calculate_distance_haversine” function computes the distance between two points based on their latitude and longitude. It uses this function to fill a new column in the ”data” .DataFrame with distances between each restaurant and delivery location. The loop goes through each row of the DataFrame, applies the distance calculation, and stores the result in the ”distance” column.

4.3.5 Entries Print

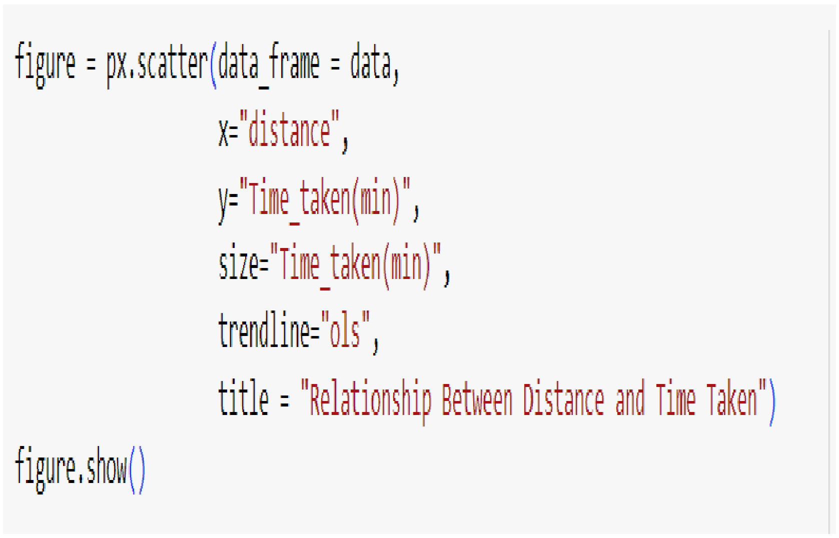


```python
print(data.head(10))
```

Figure 4.5: Entries Print

It prints the first 10 rows of the ‘data‘ DataFrame, giving a quick look at the initial entries to check their content.

4.3.6 Distance vs. Time Taken Analysis

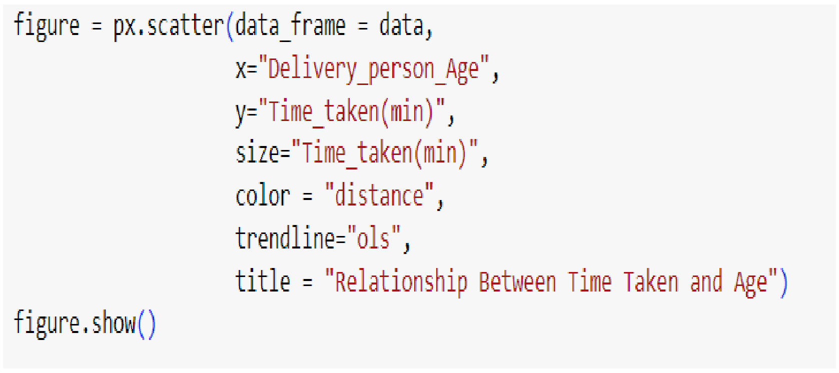


```python
figure = px.scatter(data_frame = data,
                    x="distance",
                    y="Time_taken(min)",
                    size="Time_taken(min)",
                    trendline="ols",
                    title = "Relationship Between Distance and Time Taken")
figure.show()
```

Figure 4.6: Scatter Plot of Distance and Time Taken with Trendline

It creates a scatter plot using Plotly Express to visualize the relationship between distance and time taken. It plots distance on the x-axis and time taken on the y-axis, with the size of each point representing the time taken. The trendline, calculated using Ordinary Least Squares (OLS) regression, helps show the overall trend in the data. The plot is titled ”Relationship Between Distance and Time Taken” and is displayed using ”figure.show()”.

4.3.7 Age vs. Time Taken Analysis

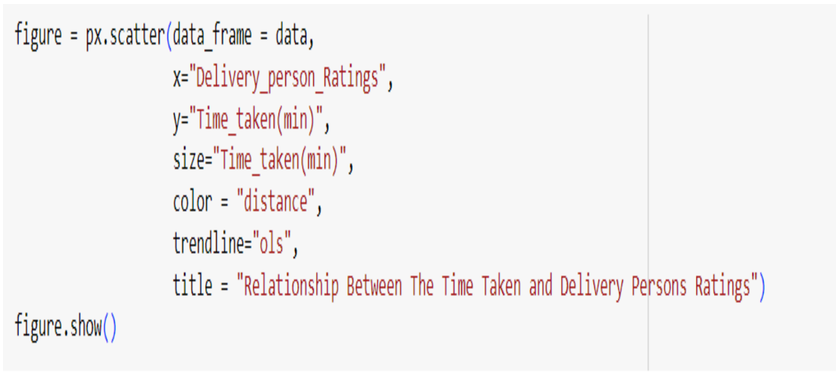


```
figure = px.scatter(data_frame = data,
                    x="Delivery_person_Age",
                    y="Time_taken(min)",
                    size="Time_taken(min)",
                    color = "distance",
                    trendline="ols",
                    title = "Relationship Between Time Taken and Age")
figure.show()
```

Figure 4.7: Scatter Plot of Time Taken and Delivery Person's Age

It creates a scatter plot to explore the relationship between the age of the delivery person and the time taken for delivery. It plots the age on the x-axis and time taken on the y-axis, with the size of each point representing the time taken. The color of each point indicates the distance, and an Ordinary Least Squares (OLS) trendline shows the overall trend. The plot is titled ”Relationship Between Time Taken and Age” and is displayed using ”figure.show()”.

4.3.8 Ratings vs. Time Taken Analysis

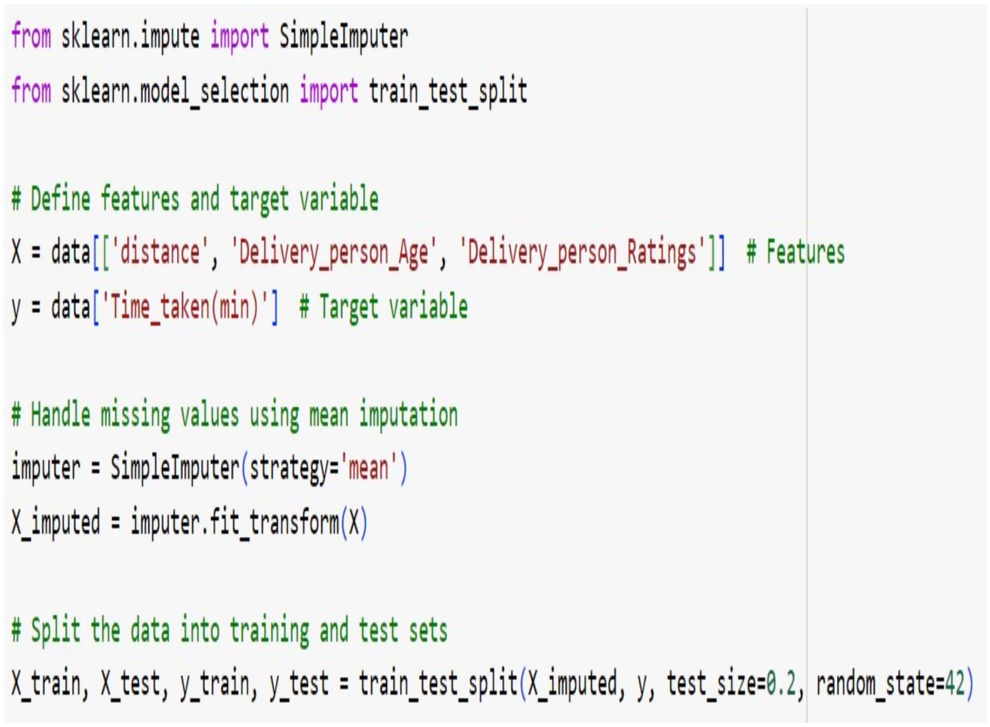


```
figure = px.scatter(data_frame = data,
                    x="Delivery_person_Ratings",
                    y="Time_taken(min)",
                    size="Time_taken(min)",
                    color = "distance",
                    trendline="ols",
                    title = "Relationship Between The Time Taken and Delivery Persons Ratings")
figure.show()
```

Figure 4.8: Scatter Plot of Time Taken and Delivery Ratings

It creates a scatter plot to show how delivery person ratings affect delivery time. It places ratings on the x-axis and time taken on the y-axis, with point sizes representing the time taken and colors showing the distance. The trendline helps to visualize the overall pattern in the data. The plot is titled ”Relationship Between The Time Taken and Delivery Person’s Ratings.”

4.3.9 Data Preparation for Modeling

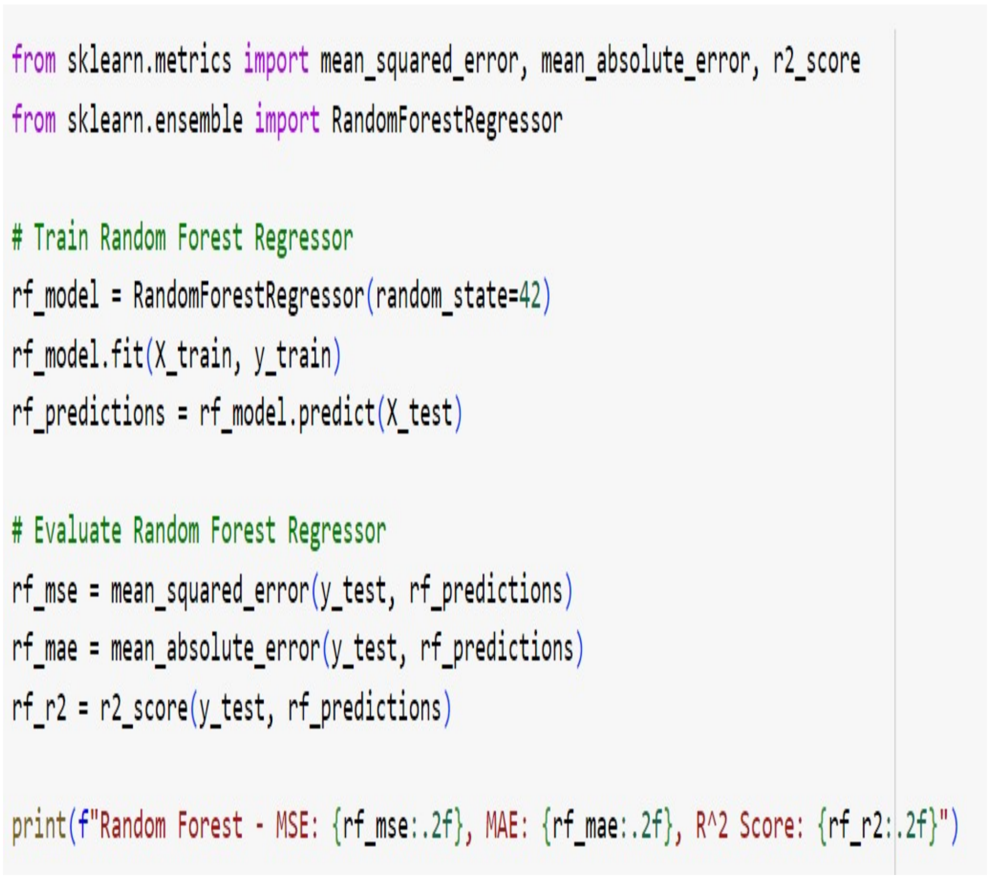


```python
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split

# Define features and target variable
X = data[['distance', 'Delivery_person_Age', 'Delivery_person_Ratings']]  # Features
y = data['Time_taken(min)']  # Target variable

# Handle missing values using mean imputation
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.2, random_state=42)
```

Figure 4.9: Feature and Target Variable Splitting with Missing Values Imputation

This section of the code prepares the data for modeling by first handling any missing values in the features using mean imputation. It then splits the data into training and test sets, with 20% of the data used for testing. The features include distance, delivery person age, and ratings, while the target variable is the time taken.

4.3.10 Evaluating Random Forest Regressor Performance

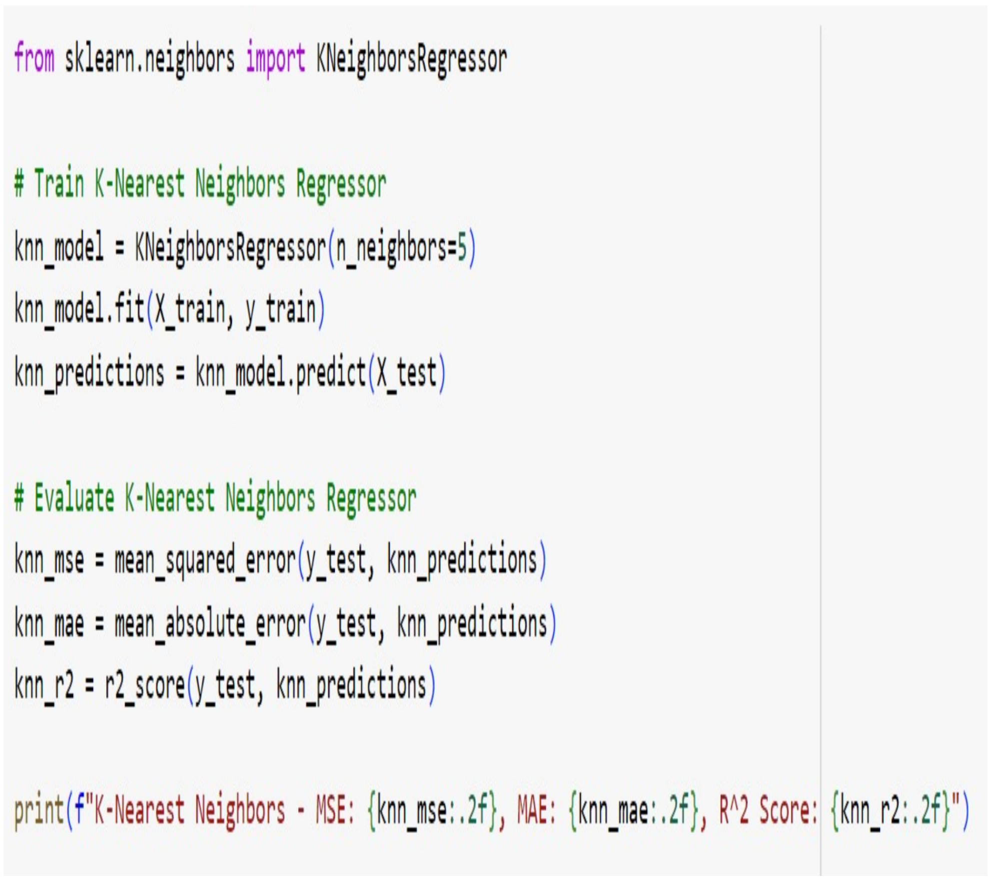


```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor

# Train Random Forest Regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)

# Evaluate Random Forest Regressor
rf_mse = mean_squared_error(y_test, rf_predictions)
rf_mae = mean_absolute_error(y_test, rf_predictions)
rf_r2 = r2_score(y_test, rf_predictions)

print(f"Random Forest - MSE: {rf_mse:.2f}, MAE: {rf_mae:.2f}, R^2 Score: {rf_r2:.2f}")
```

Figure 4.10: Performance Metrics for Random Forest Model

This part of the code trains a Random Forest Regressor model on the training data and makes predictions on the test data. It then evaluates the model's performance using three metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared score (R^2). These metrics help assess how well the model predicts the time taken based on the features.

4.3.11 Evaluating K-Nearest Neighbors Regressor Performance

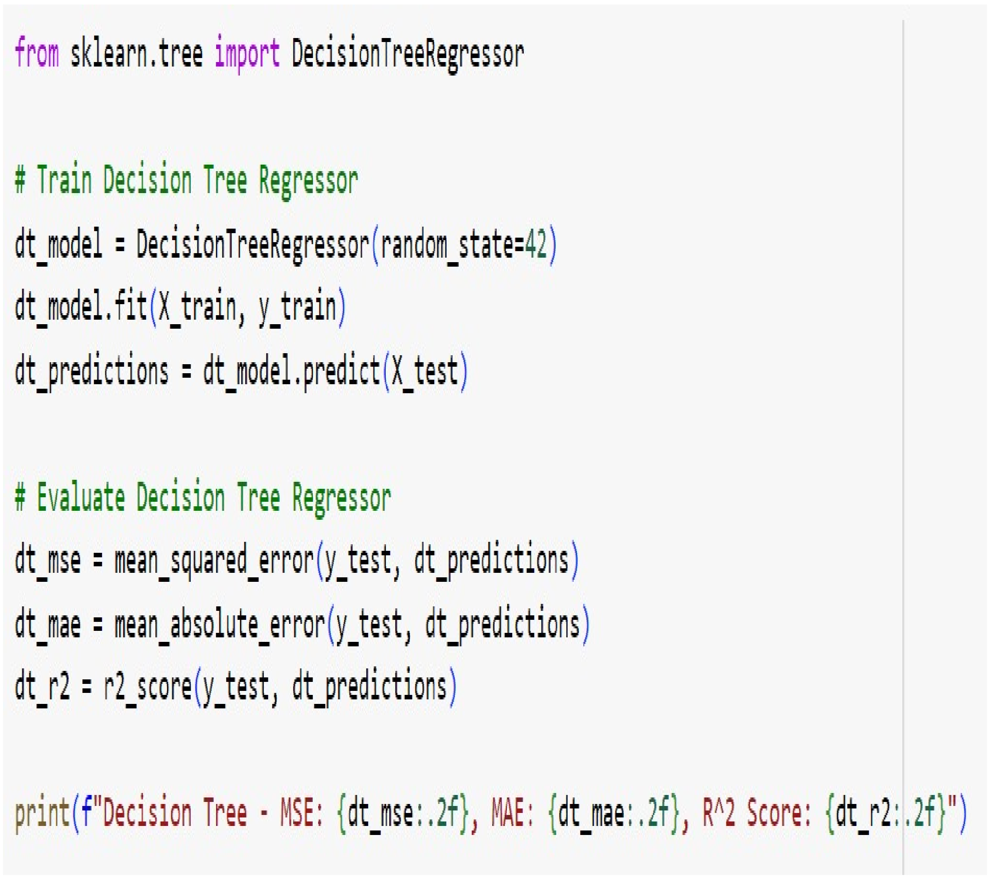


```python
from sklearn.neighbors import KNeighborsRegressor

# Train K-Nearest Neighbors Regressor
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_predictions = knn_model.predict(X_test)

# Evaluate K-Nearest Neighbors Regressor
knn_mse = mean_squared_error(y_test, knn_predictions)
knn_mae = mean_absolute_error(y_test, knn_predictions)
knn_r2 = r2_score(y_test, knn_predictions)

print(f"K-Nearest Neighbors - MSE: {knn_mse:.2f}, MAE: {knn_mae:.2f}, R^2 Score: {knn_r2:.2f}")
```

Figure 4.11: Performance Metrics for K-Nearest Neighbors Model

This section of the code trains a K-Nearest Neighbors (KNN) Regressor model on the training data, using 5 neighbors for predictions. It then evaluates the model's performance on the test data by calculating the Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared score (R^2). These metrics help assess how accurately the KNN model predicts the time taken. The results are printed to show how well the KNN model performs.

4.3.12 Evaluating Decision Tree Regressor Performance

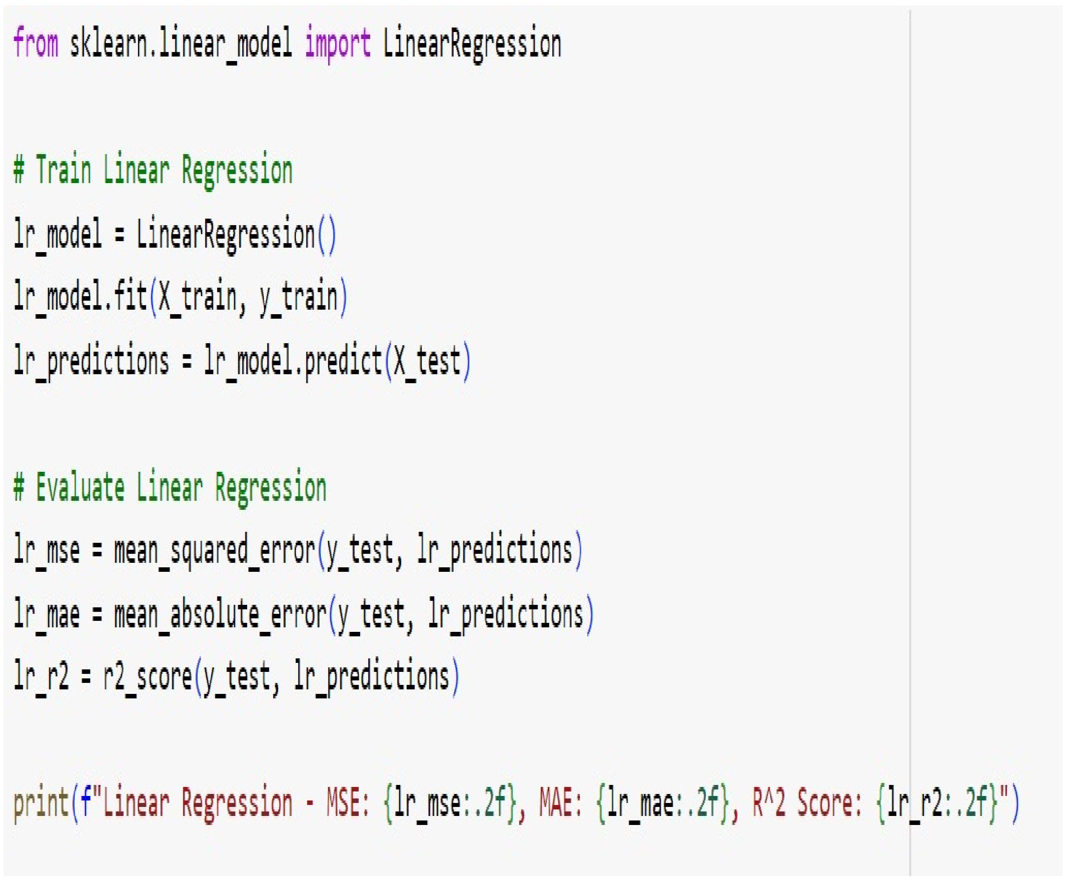


```
from sklearn.tree import DecisionTreeRegressor

# Train Decision Tree Regressor
dt_model = DecisionTreeRegressor(random_state=42)
dt_model.fit(X_train, y_train)
dt_predictions = dt_model.predict(X_test)

# Evaluate Decision Tree Regressor
dt_mse = mean_squared_error(y_test, dt_predictions)
dt_mae = mean_absolute_error(y_test, dt_predictions)
dt_r2 = r2_score(y_test, dt_predictions)

print(f"Decision Tree - MSE: {dt_mse:.2f}, MAE: {dt_mae:.2f}, R^2 Score: {dt_r2:.2f}")
```

Figure 4.12: Performance Metrics for Decision Tree Model

It trains a Decision Tree Regressor model on the training data to predict the time taken for deliveries. It uses the trained model to make predictions on the test data and evaluates its performance by calculating the Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared score (R^2). These metrics measure how well the Decision Tree model predicts the time taken, with lower MSE and MAE indicating better accuracy and a higher R^2 score reflecting a better fit to the data.

4.3.13 Evaluating Linear Regression Model Performance

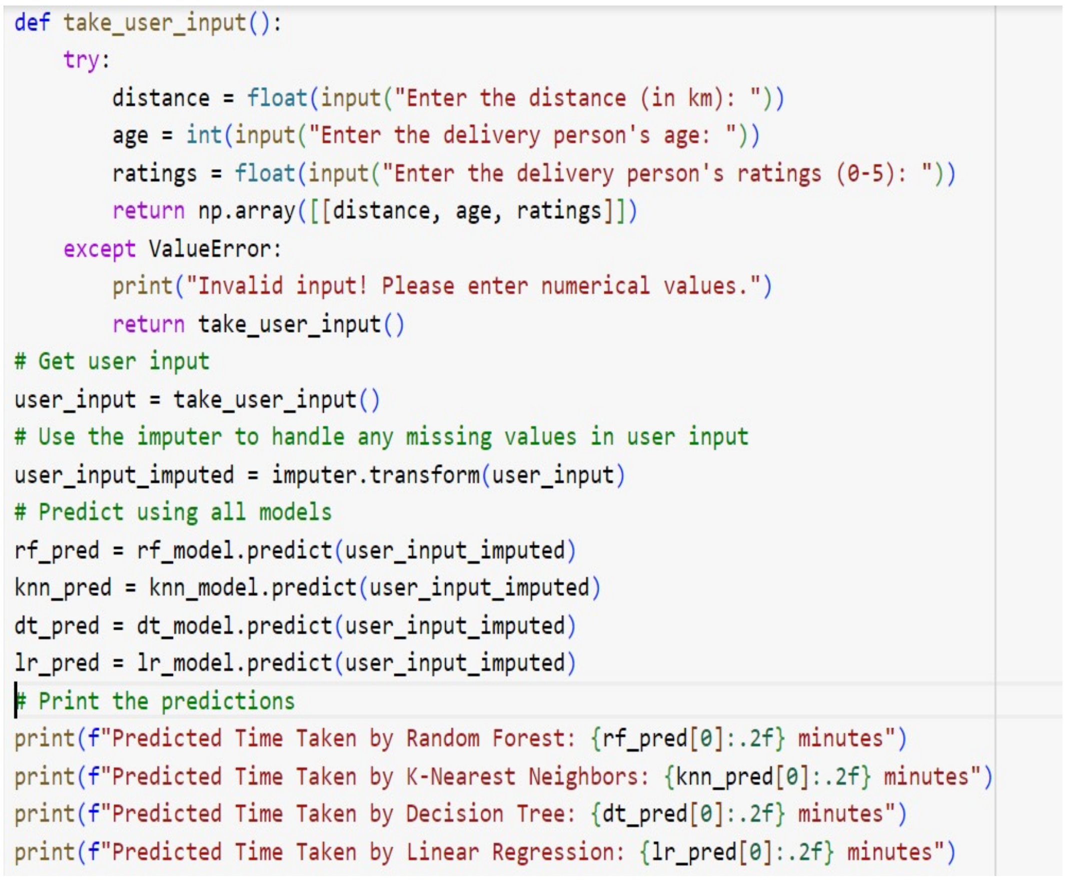


```python
from sklearn.linear_model import LinearRegression

# Train Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)

# Evaluate Linear Regression
lr_mse = mean_squared_error(y_test, lr_predictions)
lr_mae = mean_absolute_error(y_test, lr_predictions)
lr_r2 = r2_score(y_test, lr_predictions)

print(f"Linear Regression - MSE: {lr_mse:.2f}, MAE: {lr_mae:.2f}, R^2 Score: {lr_r2:.2f}")
```

Figure 4.13: Performance Metrics for Linear Regression Model

This part of the code trains a Linear Regression model using the training data to predict the time taken for deliveries. After fitting the model, it makes predictions on the test data. The model's performance is then evaluated using Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared score (R^2). These metrics help assess how accurately the Linear Regression model predicts delivery times, with lower MSE and MAE indicating better accuracy and a higher R^2 score showing a better model fit.

4.3.14 Predicting Delivery Time from User Input

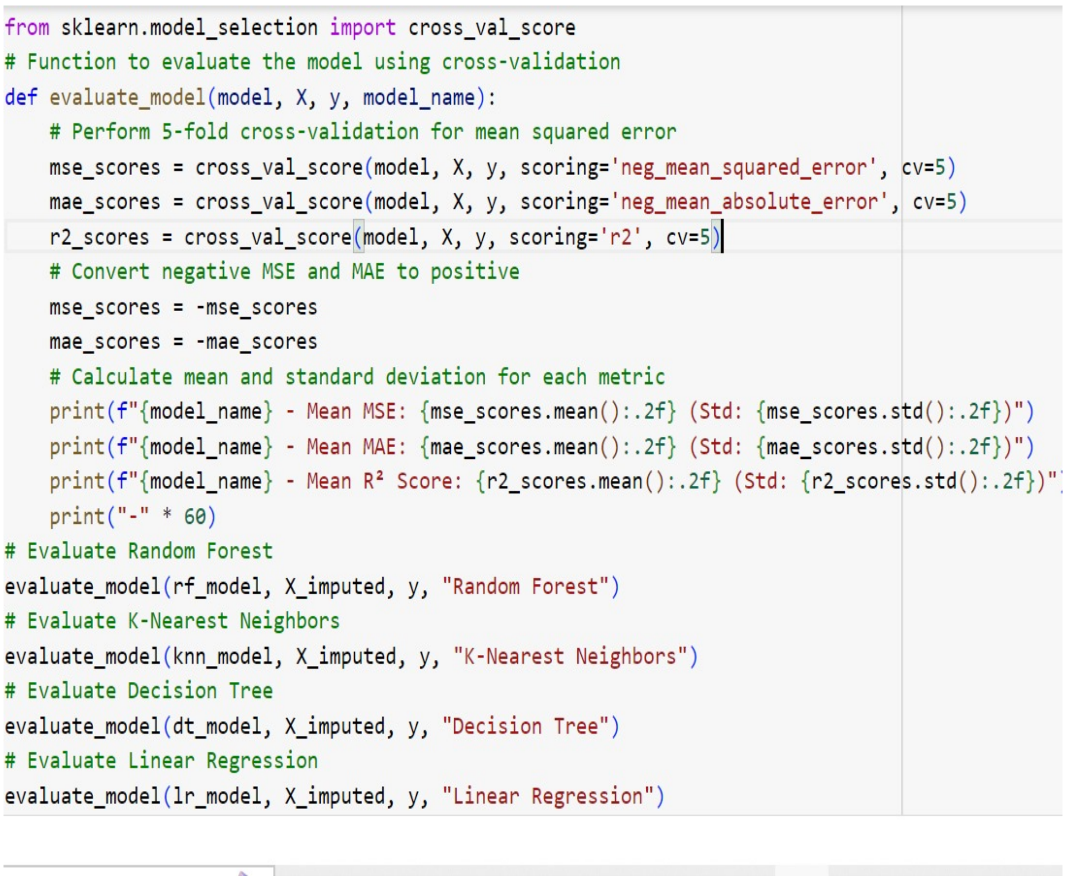


```python
def take_user_input():
    try:
        distance = float(input("Enter the distance (in km): "))
        age = int(input("Enter the delivery person's age: "))
        ratings = float(input("Enter the delivery person's ratings (0-5): "))
        return np.array([[distance, age, ratings]])
    except ValueError:
        print("Invalid input! Please enter numerical values.")
        return take_user_input()
# Get user input
user_input = take_user_input()
# Use the imputer to handle any missing values in user input
user_input_imputed = imputer.transform(user_input)
# Predict using all models
rf_pred = rf_model.predict(user_input_imputed)
knn_pred = knn_model.predict(user_input_imputed)
dt_pred = dt_model.predict(user_input_imputed)
lr_pred = lr_model.predict(user_input_imputed)
# Print the predictions
print(f"Predicted Time Taken by Random Forest: {rf_pred[0]:.2f} minutes")
print(f"Predicted Time Taken by K-Nearest Neighbors: {knn_pred[0]:.2f} minutes")
print(f"Predicted Time Taken by Decision Tree: {dt_pred[0]:.2f} minutes")
print(f"Predicted Time Taken by Linear Regression: {lr_pred[0]:.2f} minutes")
```

Figure 4.14: Delivery Time Predictions from Various Models

It defines a function to collect user inputs for distance, age, and ratings, handling any invalid inputs by prompting the user again. Once valid inputs are received, the data is processed to handle any missing values using an imputer. The function then uses four trained models (Random Forest, K-Nearest Neighbors, Decision Tree, and Linear Regression) to predict the delivery time based on the input. The predictions from each model are displayed to show how long the delivery might take according to different algorithms.

4.3.15 Cross-Validation and Model Performance Evaluation



```
from sklearn.model_selection import cross_val_score
# Function to evaluate the model using cross-validation
def evaluate_model(model, X, y, model_name):
    # Perform 5-fold cross-validation for mean squared error
    mse_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
    mae_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error', cv=5)
    r2_scores = cross_val_score(model, X, y, scoring='r2', cv=5)
    # Convert negative MSE and MAE to positive
    mse_scores = -mse_scores
    mae_scores = -mae_scores
    # Calculate mean and standard deviation for each metric
    print(f"{model_name} - Mean MSE: {mse_scores.mean():.2f} (Std: {mse_scores.std():.2f})")
    print(f"{model_name} - Mean MAE: {mae_scores.mean():.2f} (Std: {mae_scores.std():.2f})")
    print(f"{model_name} - Mean R² Score: {r2_scores.mean():.2f} (Std: {r2_scores.std():.2f})"
    print("-" * 60)
# Evaluate Random Forest
evaluate_model(rf_model, X_imputed, y, "Random Forest")
# Evaluate K-Nearest Neighbors
evaluate_model(knn_model, X_imputed, y, "K-Nearest Neighbors")
# Evaluate Decision Tree
evaluate_model(dt_model, X_imputed, y, "Decision Tree")
# Evaluate Linear Regression
evaluate_model(lr_model, X_imputed, y, "Linear Regression")
```

Figure 4.15: Model Performance Metrics from Cross-Validation

This part of the code performs cross-validation to assess the performance of different models by splitting the data into 5 parts and evaluating each model's accuracy. It calculates three metrics: Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared score (R^2), converting negative values to positive for MSE and MAE. The function evaluate_model computes the average and standard deviation of these metrics for each model, providing a comprehensive view of their performance. This process is applied to the Linear Regression, Random Forest, K-Nearest Neighbors, and Decision Tree models to compare their accuracy.

4.4 Graphical Representation

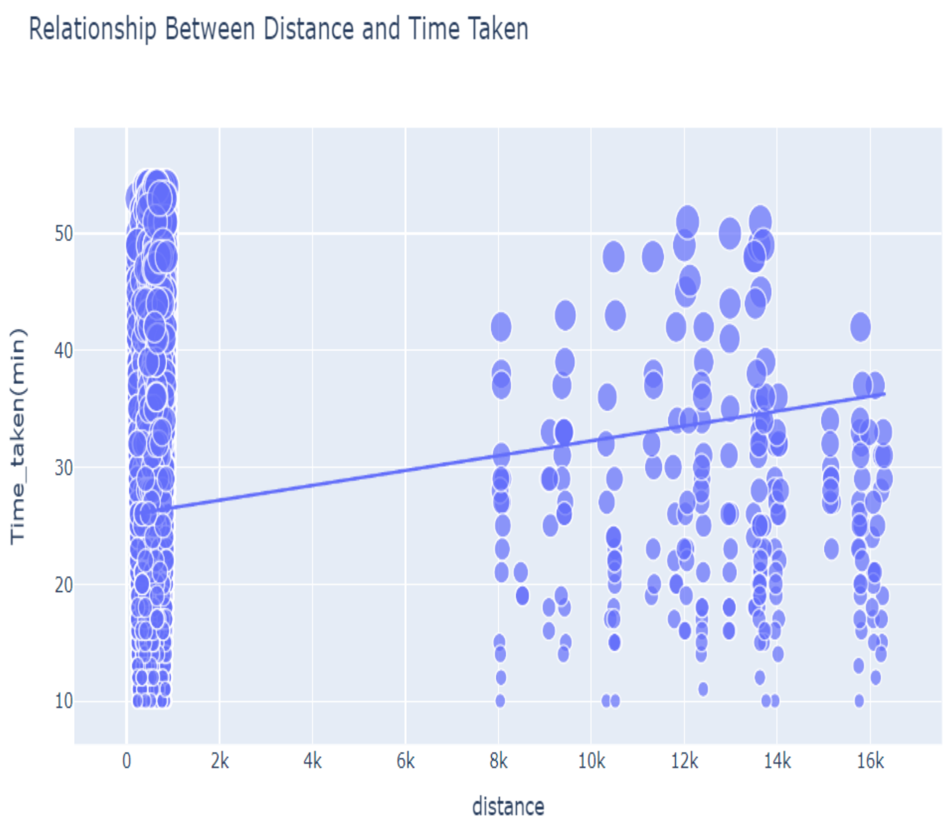


Figure 4.16: Relationship between distance and time taken

The scatter plot shows the relationship between distance and time taken. As the distance increases, the time taken also tends to increase, which is indicated by the upward sloping trend of the blue line.

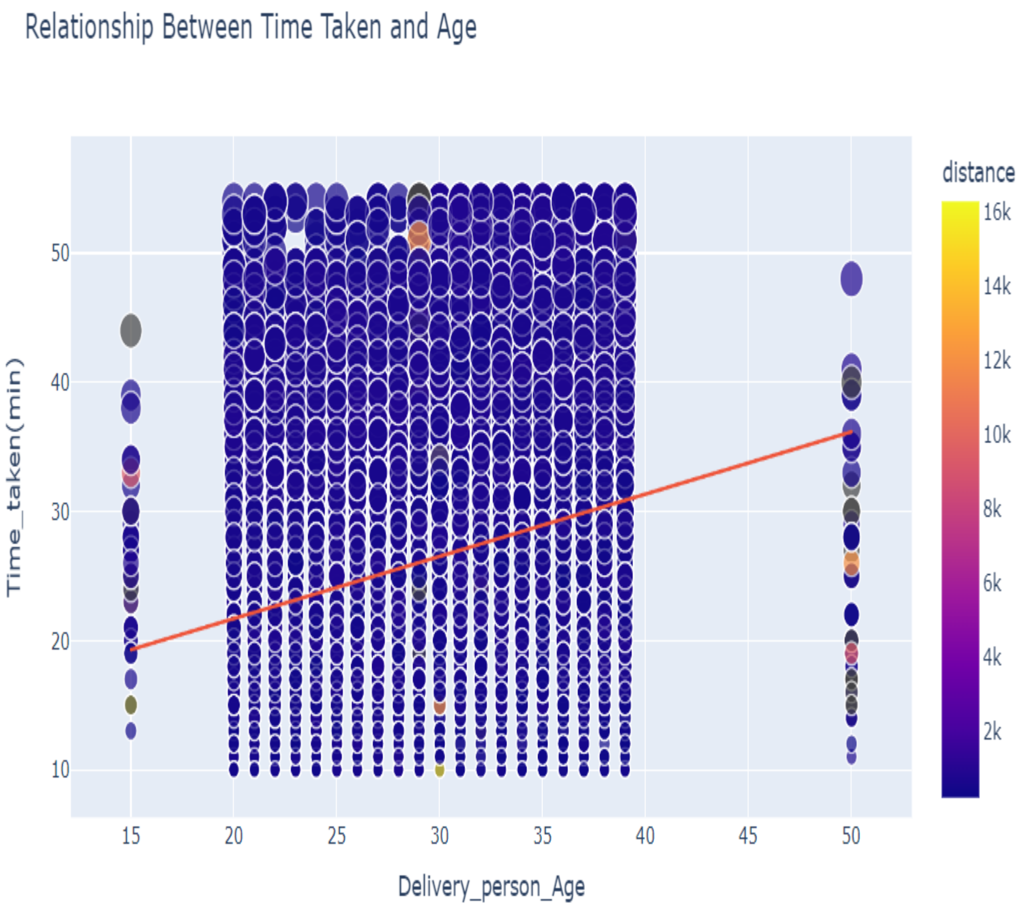


Figure 4.17: Relationship between the age of delivery persons and the time taken

The scatter plot illustrates the relationship between the age of delivery persons and the time taken for deliveries. The red line, known as the regression line, shows a slight upward trend, suggesting that as the age of delivery persons increases, the time taken for deliveries also tends to increase.

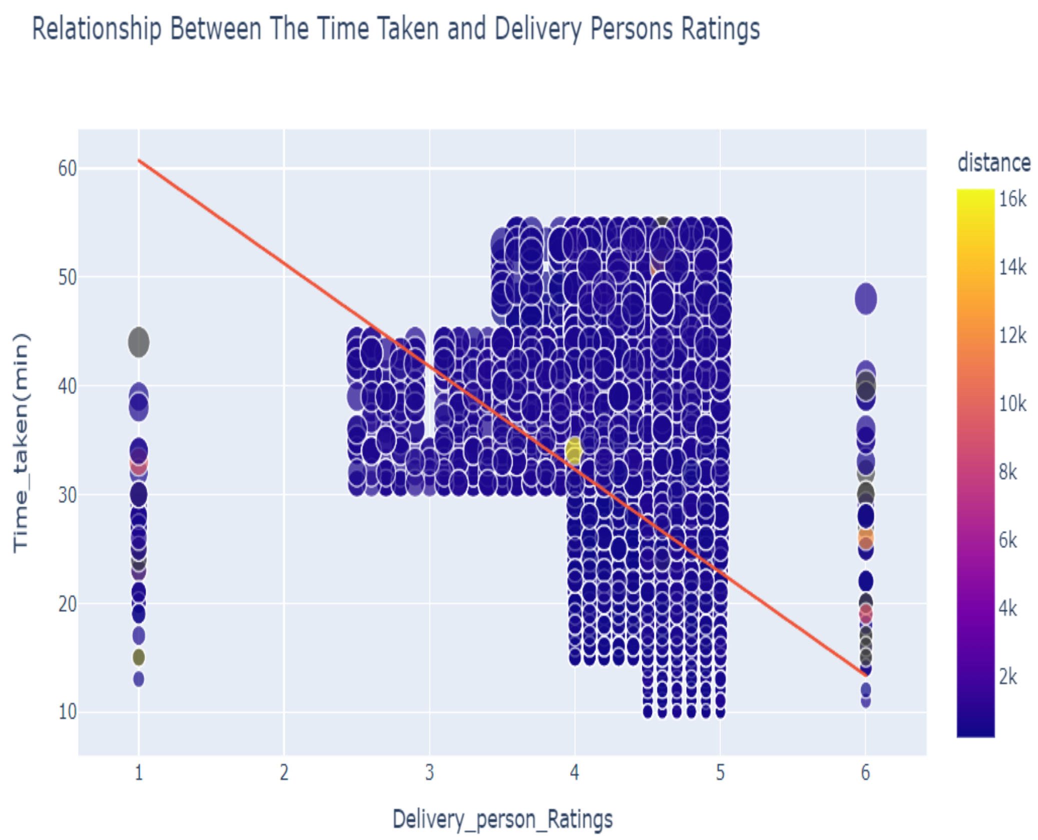


Figure 4.18: Relationship between delivery person ratings and the time taken

The scatter plot examines the relationship between delivery person ratings and the time taken for deliveries. The downward sloping red line indicates a negative correlation,as delivery person ratings increase, the time taken for deliveries tends to decrease. This implies that higher-rated delivery persons are generally more efficient in completing deliveries.

5 Result Analysis

5.1 Result Accuracy

The performance of each model was evaluated using metrics such as Mean Squared Error (MSE), R-squared and Mean Absolute Value(MAE) on a validation dataset. The following table summarizes the accuracy of each model:

Model	**Mean Squared Error (MSE)**	**R-squared**	**Mean Absolute Error (MAE)**
Random Forest	66.55	0.24	6.33
Linear Regression	71.77	0.18	6.63
Decision Tree	105.58	-0.20	7.92
K-Nearest Neighbors (KNN)	72.78	0.17	6.69

Table 1: Model Performance Metrics

Here, we have assumed random state= 42 and n neighbour k=5. From the table, it is evident that the Random Forest model achieved the lowest Mean Squared Error and the highest R-squared value. Also Random Forest has the lowest Mean Absolute Value, indicating better predictive performance compared to the other models.

5.2 Evaluation

The dataset [16] used in this project included features such as the distance between the restaurant and delivery location, the age of the delivery personnel, and their ratings. The data was preprocessed to handle missing values, normalize numerical features, and encode categorical variables.

The models were trained on this preprocessed dataset, and their performance was evaluated on a separate validation set. Cross-validation was also performed to ensure the robustness and generalizability of the models.

In conclusion, the Random Forest model outperformed the other models in terms of accuracy and robustness, making it the best choice for predicting food delivery times in this project. This result highlights the importance of using ensemble methods to capture complex relationships and improve predictive performance in machine learning tasks.

5.3 Key Metrics for Predicting Delivery Time Accuracy

- **Mean MSE:** Lower is better. Indicates the average squared difference between predicted and actual values.
- **Mean MAE:** Lower is better. Indicates the average absolute difference between predicted and actual values.
- **Mean R^2 Score:** Closer to 1 is better. Indicates how well the model fits the data.

The model with the lowest MSE and MAE and the highest R^2 Score will likely be the most appropriate model for predicting delivery time.

5.4 Graphical Representation of Model Accuracy

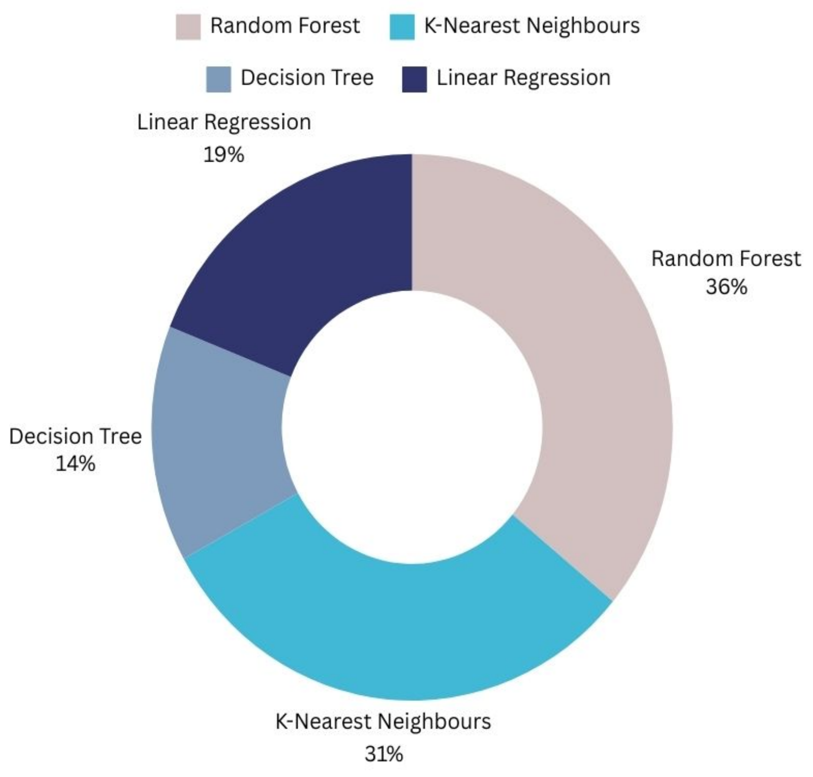


Figure 5.1: Pie Chart of model accuracy

Among the models, Random Forest performs the best, with the lowest Mean Squared Error (MSE) of 66.55, the lowest Mean Absolute Error (MAE) of 6.33, and the highest R^2 score of 0.24, giving it an accuracy of 36%. This indicates that it makes the fewest large prediction errors and explains the most variability in the data. Linear Regression (accuracy: 19%) and K-Nearest Neighbors (KNN) (accuracy: 31%) perform similarly but less accurately, with higher MSE and MAE values and lower R^2 scores, suggesting they are less effective in predicting delivery time. Decision Tree, with an accuracy of 14%, performs the worst, having the highest MSE (105.58), highest MAE (7.92), and a negative R^2 score (-0.20), indicating it fails to reflect the relationship between the input data and delivery time, making it an unreliable model.

6 Future Work

6.1 Lackings of the Project

1. **Limited Feature Set:**
   - **Current Limitation:** The project uses a limited number of features (distance, delivery person's age, and ratings) for predicting delivery time. Other potentially influential factors such as traffic conditions, weather, and time of day are not considered.
   - **Future Work:** Incorporate additional features like real-time traffic data, weather conditions, order size, and delivery time windows to improve the accuracy and reliability of the predictions.
2. **Static Model Training:**
   - **Current Limitation:** The models are trained on a static dataset and are not updated with new data.
   - **Future Work:** Implement continuous learning mechanisms where the models are periodically retrained with new data to adapt to changing conditions and improve performance over time.
3. **Scalability Issues:**

- **Current Limitation:** Models like KNN and Decision Tree might face scalability issues with very large datasets, leading to increased computation time and memory usage.

- **Future Work:** Explore more scalable algorithms and techniques such as distributed computing, and algorithm optimization to handle larger datasets efficiently.

4. **Geographical Constraints:**

- **Current Limitation:** The project assumes a relatively flat geographical area, potentially overlooking the impact of varied terrains on delivery times.

- **Future Work:** Include geographical information and terrain analysis to account for elevation changes and other geographical factors affecting delivery times.

5. **Evaluation on Limited Data:**

- **Current Limitation:** The models are evaluated on a relatively small dataset, which might not fully represent the diverse scenarios encountered in real-world food delivery.

- **Future Work:** Expand the dataset to include more diverse and comprehensive data from different regions and time periods to better generalize the model's predictions.

6. **Absence of Real-time Prediction Capability:**

- **Current Limitation:** The project does not include real-time prediction capabilities, which are crucial for practical applications.

- **Future Work:** Develop real-time prediction frameworks and integrate them with the delivery system to provide instantaneous delivery time estimates.

7. **Lack of User Feedback Integration:**

- **Current Limitation:** The models do not incorporate user feedback or satisfaction levels, which could provide valuable insights into improving delivery efficiency.

- **Future Work:** Implement mechanisms to collect and integrate user feedback into the model to continuously enhance the service quality based on customer experiences.

6.2 Future Work

1. **Integration of Real-time Data:**
   - Develop a framework to integrate real-time traffic and weather data into the predictive model. This would involve setting up APIs to fetch current conditions and dynamically updating predictions based on this real-time data.
2. **Advanced Machine Learning Techniques:**
   - Experiment with advanced machine learning and deep learning techniques such as Gradient Boosting, XGBoost, and Neural Networks to potentially improve prediction accuracy and handle complex relationships within the data.
3. **Hybrid Models:**
   - Explore the development of hybrid models that combine the strengths of multiple algorithms. For example, using a Random Forest to handle feature importance and a Neural Network to capture complex patterns.
4. **User Personalization:**
   - Implement personalized models that account for individual delivery personnel's historical performance and route preferences, providing more tailored and accurate delivery time predictions.
5. **Feedback Loop System:**
   - Create a feedback loop where delivery time predictions are continuously updated based on actual delivery times. This system would allow the model to learn from its past predictions and improve over time.

6. Expansion to Other Delivery Services:

- Extend the predictive model to other delivery services, such as groceries, packages, or medicine, to validate its applicability and effectiveness across different types of deliveries.

7. Visualization and Reporting Tools:

- Develop comprehensive visualization and reporting tools to provide insights into delivery performance, identify bottlenecks, and enable data-driven decision-making for operational improvements.

By addressing these limitations and pursuing these future work directions, the project can be significantly enhanced, leading to more accurate and reliable food delivery time predictions and ultimately improving customer satisfaction and operational efficiency.

7 Conclusion

In this project, we built a model to predict food delivery times using different machine learning methods like K-Nearest Neighbors (KNN), Linear Regression, Decision Tree, and Random Forest. The aim was to make delivery time predictions more accurate and reliable, which could help improve the efficiency of operations and customer satisfaction in the food delivery business.

Out of all the models, the Random Forest algorithm performed the best. It had the lowest Mean Squared Error and the highest R-squared value, meaning it was more accurate and reliable in its predictions. The project also showed how important it is to carefully choose and prepare the right data features, such as distance, the age of the delivery person, and their ratings. However, there were some challenges. We only used a limited set of features, trained the model once without updates, and found that the model was sensitive to outliers and had issues scaling up. These challenges suggest areas for future work, like adding real-time traffic and weather data, trying out more advanced models, and creating personalized delivery time predictions.

In summary, this project showed that machine learning can effectively predict food delivery times. The knowledge and framework developed here lay a solid foundation for further improvements, with the goal of building a strong, real-time prediction system that could greatly benefit the food delivery industry.

Bibliography

[1] C. Hirschberg, A. Rajko, T. Schumacher, and M. Wrulich, “The changing market for food delivery,” 2016.

[2] R. Luhtala *et al.*, “Quantifying the effects of external predictors on food delivery platform order volumes,” M.S. thesis, 2023.

[3] L. C. Coelho, J.-P. Gagliardi, J. Renaud, and A. Ruiz, “Solving the vehicle routing problem with lunch break arising in the furniture delivery industry,” *Journal of the Operational Research Society*, vol. 67, pp. 743–751, 2016.

[4] G. Laporte, “The traveling salesman problem: An overview of exact and approximate algorithms,” *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[5] B.-H. Ahn and J.-Y. Shin, “Vehicle-routeing with time windows and time-varying congestion,” *Journal of the Operational Research Society*, vol. 42, no. 5, pp. 393–400, 1991.

[6] G. J. Beaujon and M. A. Turnquist, “A model for fleet sizing and vehicle allocation,” *Transportation Science*, vol. 25, no. 1, pp. 19–45, 1991.

[7] D. O. Casco, B. L. Golden, E. A. Wasil, *et al.*, “Vehicle routing with backhauls: Models, algorithms and case studies,” *Vehicle routing: methods and studies*, no. 16, pp. 127–147, 1988.

[8] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis, “Time constrained routing and scheduling,” *Handbooks in operations research and management science*, vol. 8, pp. 35–139, 1995.

[9] M. Vössing, “Towards managing complexity and uncertainty in field service technician planning,” in *2017 IEEE 19th Conference on Business Informatics (CBI)*, IEEE, vol. 1, 2017, pp. 312–319.

[10] K. Sörensen, M. Sevaux, and P. Schittekat, ““multiple neighbourhood” search in commercial vrp packages: Evolving towards self-adaptive methods,” *Adaptive and multilevel metaheuristics*, pp. 239–253, 2008.

[11] H. Şahin and D. İçen, “Application of random forest algorithm for the prediction of online food delivery service delay,” *Turkish Journal of Forecasting*, vol. 5, no. 1, pp. 1–11, 2021.

[12] C. Gao, F. Zhang, G. Wu, *et al.*, “A deep learning method for route and time prediction in food delivery service,” in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2879–2889.

[13] E. Yalçinkaya and O. A. Hızıroğlu, “A comparative analysis of machine learning models for time prediction in food delivery operations,” *Artificial Intelligence Theory and Applications*, vol. 4, no. 1, pp. 43–56, 2024.

[14] M. Kumar, “Prediction of kitchen preparation time in food delivery,” Ph.D. dissertation, INDIAN INSTITUTE OF TECHNOLOGY DELHI, 2023.

[15] A. Crivellari, E. Beinat, S. Caetano, A. Seydoux, and T. Cardoso, “Multi-target cnn-lstm regressor for predicting urban distribution of short-term food delivery demand,” *Journal of Business Research*, vol. 144, pp. 844–853, 2022.

[16] A. Kharwal, *Food delivery time prediction: Case study*, en-US, Jan. 2023. [Online]. Available: `https://statso.io/food-delivery-time-prediction-case-study/`.

Contribution Matrix

	Showrin (2199)	Ohee (2204)
Introduction, Purpose, Background		■
Motivation,Optimal Performance,Objectives	■	
Related Works	■	■
Methodology	■	■
Implementation	■	■
Result Accuracy	■	■
Lacking of the Project		■
Future Work	■	
Conclusion	■	■