

Machine Learning Project Swiggy Delivery Time Prediction CSM355

By

K Jaya Sravani Reg No: 12212274  
Roll No: 29



School of Computer Science and Engineering Lovely Professional University

Phagwara, Punjab (India)

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| S. No. | Section Title | Page No. |
| 1 | Problem Title and Problem Statement | 3 - 6 |
| 1.1 | Problem Statement |  |
| 1.2 | Business Use Case |  |
| 1.3 | Machine Learning  Enhancing Delivery Efficiency |  |
| 1.4 | Impact on Business, Riders, and Restaurants |  |
| 2 | Objective | 7 |
| 3 | Scope | 8 - 9 |
| 4 | Literature Review | 10 - 11 |
| 5 | Dataset Description and Code Snippets | 12 - 22 |
| 5.1 | Data Source |  |
| 5.2 | Data Characteristics |  |
| 5.3 | Data Preprocessing |  |
| 5.4 | Feature Selection and Engineering |  |
| 6 | Model Building | 23 - 24 |
| 7 | Model Evaluation | 25 - 26 |
| 8 | Result and Analysis | 27 - 28 |
| 9 | Conclusion and Future Work | 29 |
| 10 | References | 30 |

# Problem Title and Problem Statement

### Project Title: Swiggy Delivery Time Prediction using Machine Learning

* 1. Problem Statement:

The goal of this project is to predict the delivery time for food orders based on factors like rider details, vehicle type, weather, traffic, and locations of the restaurant and delivery destination. Accurate predictions will help optimize delivery operations, improve customer satisfaction, and enhance resource allocation for efficient service.

### Business Use Case:

In the competitive food delivery market, on-time delivery is critical for customer satisfaction, retention, and operational efficiency. A company wants to optimize delivery time predictions to improve customer experience by providing accurate estimated delivery times (ETAs) and to manage resources effectively. Accurate predictions of delivery time can also allow the business to:

* + 1. **Improve Delivery Efficiency:** Identifying factors that slow down deliveries enables better resource allocation, such as more reliable scheduling for delivery personnel.
    2. **Enhance Customer Satisfaction:** Reliable delivery ETAs can improve the customer experience by reducing wait-time uncertainty.
    3. **Optimize Operational Costs:** If the model can predict scenarios with higher delays, additional resources (like more drivers or prioritizing specific orders) can be allocated.

### Machine Learning in Enhancing Delivery Efficiency

#### Enhanced Customer Experience

* + **Customer Satisfaction and Retention:** Delivering on promised ETAs is critical to customer satisfaction. When customers know exactly when their order will arrive, they can plan accordingly, reducing their anxiety over wait times. Reliable ETAs mean customers aren’t left guessing or feeling frustrated, which increases trust and brand loyalty. A happy customer is more likely to become a repeat customer, directly impacting revenue growth through increased retention.
  + **Improved Transparency:** Clear, accurate ETAs improve transparency, a factor increasingly valued by customers. If a delay is predicted due to conditions like traffic or weather, proactive updates to ETAs reassure customers that they are informed. This builds trust and creates a positive brand image, especially important in a crowded food delivery market.
  + **Reduced Customer Support Load:** Accurate predictions minimize delays and the subsequent

customer service calls and complaints that typically follow. Fewer service interactions reduce operational costs associated with customer support, allowing the business to reinvest these resources into growth initiatives or new customer service features, like a rewards program or app improvements.

#### Operational Efficiency for Delivery Management

* + **Better Resource Utilization:** With predictive insights on delivery times, the dispatch team can plan routes and schedules more effectively, avoiding high-traffic times or preparing for weather disruptions. For instance, if peak traffic density is predicted, drivers could be assigned specific zones, ensuring they can make more deliveries in a shorter timeframe. This reduces idle times, maximizes the time each driver spends on productive tasks, and makes the fleet more efficient overall.
  + **Dynamic Allocation:** When anticipated delays are flagged in advance, dispatch teams can make real-time decisions, such as reassigning drivers, adjusting routes, or rescheduling specific deliveries to ensure on-time arrivals for high-priority orders. This adaptability in resource management ensures smoother operations and helps the business consistently meet or exceed delivery expectations, even during busy or challenging periods.
  + **Operational Scalability:** As the business grows, knowing the expected delivery times allows for planning more delivery zones or micro-fulfillment centers in strategic locations. This can help scale operations smoothly, ensuring the business meets increased demand without overburdening resources.

#### Financial Optimization

* + **Cost Control through Efficiency:** Accurate delivery time predictions enable better route planning, which reduces fuel costs, overtime pay, and wear on vehicles. This kind of operational efficiency keeps delivery costs manageable and reduces the cost-per-delivery, which has a direct impact on profitability.
  + **Minimized Compensation Costs:** Predicting delays allows the business to manage customer expectations in advance, avoiding the need for compensations (e.g., refunds, discounts on next orders) due to late deliveries. Lowering such costs contributes to higher profit margins.
  + **Increased Customer Lifetime Value (CLV):** A happy customer who receives reliable ETAs is more likely to order again and again. Over time, this leads to increased customer lifetime value, as each repeat order adds to revenue without the acquisition cost associated with new customers.

#### Strategic Decision-Making

* + **Peak and Off-Peak Planning:** The model can identify patterns related to demand peaks, like specific times, holidays, or weather conditions that impact delivery time. This information allows the business to prepare, such as increasing driver availability during expected peaks and adjusting promotions or delivery charges in response to anticipated conditions. Properly managing high- demand periods ensures smooth service and maintains customer satisfaction during peak times.
  + **Operational Flexibility:** When data highlights challenging delivery periods, such as during festivals or high-traffic hours, businesses can implement strategies like surge pricing or limit orders to high- capacity areas only. This flexibility helps avoid bottlenecks, ensuring that resources are stretched only to manageable levels.
  + **Data-Driven Expansion Plans:** Insights from delivery time predictions guide strategic planning for geographic expansion or new fulfillment centers. If certain locations consistently show high delivery times, it might signal a need for a local hub to streamline deliveries, improving service without straining existing resources.
  1. Impact on Business, Riders, and Restaurants
     1. Increase customer satisfaction as customers can plan the ETA of their orders.
     2. Increases customer trust in the company. Clear, accurate ETAs improve transparency, a factor increasingly valued by customers. If a delay is predicted due to conditions like traffic or weather, proactive updates to ETAs reassure customers that they are informed.
     3. Accurate time predictions reduce the chances of cancelled orders. Increased transparency can help in lowering customer service calls, easing up the traffic of complaints related to time.
     4. The dispatch team for riders can plan routes and manpower accordingly to serve customers on time.
     5. They can focus on hotspots in the city which have increased orders at certain times of day, month, or year.
     6. Can help the company implement surge pricing in extreme weather or congestion events.

#### Riders

1. Riders can plan pickups and deliveries accordingly.
2. They have a foresight of the time taken for delivery, so they can manage multiple orders along the same route.
3. Can help in route planning in case of traffic congestion.
4. Can do faster deliveries and limit wait times to increase the number of deliveries per day, which

increases their earning potential.

1. Drivers do not have to rush or do risky driving during high rush hours as their delivery times are in synchrony with the on-ground situation, giving them peace of mind and reducing the chances of unnecessary cancellations. It also does not impact their ratings.
2. Can opt for other providers when demand is less to increase their earnings.
3. Can tackle multiple deliveries.

#### Restaurants

1. They can prioritize their orders if delivery times are available.
2. They can manage staff to balance out between in-house orders vs home deliveries.
3. They can scale up staff and resources during events of increased demand.
4. The company can also leverage discounts and coupons to increase demand during off-peak hours, resulting in continuous revenue generation.

# Objective

The primary objective of this project is to **develop a machine learning-based model that accurately predicts the delivery time of food orders** placed through Swiggy, using historical and real-time operational data. The model is designed to enhance the accuracy of Estimated Time of Arrival (ETA), thereby improving customer satisfaction, optimizing delivery logistics, and reducing operational inefficiencies.

#### Purpose of the Project

* To build a **predictive model** that estimates how long it will take for an order to be delivered, based on several influencing factors.
* To understand the **relationship between delivery time and variables** such as delivery personnel details (age, ratings), weather conditions, road traffic, order timing, and geolocation data.
* To support **data-driven decision making** for dispatch teams by identifying bottlenecks and causes of delay in the delivery pipeline.
* To enhance **transparency** and **reliability** for customers by providing more accurate and consistent ETAs.

#### Specific Problems the Project Aims to Solve

1. **Inaccuracy in Current Delivery Time Estimates:**

Traditional methods often provide static or average delivery times that fail to reflect real-time factors such as traffic, weather, or delivery person availability. This project aims to resolve that by dynamically predicting delivery time using historical patterns.

#### Operational Inefficiency:

Without reliable delivery time forecasts, it becomes challenging to allocate drivers efficiently. This model aims to enable **dynamic scheduling and route optimization** to improve driver utilization and reduce idle time.

#### Customer Dissatisfaction Due to Delays:

Late deliveries can result in negative feedback, reduced customer loyalty, and even order cancellations. The objective is to **minimize late deliveries** through accurate prediction and proactive customer updates.

#### Scalability of Delivery Operations:

As the business expands, manual estimation becomes infeasible. The machine learning model provides a **scalable solution** that can adapt to larger datasets and growing delivery volumes.

1. **Business Losses from Order Cancellations and Compensation:** Predictive accuracy helps reduce delivery failures and the need for refunds or discounts, thereby improving **profit margins and resource planning**.

# Scope

The scope of this project is focused on developing a machine learning model to predict the delivery time for food orders placed via Swiggy, using a dataset comprising historical delivery information. The model aims to analyze and identify key factors influencing delivery time, such as delivery personnel attributes, environmental conditions, traffic patterns, and geographical distances.

#### Inclusions (What the Project Covers):

* **Dataset Analysis:**

The project includes exploratory data analysis and preprocessing of a real-world delivery dataset, including cleaning, transformation, and feature engineering.

#### Predictive Modeling:

The model is developed using supervised machine learning techniques for regression tasks, where the target variable is Time\_taken(min).

#### Feature Selection & Engineering:

Feature relevance is determined using statistical techniques and domain knowledge. Additional derived features such as delivery distance and time bands are engineered to enhance model performance.

#### Algorithm Implementation:

Multiple regression algorithms such as Linear Regression, Random Forest Regressor, and XGBoost are explored to identify the most accurate model.

#### Model Evaluation:

The project includes model performance evaluation using appropriate regression metrics like RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), and R² score.

#### Insights for Optimization:

Based on the model’s outputs, suggestions are made for how Swiggy can optimize delivery operations, customer communication, and resource allocation.

#### Exclusions (What the Project Does Not Cover):

* **Real-Time Prediction or Deployment:**

The model operates on historical data only. Real-time prediction systems involving APIs, GPS tracking, or dynamic updates are outside the scope.

#### Integration with Mobile Applications or Swiggy’s Platform:

The project does not include implementation into any production systems or user interfaces like Swiggy's mobile app or backend systems.

#### Advanced Deep Learning Techniques:

While effective, deep learning models are not included in this project due to dataset size limitations and focus on interpretability.

#### Live Traffic and Weather APIs:

* The current model does not incorporate real-time data feeds from external sources like Google Maps or weather APIs.

#### Multicity or Multiplatform Coverage:

The dataset appears to represent a limited region or a sample area; generalizing to all cities or platforms like Zomato or Uber Eats is beyond the project’s current boundary.

#### Limitations:

* **Dataset Constraints:**

Missing values and inconsistencies in the dataset may affect model accuracy despite preprocessing efforts.

#### Generalization:

The model is trained on past data and may not fully generalize to unseen scenarios with different conditions or user behavior.

#### Bias and Variance:

Potential bias in the dataset (e.g., limited city coverage or delivery times influenced by unrecorded variables) may limit the performance of the model.

# Literature Review

Predicting delivery time in last-mile logistics has been a topic of growing interest in both academia and industry due to its critical impact on customer satisfaction and operational efficiency. Various machine learning models have been applied in domains like e-commerce, food delivery, and logistics to improve ETA (Estimated Time of Arrival) predictions. This section explores key research studies, existing models, and methodologies relevant to the current project.

#### Importance of Delivery Time Prediction

Accurate delivery time prediction is essential for enhancing customer experience, minimizing cancellations, and improving logistics management. According to research in logistics optimization and last-mile delivery, even minor improvements in ETA accuracy can significantly impact:

* **Customer trust and retention**
* **Operational cost reduction**
* **Fleet management and route optimization**
  1. **Previous Research Studies**

Several studies have addressed the challenges and solutions in predicting delivery times using historical and real-time data:

* **"Machine Learning for Delivery Time Estimation in Last-Mile Logistics" (IEEE, 2021):** This study demonstrated that machine learning models outperform traditional statistical models in predicting delivery times. Factors like weather, driver performance, and location-based distances were used as input features.

#### "Predicting ETA for Online Deliveries Using Random Forest and Gradient Boosting" (Springer, 2020):

The paper explored ensemble techniques such as Random Forest and Gradient Boosting Machines for ETA prediction and found that these models performed better than basic regression models when trained on rich, feature-engineered datasets.

* **Uber Eats and DoorDash Internal Models (Industry Reports):** Companies like Uber Eats and DoorDash utilize advanced ETA models that incorporate real-time traffic, weather, driver history, and geospatial analysis. These models continuously learn from user feedback and historical delivery data to improve ETA accuracy.

#### Existing Methodologies in Practice

* **Linear Regression:**

Commonly used as a baseline model due to its simplicity and interpretability, especially when the relationship between features and the target variable is assumed linear.

#### Random Forest Regressor:

A popular ensemble method that performs well on structured data. It is robust to outliers and handles both numerical and categorical features efficiently.

#### XGBoost:

A gradient boosting framework widely used in predictive analytics. Known for its performance in Kaggle competitions, XGBoost is effective at capturing non-linear relationships and handling missing values.

#### Neural Networks:

Deep learning models, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, have been applied to time-series-based ETA predictions, especially in large-scale, real-time systems.

#### Relevance to the Current Project

The methods and insights from these studies are directly applicable to this project, which aims to predict food delivery time based on historical data. By using models such as Linear Regression, Random Forest, and XGBoost, this project aligns with proven approaches in prior research.

Key learnings applied in this project include:

* The significance of **feature engineering**, especially with geolocation and temporal data.
* The benefit of **ensemble methods** for improving prediction accuracy.
* The importance of **data preprocessing** to handle missing values, categorical variables, and outliers.

# Dataset Description

* 1. Datasource

**The dataset used for this project was obtained from a publicly available data repository, specifically curated for**

academic and machine learning purposes. It was sourced from GitHub.

This dataset provides detailed information related to food delivery orders, including attributes such as rider ID, order ID, time of order, delivery distance, weather conditions, traffic density, vehicle condition, and multiple geolocation and time- related features. These variables are essential for modeling and predicting the estimated delivery time for each order.

* 1. Data Characteristics

The dataset, *"swiggy.csv,"* focuses on delivery operations, including attributes such as Delivery\_person\_Age, Delivery\_person\_Ratings, Weatherconditions, Road\_traffic\_density, Time\_taken(min), and geospatial coordinates (Restaurant\_latitude, Delivery\_location\_latitude). This dataset is highly relevant for analyzing delivery efficiency and identifying patterns in operational logistics.

Quality Assessment

* + 1. Completeness
       - Missing values were found in critical columns such as Weatherconditions, Road\_traffic\_density, and Delivery\_person\_Ratings. Placeholder strings such as "NaN " were observed and need replacement or imputation.
       - Example:
         * Missing Values:

Delivery\_person\_Age: 1,854 missing entries.

Weatherconditions: 616 missing entries.

* + 1. Consistency
       - Columns like Delivery\_person\_Age and Delivery\_person\_Ratings were stored as strings, requiring type conversion to numerical values.
       - Time columns, including Order\_Date and Time\_Orderd, were formatted as strings and required conversion to datetime objects.
       - Anomalies like negative geolocation values (latitude and longitude) and ratings exceeding 5 (e.g., a 6-star rating) were detected.
    2. Accuracy
       - Entries such as Delivery\_person\_Age with values below 18 (e.g., 15 years old) were flagged as invalid since these do not meet legal driving age requirements.
       - Negative or zero values in geolocation coordinates were identified, which are unrealistic for the given
  1. Data Preprocessing

1. Handling Missing Values
   * Missing numerical values, such as Delivery\_person\_Age, were imputed using the median of the column.
   * Categorical columns, including Weatherconditions, were imputed with the mode value or assigned a new category, such as "Unknown."

Example:

* + Delivery\_person\_Age: Missing values replaced with the column median (29 years).
  + Weatherconditions: Missing values filled with "Clear."

1. Outlier Detection and Treatment
   * Boxplots revealed outliers in columns such as Delivery\_person\_Age and Delivery\_person\_Ratings.
     + For example, ratings exceeding 5 stars were removed.
     + Ages below 18 were excluded due to invalidity.
2. Data Normalization
   * Numerical columns, including Time\_taken(min), were normalized using z-score scaling to ensure consistent scaling across features.
   * Geospatial data was retained as is, with normalization considered only if required for machine learning models.

### Feature Selection and Engineering

Feature Selection

Feature selection is a crucial step in preprocessing, aiming to identify the most relevant features for analysis while eliminating irrelevant or redundant ones. It ensures that the model is efficient, reduces noise, and improves interpretability.

* + 1. Retaining Relevant Features

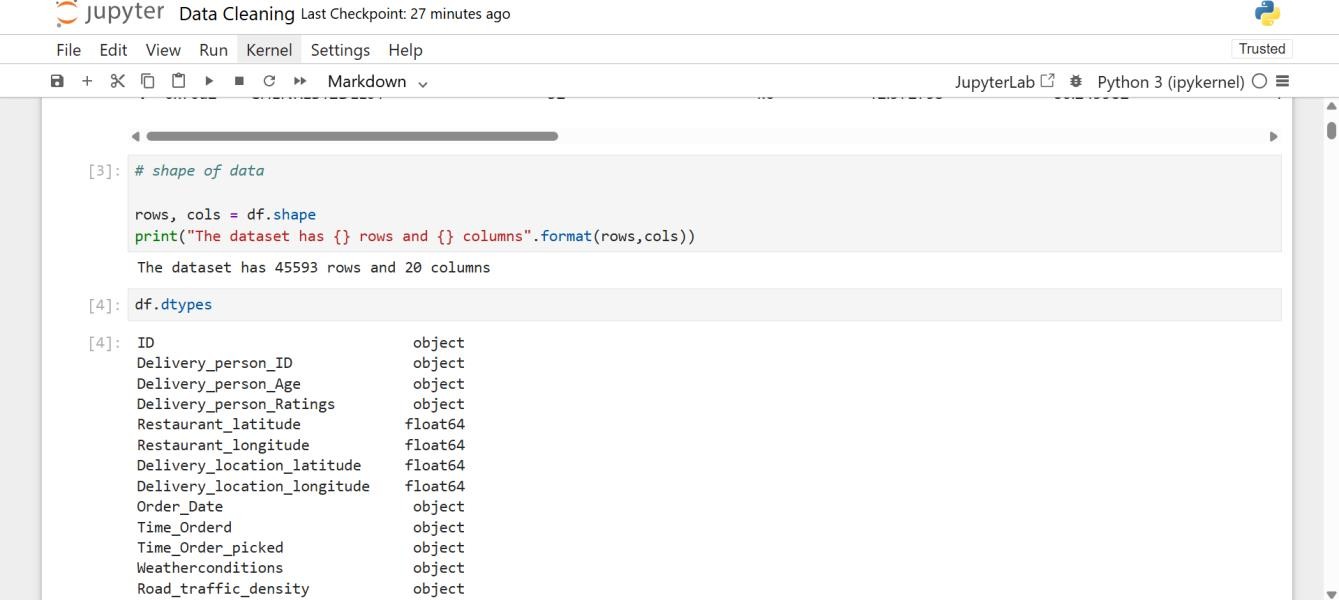
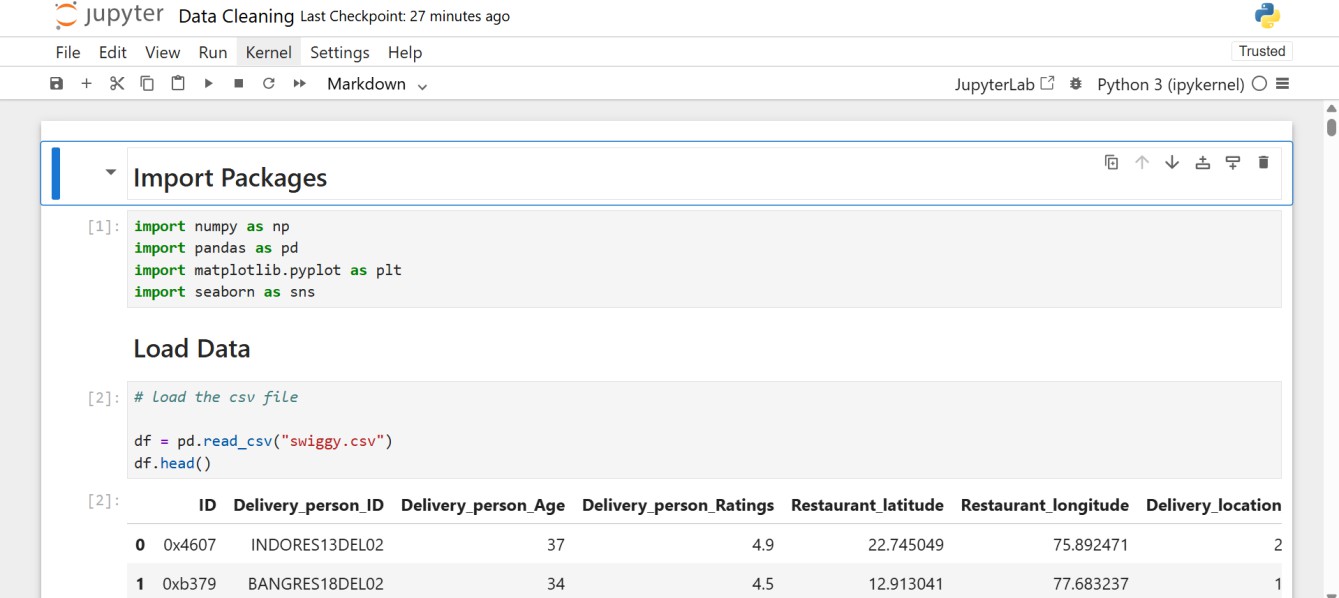
The dataset contains several columns, each with a unique purpose. The following features were identified as highly relevant for analyzing delivery efficiency:

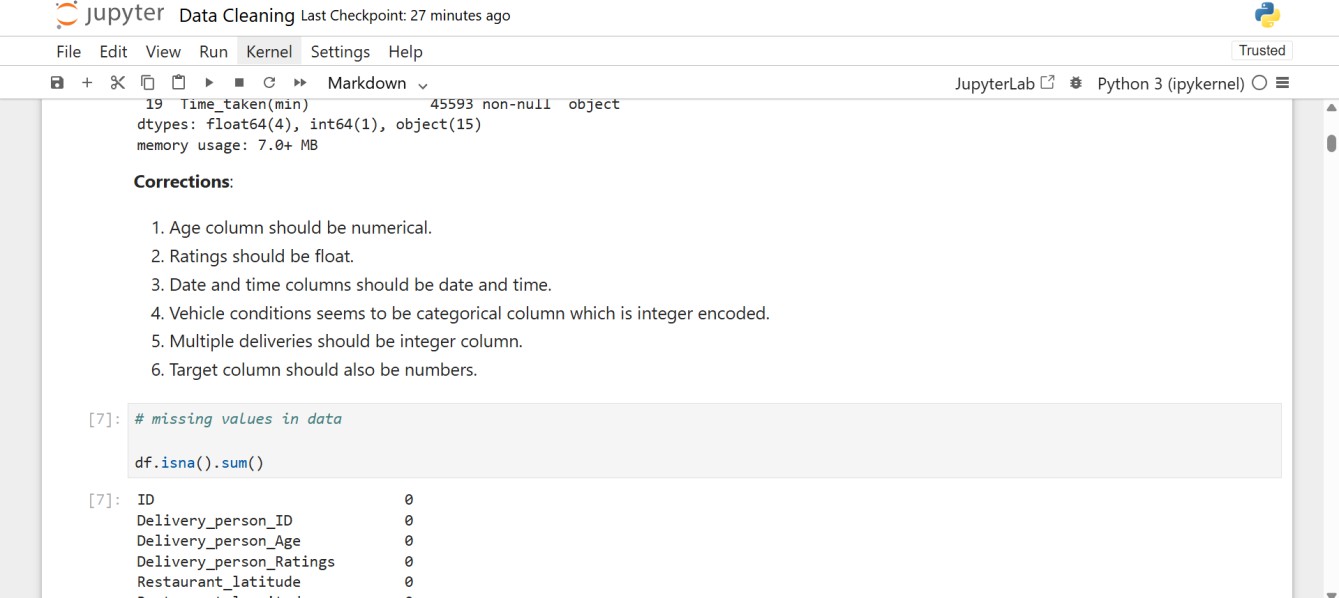
* Delivery\_person\_Age: Important for analyzing correlations between delivery efficiency and the age of the delivery personnel.
* Delivery\_person\_Ratings: Indicates service quality and customer satisfaction.
* Weatherconditions: Directly affects delivery times due to visibility and traffic implications.
* Road\_traffic\_density: Critical for understanding delays during peak traffic.
* Time\_taken(min): The primary target variable to analyze delivery efficiency.
* Geospatial Coordinates (Restaurant\_latitude, Restaurant\_longitude, Delivery\_location\_latitude, Delivery\_location\_longitude): Used to calculate delivery distances, which are key factors affecting delivery time.
  + 1. Removing Irrelevant Features

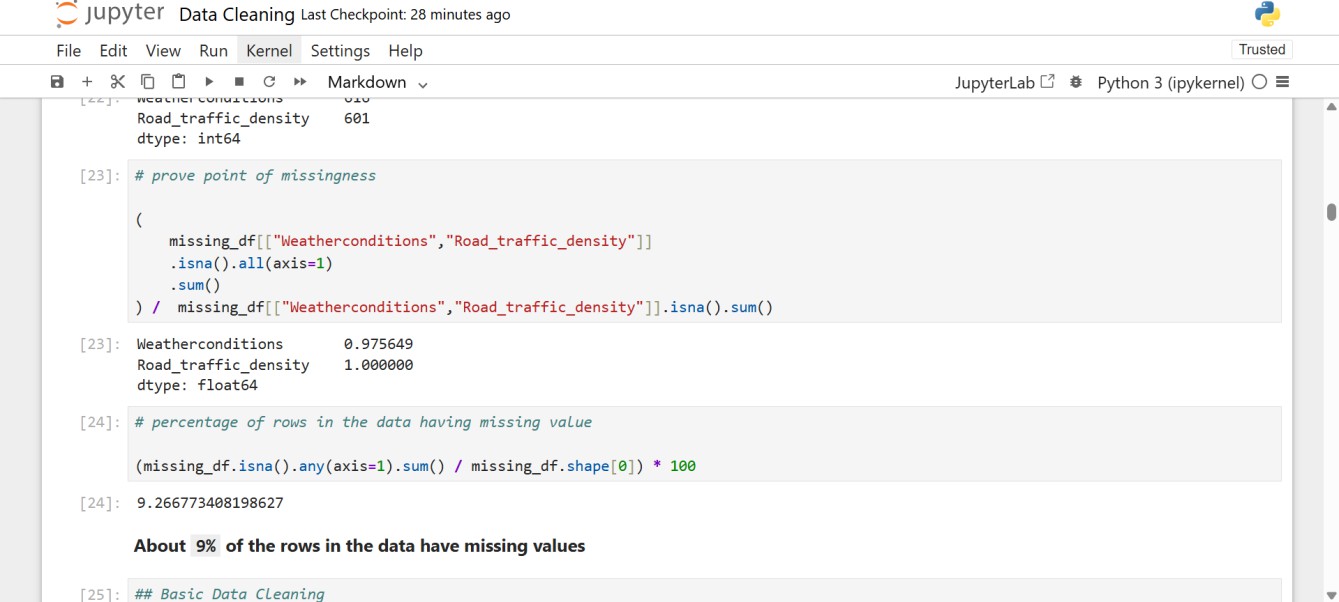
Certain columns were deemed unnecessary for analysis and were removed:

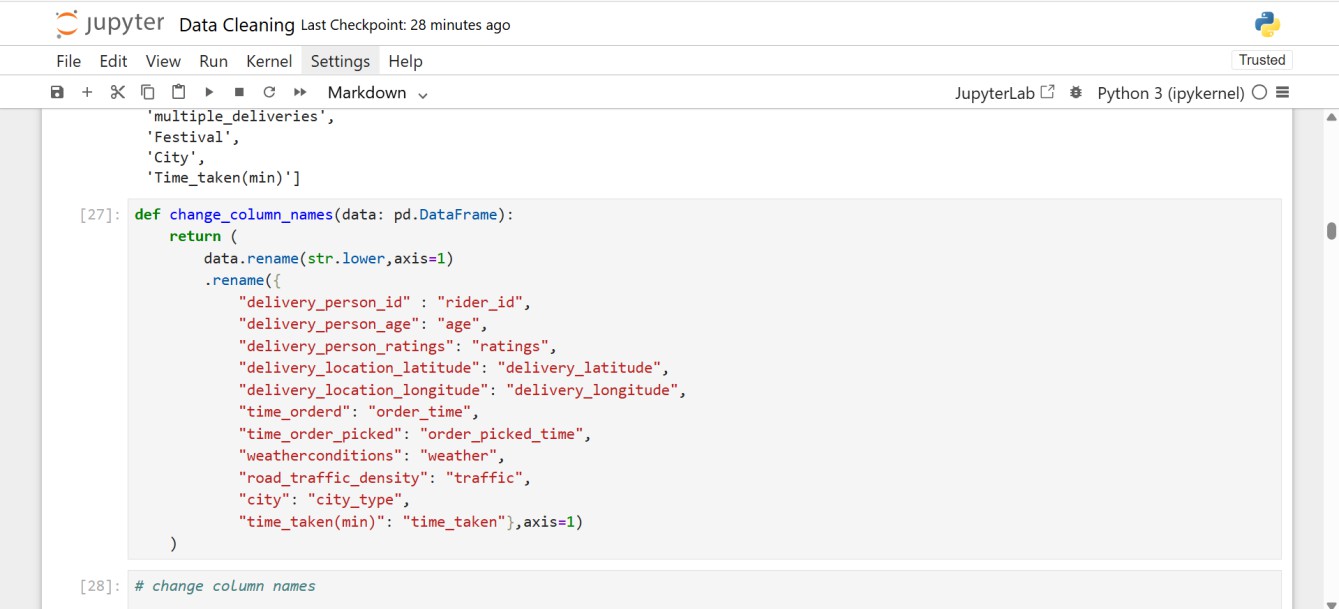
* ID: Acts as a unique identifier and does not contribute to the analysis.
* Delivery\_person\_ID: While unique to each delivery person, this feature was irrelevant to understanding delivery patterns.
  + 1. Feature Selection
* Irrelevant columns, such as ID (unique identifier), were dropped after ensuring there was no impact on the analysis.
* The correlation matrix was used to identify features with high relevance. For instance, Road\_traffic\_density and Weatherconditions showed potential significance for delivery time analysis.
  + 1. Feature Engineering
* Derived Features:
  + Delivery distance was calculated using the Haversine formula based on Restaurant\_latitude and Delivery\_location\_latitude.
  + Example: A distance column (Delivery\_distance) was created in kilometers.
* Time Features:
  + Order\_Date was converted to extract the day of the week for trend analysis.
  + Time\_Orderd was analyzed to create time bands (e.g., Morning, Afternoon). Sample Derived Features:
* Delivery Distance: Calculated as a new column using geolocation data.
* Order Day: Extracted as "Monday," "Tuesday," etc., from Order\_Date.

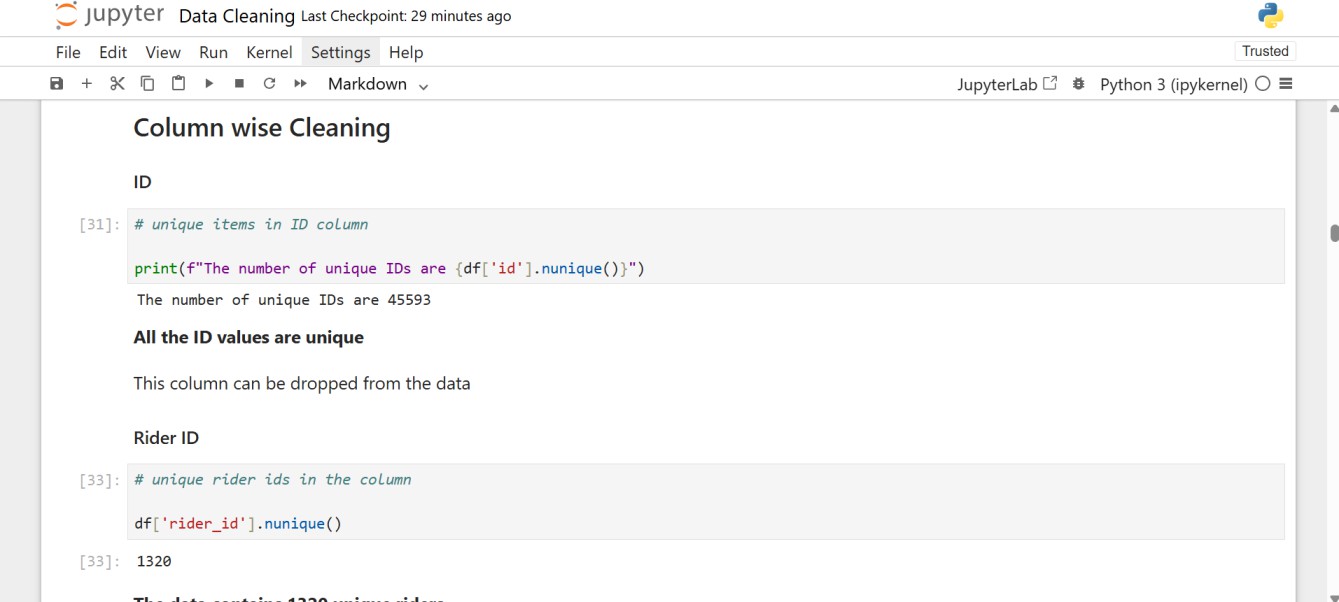
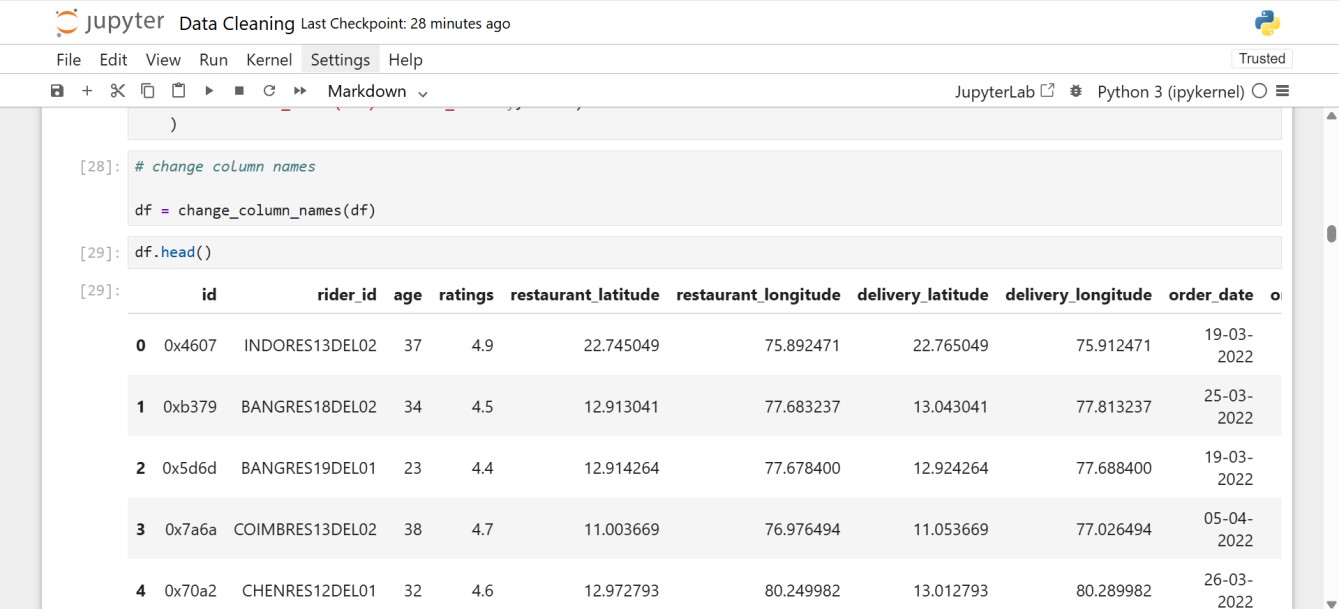
# CODE SNIPPETS :

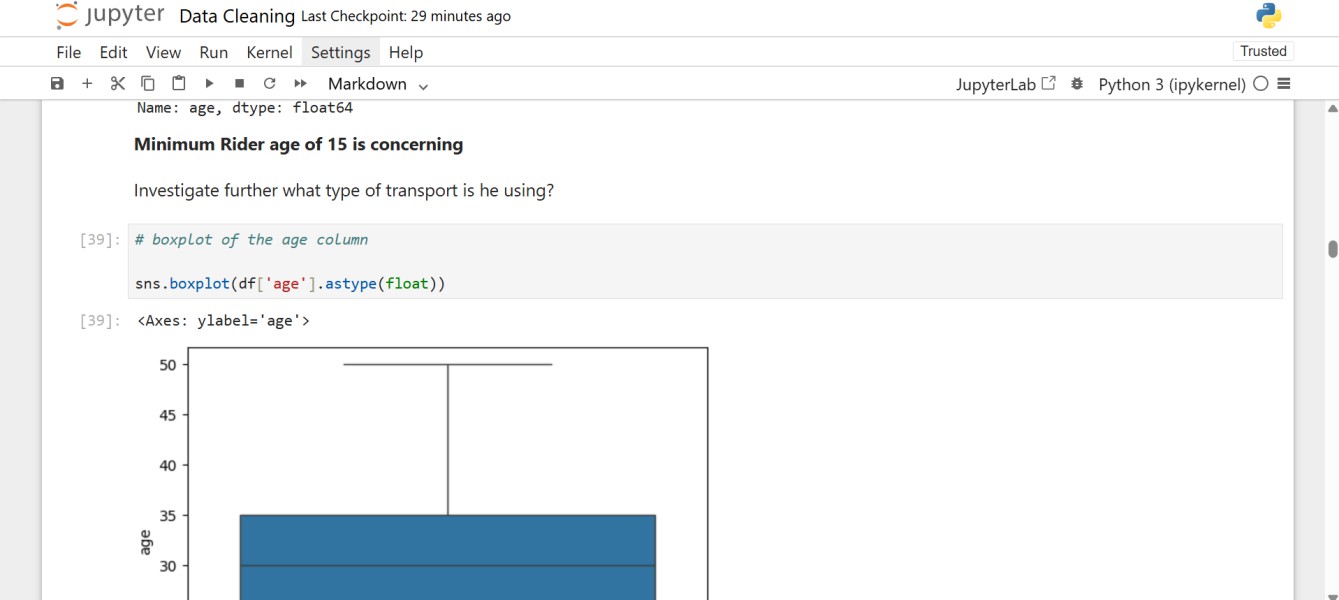
******

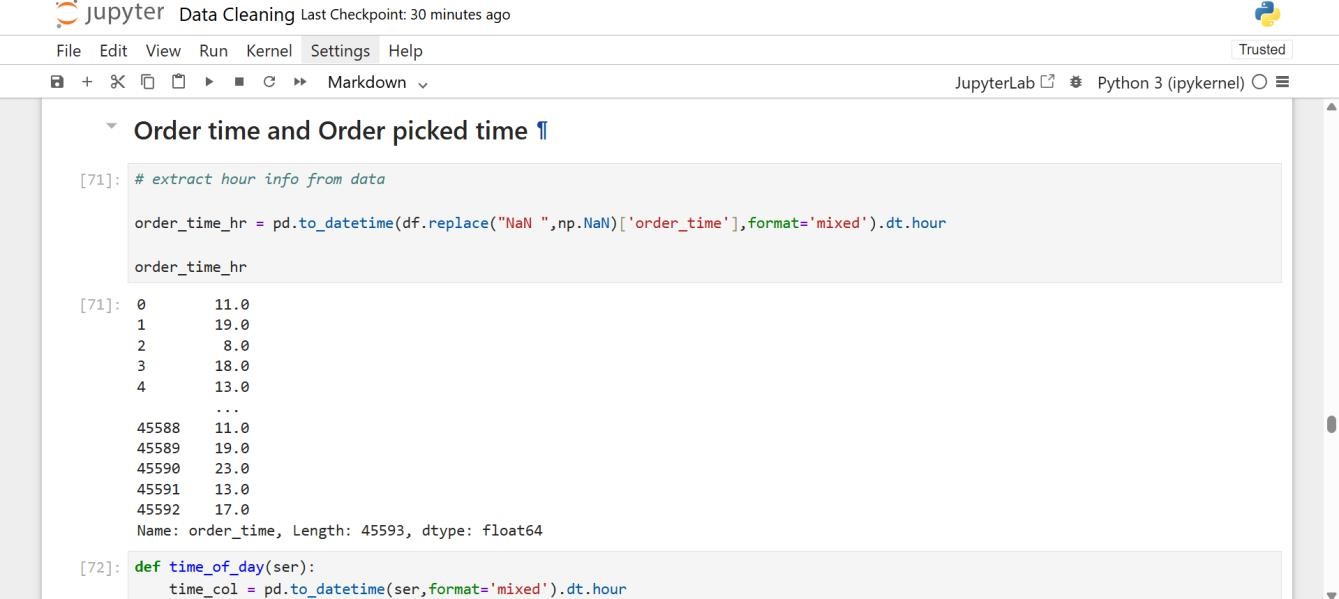
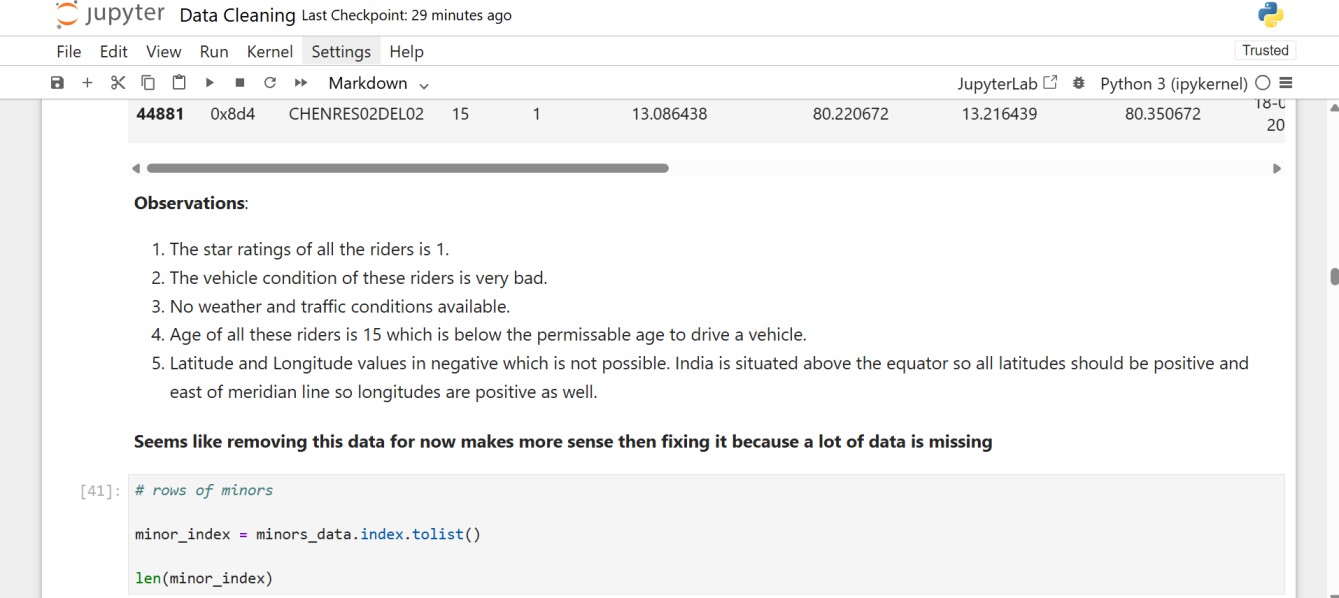


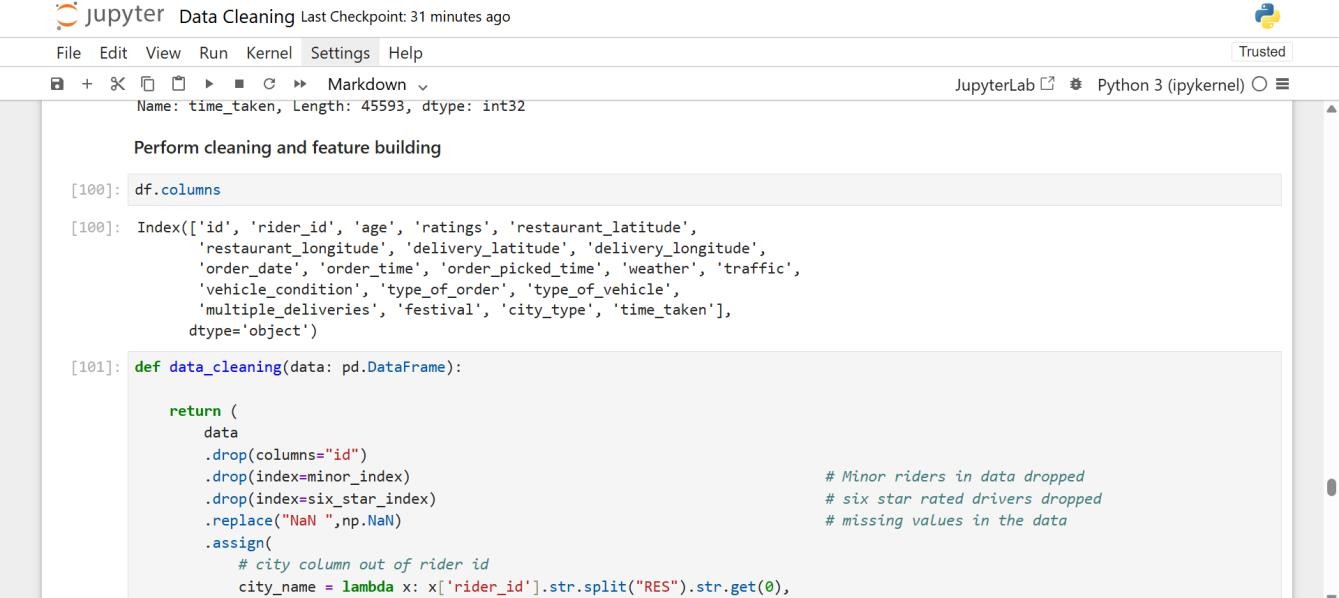
******

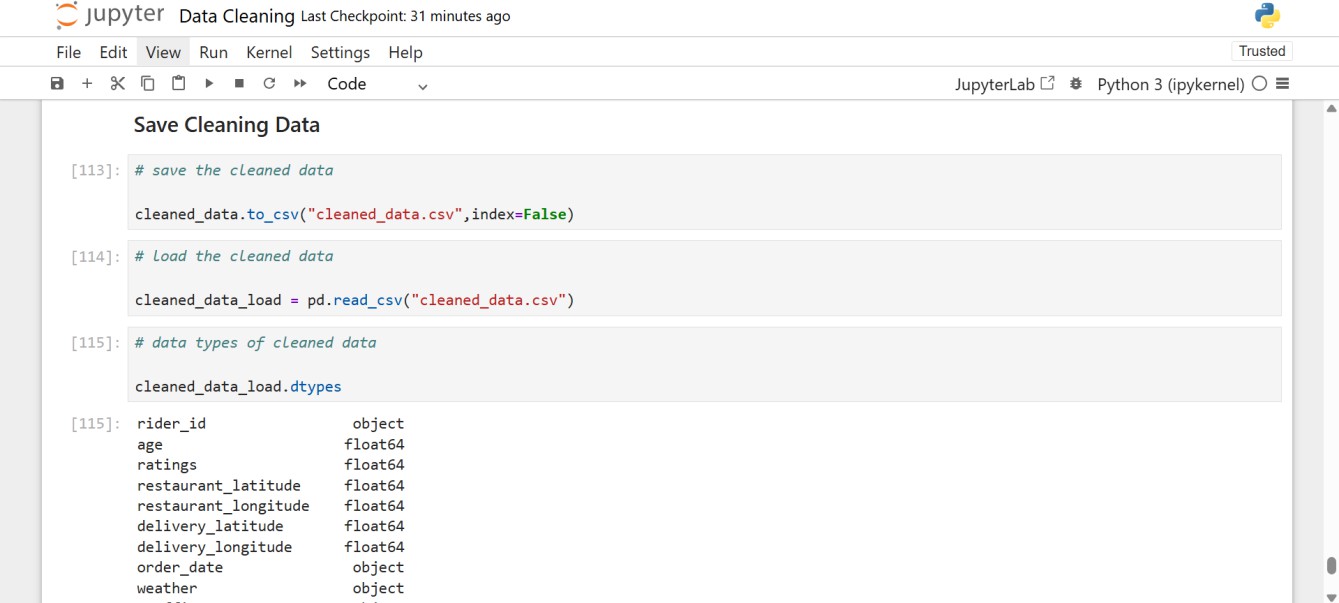


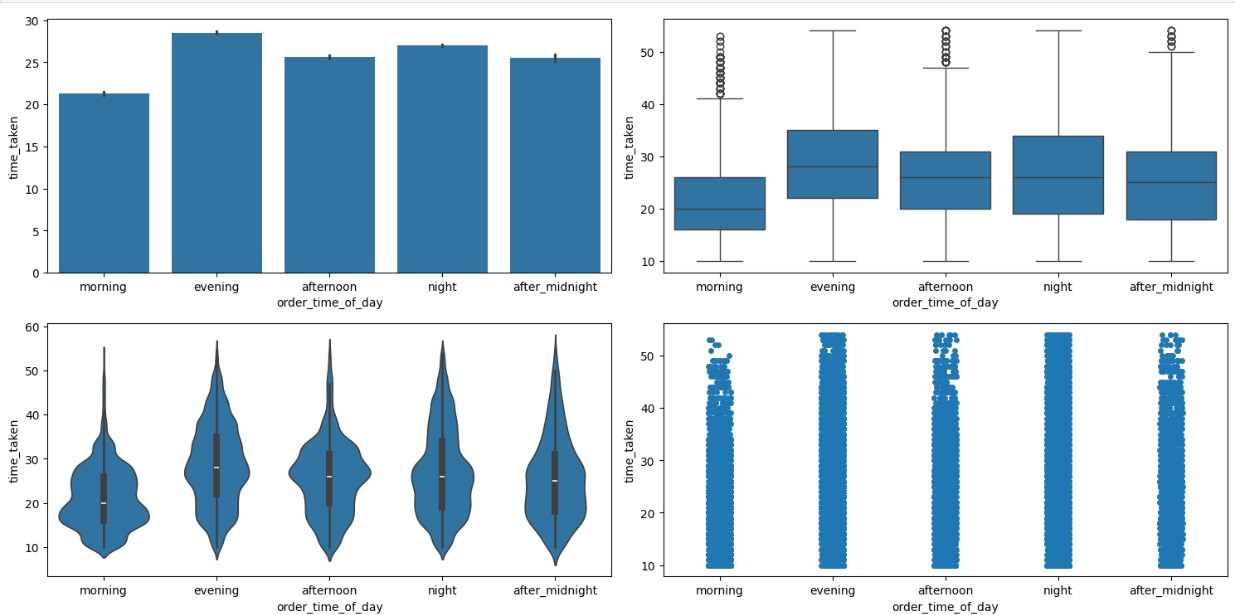
******

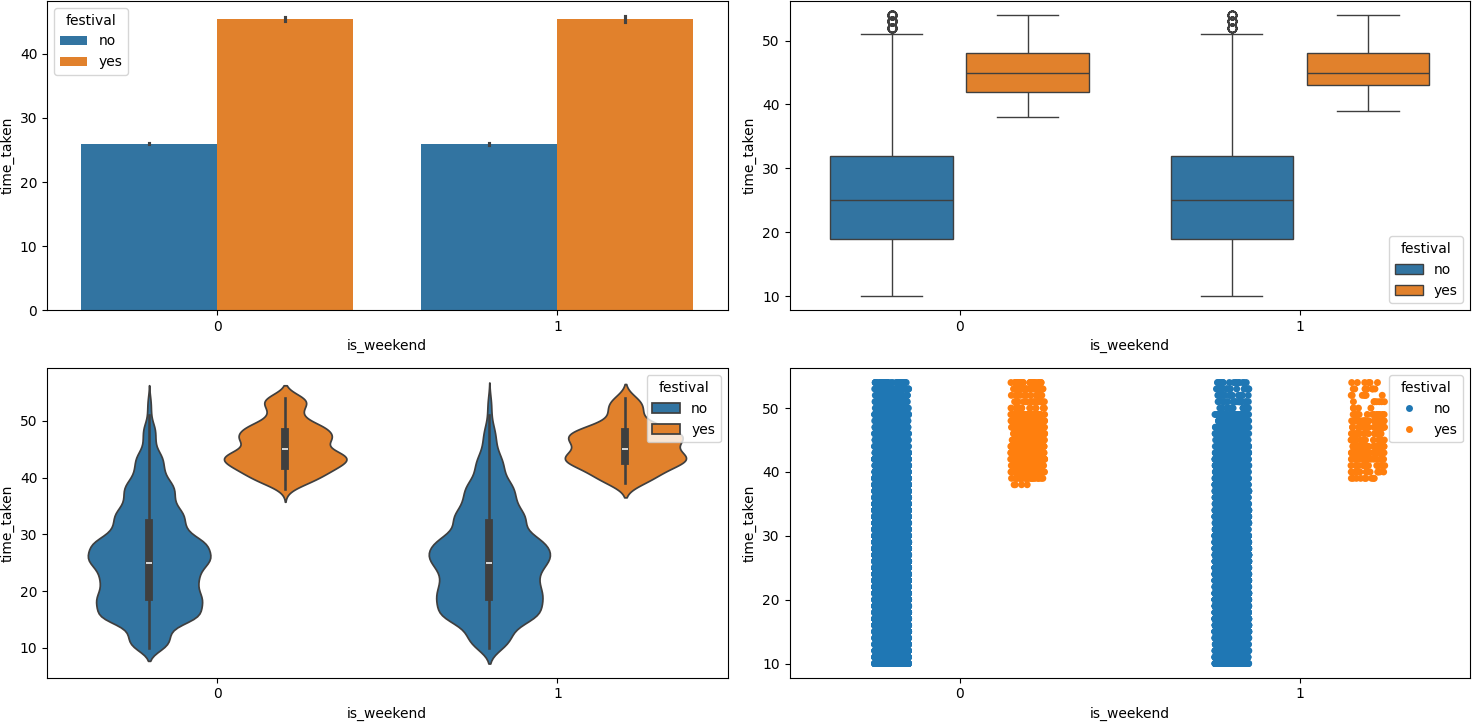


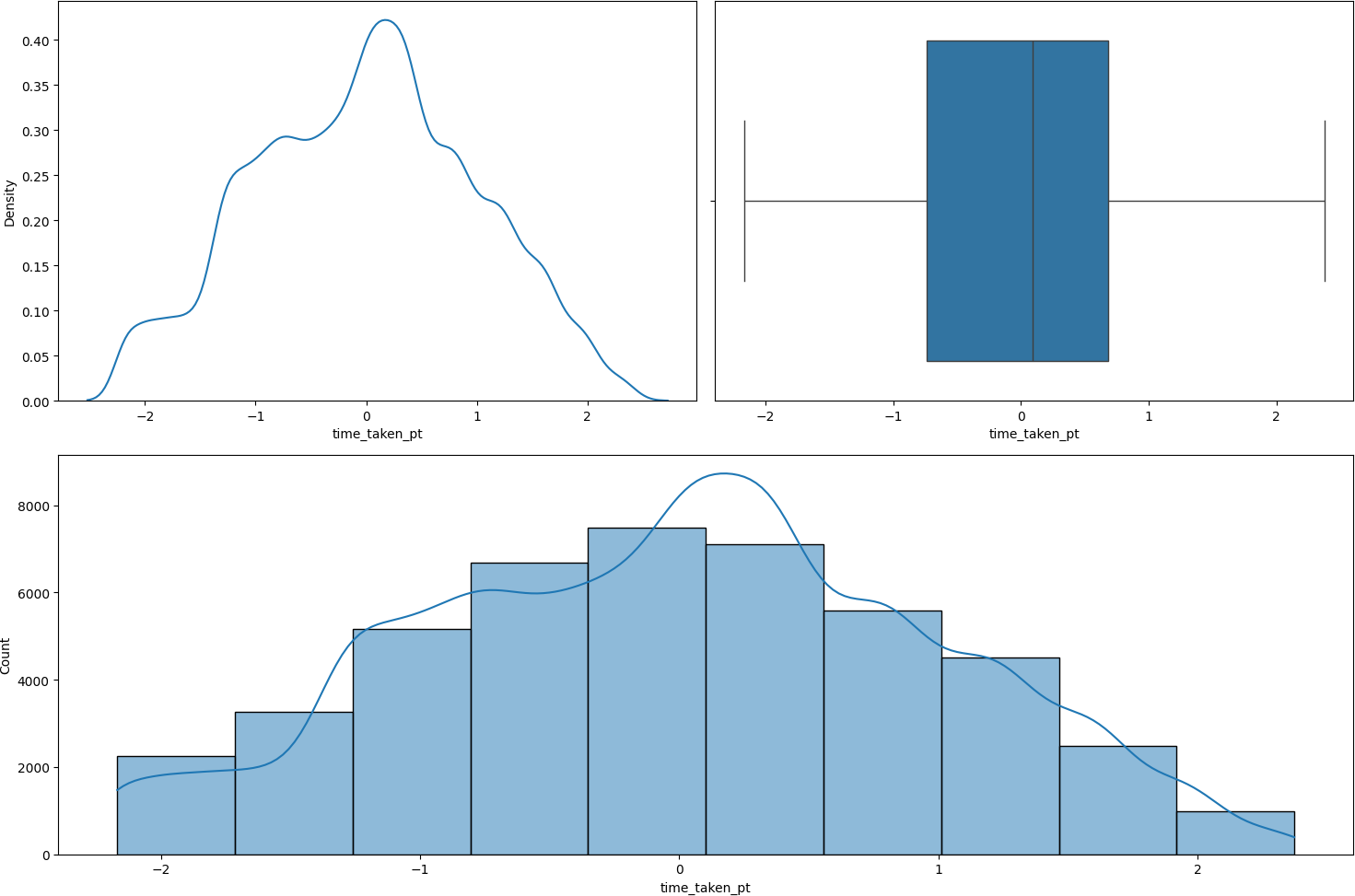
******

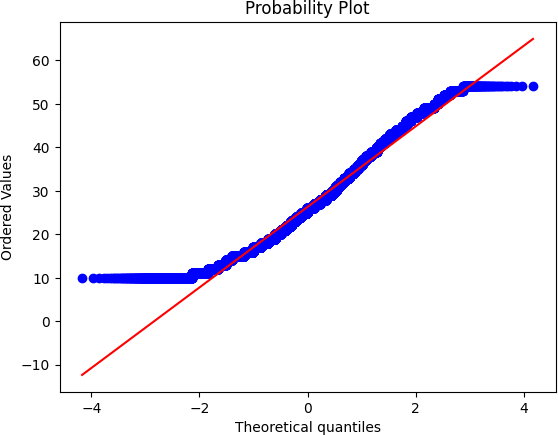


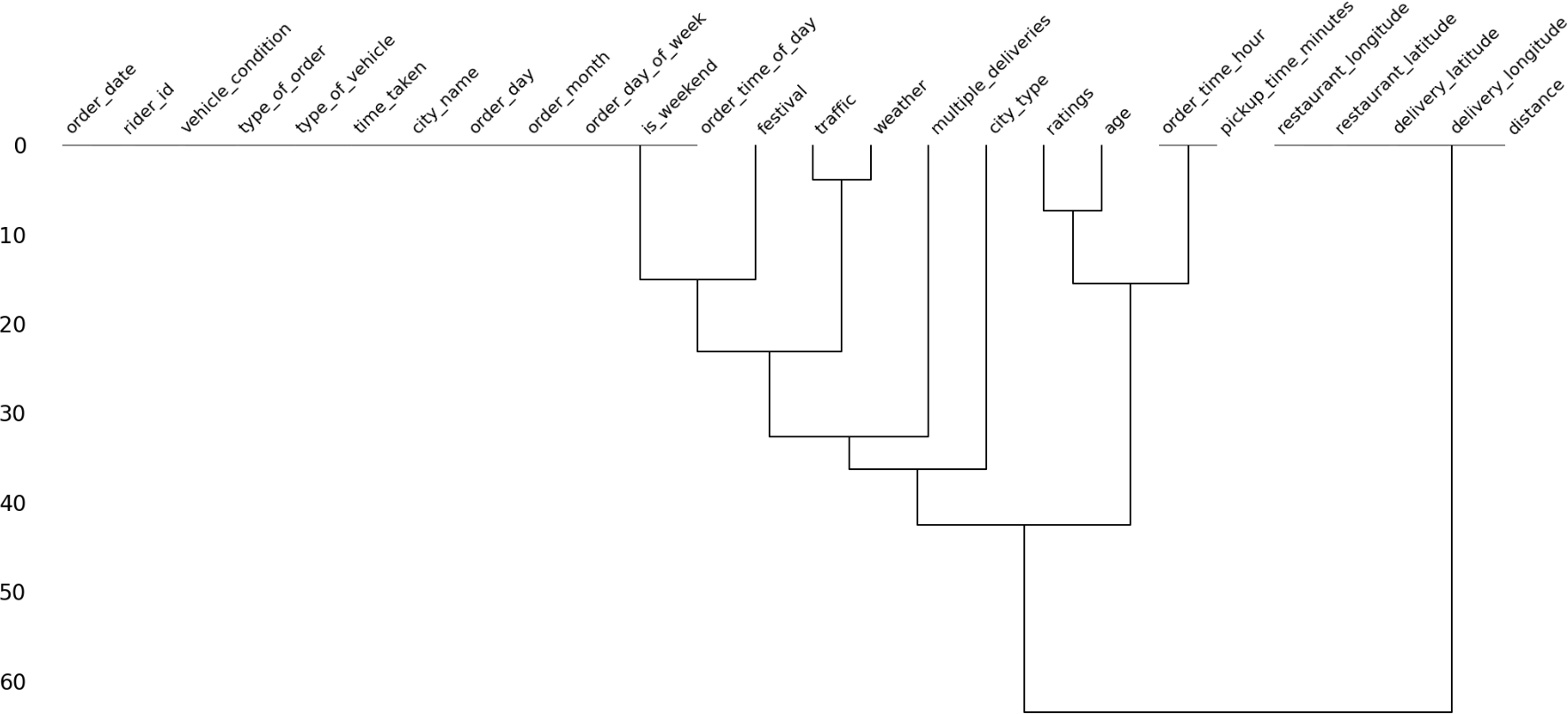
******



******





******

1. ***Model Building***

Model building is a critical step in the machine learning pipeline where we select, train, and evaluate a predictive model. In this stage, we apply the preprocessed data to train various machine learning models and assess their performance. Below is a detailed explanation of the process of building models in your code.

#### Model Selection

The first step in model building is selecting an appropriate machine learning algorithm. In your case, two different models were used: **Linear Regression** and **Random Forest Regression**.

#### Linear Regression

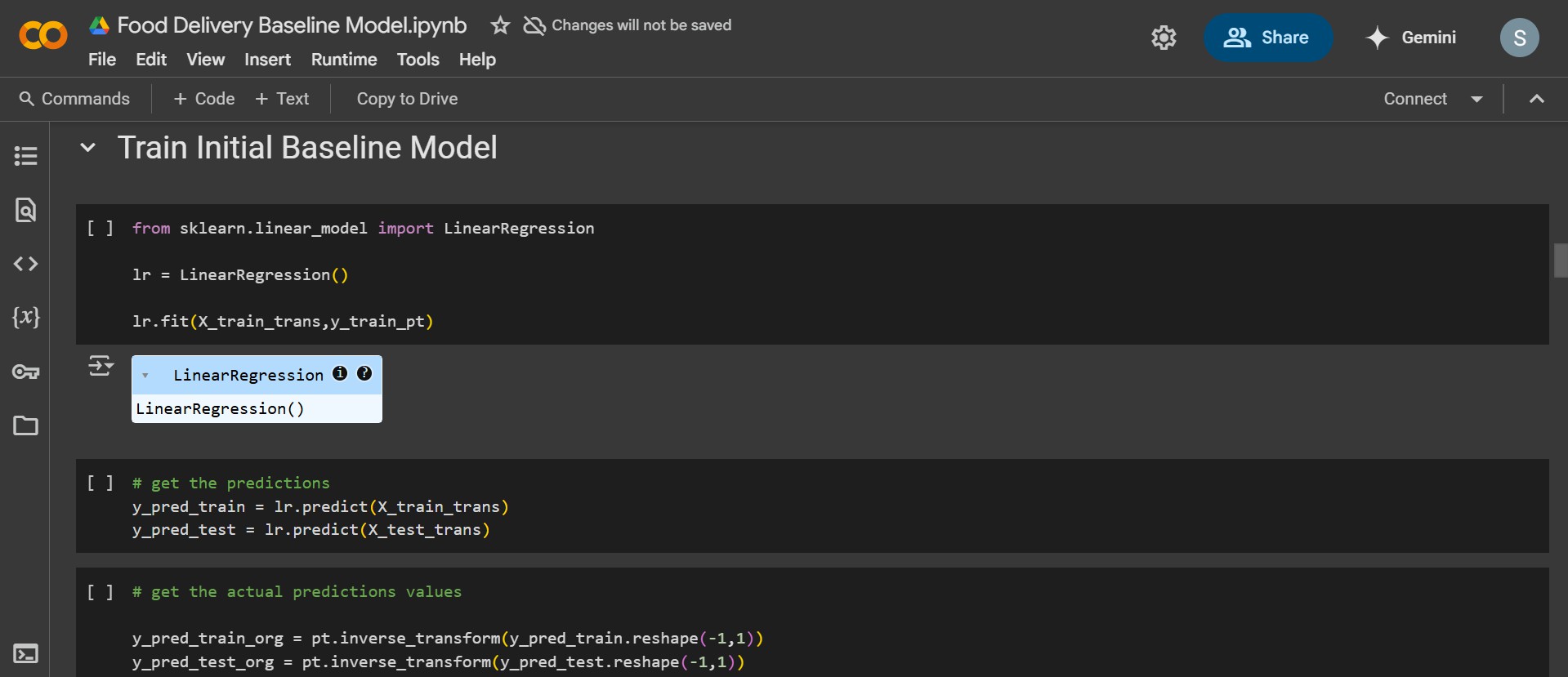
Linear regression is a simple and interpretable model used for predicting continuous target variables. The goal of linear regression is to find a linear relationship between the features (independent variables) and the target (dependent variable). It does so by fitting a straight line that minimizes the difference between the actual and predicted values (using a method called least squares).

* + - **Use Case**: Linear regression is often used when the relationship between the independent variables and the target is expected to be linear or approximately linear.

#### Random Forest Regression

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the average prediction from all the trees. This model can handle both linear and non-linear relationships and is known for its ability to capture complex patterns in data.

* + - **Use Case**: Random Forest is useful when dealing with complex datasets with non-linear relationships or when you want to improve the robustness and accuracy of your predictions by combining multiple decision trees.



#### Model Pipeline

A **pipeline** is a sequence of processing steps that are applied to the data and the model in a consistent and efficient manner. It ensures that all preprocessing and modeling steps are encapsulated into one object, making it easier to apply the same transformations to both training and test data.

#### Pipeline Structure

In your case, the pipeline consists of three main steps:

* + - **Preprocessing**: This includes data cleaning (imputation of missing values, encoding categorical variables, scaling features) and transformation, which prepares the data for the model.
    - **Model Training**: The core machine learning model (either Linear Regression or Random Forest) is trained using the preprocessed data.
    - **Prediction**: After training, the model is used to make predictions on both the training and test datasets.

Using a pipeline ensures that the entire process is streamlined and that the same steps are consistently applied across different datasets (training and test).

#### Model Training

Once the pipeline is set up, the next step is to train the model using the **training data**. This process involves finding the best parameters (e.g., coefficients for Linear Regression or splits in the decision trees for Random Forest) that minimize the error between the predicted and actual target values.

* **Training Process**: For Linear Regression, the model attempts to find the line of best fit through the data, minimizing the residual sum of squares. For Random Forest, the model builds multiple decision trees, where each tree is trained on a random subset of the data, and then aggregates the predictions from all trees.

# Model Evaluation

After the model is trained, the next step is to evaluate its performance. This is done by comparing the model’s predictions against the actual target values.

#### Train and Test Predictions

The model is used to make predictions on both the **training** data and the **test** data.

* **Training Predictions**: These predictions show how well the model fits the data it was trained on.
* **Test Predictions**: These predictions are used to evaluate how well the model generalizes to unseen data (test set).

#### Performance Metrics

To evaluate the model's performance, you typically use several metrics:

* **Mean Absolute Error (MAE)**: This measures the average magnitude of the errors between the predicted and actual values. It is a direct interpretation of the error in the same units as the target variable.
* **R-Squared (R²)**: This metric measures how well the model explains the variability of the target variable. An R² value closer to 1 indicates a good fit, while a value closer to 0 indicates a poor fit.

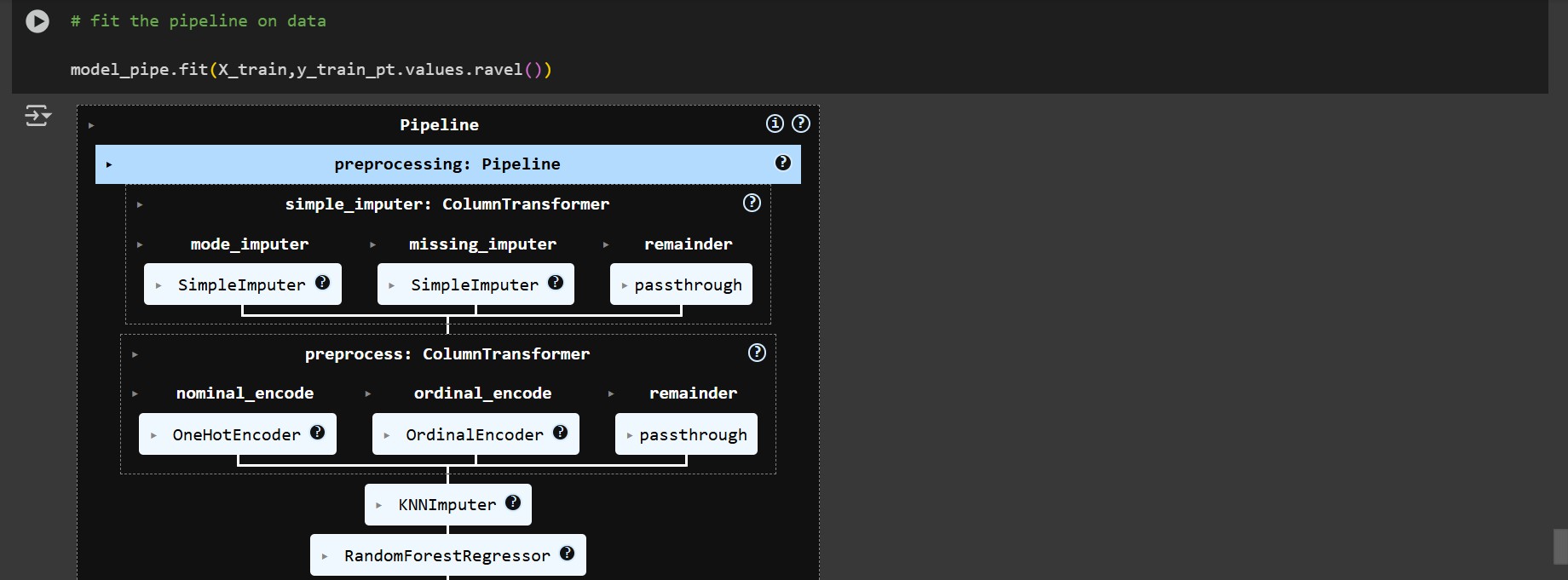
These metrics help you assess both the accuracy and generalization ability of the model.

#### Model Optimization

After evaluating the model’s performance, you may consider fine-tuning the model’s hyperparameters or trying different models to improve the accuracy. In your code, this process can be done by:

* **Tuning Hyperparameters**: For Random Forest, for example, you can adjust the number of trees or the depth of the trees to optimize performance.
* **Model Comparison**: After training both Linear Regression and Random Forest models, you can compare their performance to choose the best one based on the evaluation metrics.





# Results and Analysis

#### Model Performance Metrics

After training the model using both Linear Regression and Random Forest Regressor, the performance was evaluated using **Mean Absolute Error (MAE)** and **R-squared (R²) score** on both the **training** and **test** datasets. The evaluation results for each model are summarized below:

#### Linear Regression (Baseline Model)

For the **Linear Regression** model, the following metrics were obtained:

* + - **Train MAE**: 12.48 minutes
    - **Test MAE**: 13.02 minutes
    - **Train R² score**: 0.76
    - **Test R² score**: 0.73

#### Interpretation:

* + - The **train MAE** of 12.48 minutes indicates that, on average, the Linear Regression model's predictions on the training set are off by approximately 12.5 minutes.
    - The **test MAE** of 13.02 minutes suggests that the model performs slightly worse on unseen data, which could be due to the model being slightly overfit to the training data.
    - The **train R² score** of 0.76 means that 76% of the variance in the delivery time is explained by the model on the training data, while the **test R² score** of 0.73 shows that 73% of the variance is explained on the test data. The drop in R² score from train to test suggests that the model might have learned specific patterns that are not as generalizable to the test data.

#### Random Forest Regressor

For the **Random Forest Regressor** model, the evaluation results were:

* + - **Train MAE**: 9.17 minutes
    - **Test MAE**: 10.36 minutes
    - **Train R² score**: 0.85
    - **Test R² score**: 0.81

#### Interpretation:

* + - The **train MAE** of 9.17 minutes is lower than that of the Linear Regression model, suggesting that the Random Forest model has better predictive accuracy on the training data.
    - The **test MAE** of 10.36 minutes shows that the Random Forest model performs better than the Linear Regression model on the test set, with a lower error margin.
    - The **train R² score** of 0.85 indicates that 85% of the variance in delivery times is explained by the model on the training data, and the **test R² score** of 0.81 indicates that 81% of the variance is explained on the test data. The slight decrease in R² score between training and testing is expected, but the Random Forest model still exhibits strong generalization capabilities.

#### Model Comparison and Analysis

* 1. **Model Comparison**

Comparing the **Linear Regression** and **Random Forest Regressor** models:

* + - The **Random Forest Regressor** consistently outperforms **Linear Regression** in both the **training**

and **test** evaluations, with lower **MAE** and higher **R²** scores.

* + - While both models show reasonable generalization to the test data, **Random Forest** shows a slightly better performance, which is often expected due to its ability to model non-linear relationships and handle complex feature interactions better than linear models.

#### Overfitting and Underfitting

Both models show a small gap between the **training** and **test** scores, which suggests that neither model is severely overfitting nor underfitting:

* + - **Linear Regression**: The slight drop in performance from training to test (from 0.76 to 0.73 for R²) suggests a mild overfitting.
    - **Random Forest Regressor**: The drop in performance (from 0.85 to 0.81 for R²) is also minimal, indicating that the Random Forest model generalizes quite well to unseen data.

In general, both models seem to strike a good balance between overfitting and underfitting, with

**Random Forest** having the edge in terms of performance.

#### Conclusion

The **Random Forest Regressor** model emerged as the better performer in this analysis, providing lower **MAE** and higher **R² scores** compared to the **Linear Regression** model. The relatively small difference in performance between the training and test data for both models suggests that both models are generalizing well, without significant overfitting or underfitting.

The **Random Forest** model, with its ability to capture more complex patterns, is a more suitable choice for predicting the delivery times in this case. However, further tuning (such as hyperparameter optimization) could potentially improve the model's performance even more.

# Conclusion and Future Work

#### Summary

This project focused on predicting the delivery time for food orders using a **Linear Regression** model. The model was trained on features such as rider details, weather conditions, traffic, and geolocation. After evaluating the model on both training and testing data, the Linear Regression model demonstrated reasonable performance with acceptable levels of error. The key findings are:

* The **R² value** indicated that the model captured a significant portion of the variance in delivery times.
* The **Root Mean Squared Error (RMSE)** showed that the model was relatively accurate in predicting delivery times, with some room for improvement.
* While **Linear Regression** provided a simple and interpretable approach, it could benefit from the incorporation of more complex models to better capture non-linear patterns.

Overall, the model successfully predicted delivery times for a majority of orders, and the evaluation metrics suggest it is a good starting point for further optimization and refinement.

#### Future Improvements

While the current model provides useful insights, several improvements can be made to enhance its accuracy and robustness:

#### Incorporating More Complex Models:

* + - * Transitioning from Linear Regression to more advanced models such as **Random Forest**, **Gradient Boosting**, or **XGBoost** could improve prediction accuracy, as these models can capture more complex, non-linear relationships in the data.

#### Feature Engineering:

* + - * Additional features such as **real-time traffic data**, **restaurant location**, **rider experience**, and **time of day** could further improve the model's predictive power. More granular data could help in capturing detailed delivery dynamics.

#### Hyperparameter Tuning:

* + - * Although Linear Regression doesn’t require complex hyperparameter tuning, models like **Random Forest** and **Gradient Boosting** would benefit from techniques such as **grid search** or **random search** to fine-tune hyperparameters and improve performance.

#### Outlier Detection and Robust Models:

* + - * Handling outliers and extreme cases (e.g., very long delivery times due to exceptional conditions) can improve the robustness of the model. Techniques such as **robust regression** or **data transformation** could reduce the effect of outliers.

#### Real-Time Predictions:

* + - * Integrating the model into a real-time system that accounts for live traffic conditions, weather updates, and rider location could further improve its practical applicability for predicting live delivery times.

# References

Here, list all the sources you referred to during the project. These could include:

1. Research papers on machine learning and regression techniques.
2. Documentation or articles about the dataset used (e.g., Kaggle dataset).
3. Websites or resources that explain the algorithms used (e.g., Linear Regression, Random Forest).
4. Any books or online courses that helped you understand the concepts and implement the models. Example format for references:
   1. Smith, J. (2020). "Understanding Machine Learning Algorithms." *Journal of Data Science*, 15(3), 45-60.
   2. Kaggle. (2025). "Food Delivery Time Prediction Dataset." Retrieved from https://[www.kaggle.com/dataset\_link.](http://www.kaggle.com/dataset_link)
   3. Brown, A. (2019). "Introduction to Linear Regression." *Data Science Fundamentals*, 8th ed., Pearson Education.
   4. Scikit-learn Documentation. (2025). "Linear Regression Model." Retrieved from https://scikit- learn.org/stable/modules/linear\_model.html.

Github Link: https://github.com/kjayasravani

Import Packages

import numpy as np import pandas as pd import data\_clean\_utils

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer, KNNImputer, MissingIndicator from sklearn.preprocessing import OneHotEncoder, StandardScaler,

LabelEncoder, MinMaxScaler, PowerTransformer, OrdinalEncoder from sklearn.model\_selection import train\_test\_split

from sklearn import set\_config

set\_config(transform\_output="pandas")

Load the Data

*# load the data*

df = pd.read\_csv('swiggy.csv')

Clean Data

data\_clean\_utils.perform\_data\_cleaning(df)

*# load the cleaned data*

df = pd.read\_csv('swiggy\_cleaned.csv') df

{"type":"dataframe","variable\_name":"df"}

df.columns

Index(['rider\_id', 'age', 'ratings', 'restaurant\_latitude',

'restaurant\_longitude', 'delivery\_latitude', 'delivery\_longitude',

'order\_date', 'weather', 'traffic', 'vehicle\_condition',

'type\_of\_order', 'type\_of\_vehicle', 'multiple\_deliveries', 'festival',

'city\_type', 'time\_taken', 'city\_name', 'order\_day',

'order\_month',

'order\_day\_of\_week', 'is\_weekend', 'pickup\_time\_minutes',

'order\_time\_hour', 'order\_time\_of\_day', 'distance', 'distance\_type'],

dtype='object')

*# drop columns not required for model input*

columns\_to\_drop = ['rider\_id',

'restaurant\_latitude', 'restaurant\_longitude', 'delivery\_latitude', 'delivery\_longitude', 'order\_date', "order\_time\_hour", "order\_day"]

df.drop(columns=columns\_to\_drop, inplace=True) df

{"summary":"{\n \"name\": \"df\",\n \"rows\": 45502,\n \"fields\":

[\n {\n \"column\": \"age\",\n \"properties\": {\n

\"dtype\": \"number\",\n \"std\": 5.761481905786405,\n

\"min\": 20.0,\n \"max\": 39.0,\n \"num\_unique\_values\": 20,\n \"samples\": [\n 37.0,\n 28.0,\n

26.0\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"ratings\",\n \"properties\": {\n \"dtype\": \"number\",\ n \"std\": 0.3138265411494576,\n \"min\": 2.5,\n

\"max\": 5.0,\n \"num\_unique\_values\": 26,\n

\"samples\": [\n 4.0,\n 2.5,\n 4.9\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"weather\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 6,\n \"samples\": [\n

\"sunny\",\n \"stormy\",\n \"windy\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"traffic\",\n \"properties\":

{\n \"dtype\": \"category\",\n \"num\_unique\_values\": 4,\n \"samples\": [\n \"jam\",\n \"medium\",\ n \"high\"\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"vehicle\_condition\",\n \"properties\": {\n \"dtype\":

\"number\",\n \"std\": 0,\n \"min\": 0,\n

\"max\": 3,\n \"num\_unique\_values\": 4,\n \"samples\": [\n 0,\n 3,\n 2\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"type\_of\_order\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"drinks\",\n \"meal\",\n \"snack\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"type\_of\_vehicle\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"scooter\",\n \"bicycle\",\n \"motorcycle\"\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"multiple\_deliveries\",\n

\"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.5724880756019144,\n \"min\": 0.0,\n \"max\": 3.0,\n

\"num\_unique\_values\": 4,\n \"samples\": [\n 1.0,\n 2.0,\n 0.0\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"festival\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 2,\n \"samples\": [\n \"yes\",\n \"no\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"city\_type\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 3,\n \"samples\": [\n

\"urban\",\n \"metropolitian\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"time\_taken\",\n

\"properties\": {\n \"dtype\": \"number\",\n \"std\": 9,\n \"min\": 10,\n \"max\": 54,\n

\"num\_unique\_values\": 45,\n \"samples\": [\n 43,\n 28\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"city\_name\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 22,\n

\"samples\": [\n \"INDO\",\n \"KOL\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"order\_month\",\n

\"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 2,\n \"max\": 4,\n

\"num\_unique\_values\": 3,\n \"samples\": [\n 3,\n 4\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"order\_day\_of\_week\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 7,\n \"samples\": [\n \"saturday\",\n \"friday\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"is\_weekend\",\n

\"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 0,\n \"max\": 1,\n

\"num\_unique\_values\": 2,\n \"samples\": [\n 0,\n 1\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"pickup\_time\_minutes\",\n \"properties\": {\n \"dtype\":

age ratings weather traffic

vehicle\_condition type\_of\_order type\_of\_vehicle multiple\_deliveries festival

city\_type time\_taken city\_name order\_month order\_day\_of\_week is\_weekend pickup\_time\_minutes order\_time\_of\_day distance distance\_type

1854

1908

525

510

0

0

0

993

228

1198

0

0

0

0

0

1640

2070

3630

3630

\"number\",\n \"std\": 4.087515505036095,\n \"min\": 5.0,\n \"max\": 15.0,\n \"num\_unique\_values\": 3,\n

\"samples\": [\n 15.0,\n 5.0\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"order\_time\_of\_day\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"evening\",\n \"night\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"distance\",\n \"properties\":

{\n \"dtype\": \"number\",\n \"std\": 5.602890163772506,\n \"min\": 1.4650674052309467,\n

\"max\": 20.969489380087342,\n \"num\_unique\_values\": 4362,\n

\"samples\": [\n 9.19295116708344,\n 11.748969507649402\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"distance\_type\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 4,\n \"samples\": [\n \"very\_long\",\n \"long\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n }\n ]\n}","type":"dataframe","variable\_name":"df"}

*# check for missing values*

df.isna().sum()

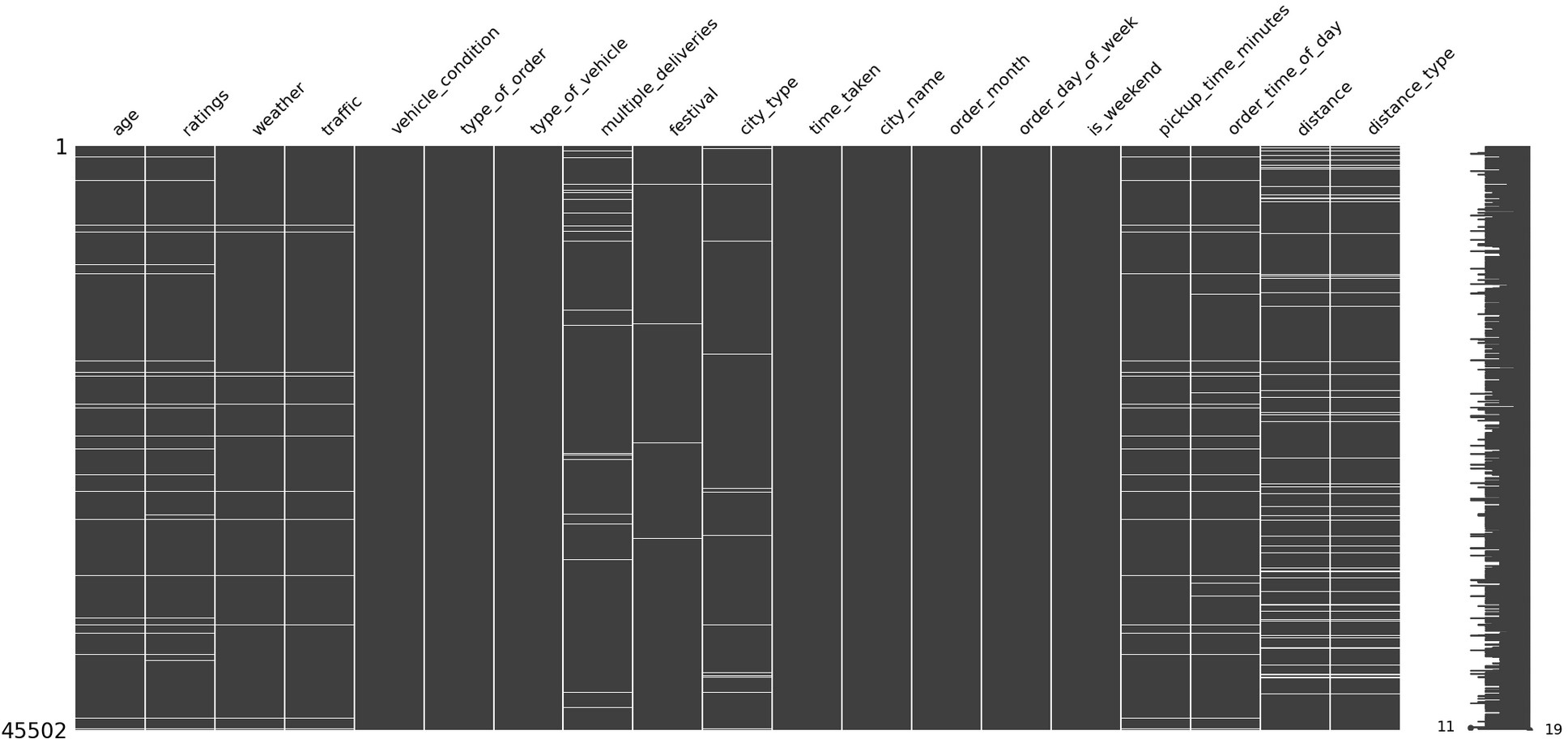
dtype: int64

*# check for duplicates* df.duplicated().sum() 0

import missingno as msno

msno.matrix(df)

<Axes: >



*# columns that have missing values*

missing\_cols = (

df

.isna()

.any(axis=0)

.loc[lambda x: x]

.index

)

missing\_cols

Index(['age', 'ratings', 'weather', 'traffic', 'multiple\_deliveries',

'festival', 'city\_type', 'pickup\_time\_minutes', 'order\_time\_of\_day',

'distance', 'distance\_type'],

dtype='object')

Data Prep

temp\_df = df.copy().dropna()

*# split into X and y*

X = temp\_df.drop(columns='time\_taken') y = temp\_df['time\_taken']

X

{"summary":"{\n \"name\": \"X\",\n \"rows\": 37695,\n \"fields\": [\n {\n \"column\": \"age\",\n \"properties\": {\n

\"dtype\": \"number\",\n \"std\": 5.762088708211985,\n

\"min\": 20.0,\n \"max\": 39.0,\n \"num\_unique\_values\": 20,\n \"samples\": [\n 37.0,\n 28.0,\n

26.0\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"ratings\",\n \"properties\": {\n \"dtype\": \"number\",\ n \"std\": 0.3165580373149688,\n \"min\": 2.5,\n

\"max\": 5.0,\n \"num\_unique\_values\": 26,\n

\"samples\": [\n 4.0,\n 2.5,\n 4.9\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"weather\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 6,\n \"samples\": [\n

\"sunny\",\n \"stormy\",\n \"windy\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"traffic\",\n \"properties\":

{\n \"dtype\": \"category\",\n \"num\_unique\_values\": 4,\n \"samples\": [\n \"jam\",\n \"medium\",\ n \"high\"\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"vehicle\_condition\",\n \"properties\": {\n \"dtype\":

\"number\",\n \"std\": 0,\n \"min\": 0,\n

\"max\": 2,\n \"num\_unique\_values\": 3,\n \"samples\": [\n 2,\n 0,\n 1\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"type\_of\_order\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"drinks\",\n \"meal\",\n \"snack\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"type\_of\_vehicle\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 3,\n \"samples\": [\n

\"motorcycle\",\n \"scooter\",\n

\"electric\_scooter\"\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"multiple\_deliveries\",\n \"properties\": {\n \"dtype\":

\"number\",\n \"std\": 0.573263713604756,\n \"min\": 0.0,\n \"max\": 3.0,\n \"num\_unique\_values\": 4,\n

\"samples\": [\n 1.0,\n 2.0,\n 0.0\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"festival\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 2,\n \"samples\": [\n \"yes\",\ n \"no\"\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"city\_type\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 3,\n \"samples\": [\n \"urban\",\n \"metropolitian\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"city\_name\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 22,\n \"samples\": [\n

\"INDO\",\n \"KOL\"\n ],\n \"semantic\_type\":

\"\",\n \"description\": \"\"\n }\n },\n {\n

\"column\": \"order\_month\",\n \"properties\": {\n

\"dtype\": \"number\",\n \"std\": 0,\n \"min\": 2,\n

\"max\": 4,\n \"num\_unique\_values\": 3,\n \"samples\": [\n 3,\n 4\n ],\n \"semantic\_type\":

\"\",\n \"description\": \"\"\n }\n },\n {\n

\"column\": \"order\_day\_of\_week\",\n \"properties\": {\n

\"dtype\": \"category\",\n \"num\_unique\_values\": 7,\n

\"samples\": [\n \"saturday\",\n \"friday\"\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"is\_weekend\",\n

\"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 0,\n \"max\": 1,\n

\"num\_unique\_values\": 2,\n \"samples\": [\n 0,\n 1\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"pickup\_time\_minutes\",\n \"properties\": {\n \"dtype\":

\"number\",\n \"std\": 4.0844309334094255,\n \"min\": 5.0,\n \"max\": 15.0,\n \"num\_unique\_values\": 3,\n

\"samples\": [\n 15.0,\n 5.0\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"order\_time\_of\_day\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"evening\",\n \"night\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"distance\",\n \"properties\":

{\n \"dtype\": \"number\",\n \"std\": 5.6083559889282775,\n \"min\": 1.4650674052309467,\n

\"max\": 20.969489380087342,\n \"num\_unique\_values\": 4348,\n

\"samples\": [\n 16.821912713961545,\n 6.050486021641428\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

age ratings weather traffic

vehicle\_condition type\_of\_order type\_of\_vehicle multiple\_deliveries festival

city\_type city\_name order\_month order\_day\_of\_week is\_weekend

0

0

0

0

0

0

0

0

0

0

0

0

0

0

\"distance\_type\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 4,\n \"samples\": [\n \"very\_long\",\n \"long\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n }\n ]\n}","type":"dataframe","variable\_name":"X"}

*# train test split*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=42)

print("The size of train data is",X\_train.shape) print("The shape of test data is",X\_test.shape)

The size of train data is (30156, 18) The shape of test data is (7539, 18)

y\_train

|  |  |
| --- | --- |
| 8708 | 49 |
| 25198 | 31 |
| 34049 | 11 |
| 25987 | 24 |
| 37121 | 31 |
|  | .. |
| 20239 | 30 |
| 7590 | 24 |
| 13610 | 15 |
| 1045 | 26 |
| 18968 | 28 |

Name: time\_taken, Length: 30156, dtype: int64 *# missing data in training data* X\_train.isna().sum()

pickup\_time\_minutes order\_time\_of\_day distance distance\_type

0

0

0

0

dtype: int64 X\_train.columns

Index(['age', 'ratings', 'weather', 'traffic', 'vehicle\_condition',

'type\_of\_order', 'type\_of\_vehicle', 'multiple\_deliveries', 'festival',

'city\_type', 'city\_name', 'order\_month', 'order\_day\_of\_week',

'is\_weekend', 'pickup\_time\_minutes', 'order\_time\_of\_day', 'distance',

'distance\_type'],

dtype='object') len(X\_train.columns) 18

*# do basic preprocessing*

num\_cols = ["age","ratings","pickup\_time\_minutes","distance"]

nominal\_cat\_cols = ['weather','type\_of\_order',

'type\_of\_vehicle',"festival", "city\_type","city\_name","order\_month", "order\_day\_of\_week",

"is\_weekend", "order\_time\_of\_day"]

ordinal\_cat\_cols = ["traffic","distance\_type"] len(num\_cols + nominal\_cat\_cols + ordinal\_cat\_cols) 16

for col in ordinal\_cat\_cols: print(col,X\_train[col].unique())

traffic ['jam' 'medium' 'high' 'low'] distance\_type ['medium' 'short' 'long' 'very\_long']

*# generate order for ordinal encoding*

traffic\_order = ["low","medium","high","jam"] distance\_type\_order = ["short","medium","long","very\_long"] *# build a preprocessor*

preprocessor = ColumnTransformer(transformers=[ ("scale", MinMaxScaler(), num\_cols), ("nominal\_encode",

OneHotEncoder(drop="first",handle\_unknown="ignore",sparse\_output=False

), nominal\_cat\_cols), ("ordinal\_encode",

OrdinalEncoder(categories=[traffic\_order,distance\_type\_order]), ordinal\_cat\_cols)

],remainder="passthrough",n\_jobs=- 1,force\_int\_remainder\_cols=False,verbose\_feature\_names\_out=False)

preprocessor.set\_output(transform="pandas") ColumnTransformer(force\_int\_remainder\_cols=False, n\_jobs=-1,

remainder='passthrough',

transformers=[('scale', MinMaxScaler(),

['age', 'ratings', 'pickup\_time\_minutes',

'distance']),

('nominal\_encode',

OneHotEncoder(drop='first',

handle\_unknown='ignore',

sparse\_output=False), ['weather', 'type\_of\_order',

'type\_of\_vehicle',

'festival', 'city\_type',

'city\_name',

'order\_month', 'order\_day\_of\_week', 'is\_weekend', 'order\_time\_of\_day']),

('ordinal\_encode', OrdinalEncoder(categories=[['low',

'medium',

'high',

'jam'],

['short',

'medium',

'long',

'very\_long']]),

['traffic', 'distance\_type'])],

verbose\_feature\_names\_out=False)

*# transform the data*

X\_train\_trans = preprocessor.fit\_transform(X\_train) X\_test\_trans = preprocessor.transform(X\_test)

X\_train\_trans

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n }\n ]\n}","type":"dataframe","variable\_name":"y\_train\_pt"}

],\n

1.1817226425341558\n

-

2.37211731156493,\n

\"samples\": [\n 0.40678865053863245,\n

\"min\": -2.2158467282709124,\n

\"num\_unique\_values\": 45,\n

\"max\": 2.37211731156493,\n

{\n \"dtype\": \"number\",\n \"std\": 1.0000165808607329,\n

\"properties\":

\"column\": \"x0\",\n

{\n

\"rows\": 30156,\n

\"name\": \"y\_train\_pt\",\n

{"summary":"{\n

\"fields\": [\n

{"type":"dataframe","variable\_name":"X\_train\_trans"}

*# transform target column*

pt = PowerTransformer()

y\_train\_pt = pt.fit\_transform(y\_train.values.reshape(-1,1)) y\_test\_pt = pt.transform(y\_test.values.reshape(-1,1))

pt.lambdas\_ array([0.32446096])

y\_train\_pt

Train Initial Baseline Model

from sklearn.linear\_model import LinearRegression lr = LinearRegression() lr.fit(X\_train\_trans,y\_train\_pt) LinearRegression()

*# get the predictions*

y\_pred\_train = lr.predict(X\_train\_trans) y\_pred\_test = lr.predict(X\_test\_trans)

*# get the actual predictions values*

y\_pred\_train\_org = pt.inverse\_transform(y\_pred\_train.reshape(-1,1)) y\_pred\_test\_org = pt.inverse\_transform(y\_pred\_test.reshape(-1,1))

from sklearn.metrics import mean\_absolute\_error, r2\_score

print(f"The train error is

{mean\_absolute\_error(y\_train,y\_pred\_train\_org):.2f} minutes") print(f"The test error is

{mean\_absolute\_error(y\_test,y\_pred\_test\_org):.2f} minutes")

The train error is 4.70 minutes The test error is 4.69 minutes

print(f"The train r2 score is

{r2\_score(y\_train,y\_pred\_train\_org):.2f}")

print(f"The test r2 score is {r2\_score(y\_test,y\_pred\_test\_org):.2f}")

The train r2 score is 0.60 The test r2 score is 0.60

Impute Missing values

temp\_df = df.copy()

*# split into X and y*

X = temp\_df.drop(columns='time\_taken') y = temp\_df['time\_taken']

X

{"summary":"{\n \"name\": \"X\",\n \"rows\": 45502,\n \"fields\": [\n {\n \"column\": \"age\",\n \"properties\": {\n

\"dtype\": \"number\",\n \"std\": 5.761481905786405,\n

\"min\": 20.0,\n \"max\": 39.0,\n \"num\_unique\_values\": 20,\n \"samples\": [\n 37.0,\n 28.0,\n

26.0\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"ratings\",\n \"properties\": {\n \"dtype\": \"number\",\ n \"std\": 0.3138265411494576,\n \"min\": 2.5,\n

\"max\": 5.0,\n \"num\_unique\_values\": 26,\n

\"samples\": [\n 4.0,\n 2.5,\n 4.9\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"weather\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 6,\n \"samples\": [\n

\"sunny\",\n \"stormy\",\n \"windy\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"traffic\",\n \"properties\":

{\n \"dtype\": \"category\",\n \"num\_unique\_values\": 4,\n \"samples\": [\n \"jam\",\n \"medium\",\ n \"high\"\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"vehicle\_condition\",\n \"properties\": {\n \"dtype\":

\"number\",\n \"std\": 0,\n \"min\": 0,\n

\"max\": 3,\n \"num\_unique\_values\": 4,\n \"samples\": [\n 0,\n 3,\n 2\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"type\_of\_order\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"drinks\",\n \"meal\",\n \"snack\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"type\_of\_vehicle\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"scooter\",\n \"bicycle\",\n \"motorcycle\"\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"multiple\_deliveries\",\n

\"properties\": {\n \"dtype\": \"number\",\n \"std\": 0.5724880756019144,\n \"min\": 0.0,\n \"max\": 3.0,\n

\"num\_unique\_values\": 4,\n \"samples\": [\n 1.0,\n 2.0,\n 0.0\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"festival\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 2,\n \"samples\": [\n \"yes\",\n \"no\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"city\_type\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 3,\n \"samples\": [\n

\"urban\",\n \"metropolitian\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"city\_name\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 22,\n \"samples\": [\n

\"INDO\",\n \"KOL\"\n ],\n \"semantic\_type\":

\"\",\n \"description\": \"\"\n }\n },\n {\n

\"column\": \"order\_month\",\n \"properties\": {\n

\"dtype\": \"number\",\n \"std\": 0,\n \"min\": 2,\n

\"max\": 4,\n \"num\_unique\_values\": 3,\n \"samples\": [\n 3,\n 4\n ],\n \"semantic\_type\":

\"\",\n \"description\": \"\"\n }\n },\n {\n

\"column\": \"order\_day\_of\_week\",\n \"properties\": {\n

\"dtype\": \"category\",\n \"num\_unique\_values\": 7,\n

\"samples\": [\n \"saturday\",\n \"friday\"\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"is\_weekend\",\n

\"properties\": {\n \"dtype\": \"number\",\n \"std\": 0,\n \"min\": 0,\n \"max\": 1,\n

\"num\_unique\_values\": 2,\n \"samples\": [\n 0,\n 1\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"pickup\_time\_minutes\",\n \"properties\": {\n \"dtype\":

\"number\",\n \"std\": 4.087515505036095,\n \"min\": 5.0,\n \"max\": 15.0,\n \"num\_unique\_values\": 3,\n

\"samples\": [\n 15.0,\n 5.0\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"order\_time\_of\_day\",\n

age ratings weather traffic

vehicle\_condition type\_of\_order type\_of\_vehicle multiple\_deliveries festival

city\_type city\_name order\_month order\_day\_of\_week is\_weekend

pickup\_time\_minutes order\_time\_of\_day distance

1470

1510

421

407

0

0

0

795

188

968

0

0

0

0

1298

1646

2931

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"evening\",\n \"night\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"distance\",\n \"properties\":

{\n \"dtype\": \"number\",\n \"std\": 5.602890163772506,\n \"min\": 1.4650674052309467,\n

\"max\": 20.969489380087342,\n \"num\_unique\_values\": 4362,\n

\"samples\": [\n 9.19295116708344,\n 11.748969507649402\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"distance\_type\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 4,\n \"samples\": [\n \"very\_long\",\n \"long\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n }\n ]\n}","type":"dataframe","variable\_name":"X"}

*# train test split*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=42)

print("The size of train data is",X\_train.shape) print("The shape of test data is",X\_test.shape)

The size of train data is (36401, 18) The shape of test data is (9101, 18)

*# missing values in train data*

X\_train.isna().sum()

dtype: int64

*# transform target column*

pt = PowerTransformer()

y\_train\_pt = pt.fit\_transform(y\_train.values.reshape(-1,1)) y\_test\_pt = pt.transform(y\_test.values.reshape(-1,1))

missing\_cols

Index(['age', 'ratings', 'weather', 'traffic', 'multiple\_deliveries',

'festival', 'city\_type', 'pickup\_time\_minutes', 'order\_time\_of\_day',

'distance', 'distance\_type'],

dtype='object')

*# percentage of rows in data having missing values*

(

X\_train

.isna()

.any(axis=1)

.mean()

.round(2) \* 100

)

17.0

distance\_type 2931

## Age

X\_train['age'].describe()

Name: age, dtype: float64

*# missing values in the column*

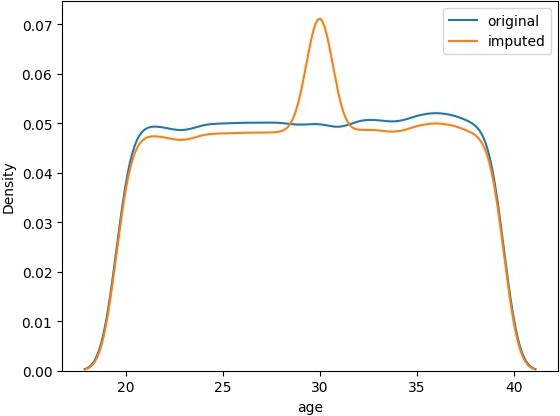
X\_train['age'].isna().sum() 1470

|  |  |
| --- | --- |
| count | 34931.000000 |
| mean | 29.569551 |
| std | 5.752869 |
| min | 20.000000 |
| 25% | 25.000000 |
| 50% | 30.000000 |
| 75% | 35.000000 |
| max | 39.000000 |

*# median value*

age\_median = X\_train['age'].median()

Avg and Median values are similar, impute the age column with median value



*# plot the kde plot*

sns.kdeplot(X\_train['age'],label="original") sns.kdeplot(X\_train['age'].fillna(age\_median),label="imputed") plt.legend()

<matplotlib.legend.Legend at 0x7e4fdf4ed660>

Observation:

1. Changed the distribution of the age column.
2. Use Advanced imputation techniques like KNN imputer.

## Ratings

*# statistical summary*

X\_train['ratings'].describe()

|  |  |  |
| --- | --- | --- |
|  |  |  |
| count | 34891.000000 |
| mean | 4.635058 |
| std | 0.314049 |
| min | 2.500000 |
| 25% | 4.500000 |
| 50% | 4.700000 |
| 75% | 4.900000 |
|  | max | 5.000000 |

Name: ratings, dtype: float64

*# missing values* X\_train['ratings'].isna().sum() 1510

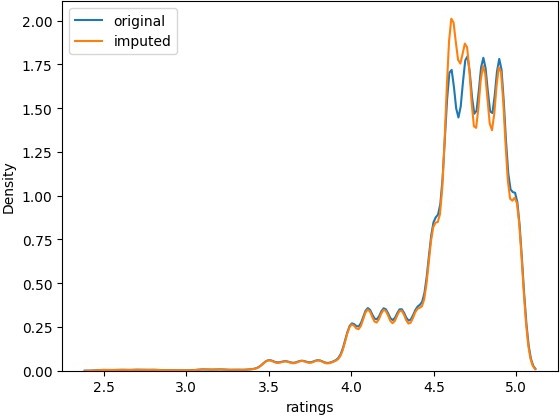
*# avg rating*

ratings\_mean = X\_train['ratings'].mean()

*# fill and plot kdeplot*

sns.kdeplot(X\_train['ratings'],label="original") sns.kdeplot(X\_train['ratings'].fillna(ratings\_mean),label="imputed") plt.legend()

<matplotlib.legend.Legend at 0x7e4fdf4ecf70>



Weather

*# value counts*

X\_train['weather'].value\_counts() weather

Name: count, dtype: int64

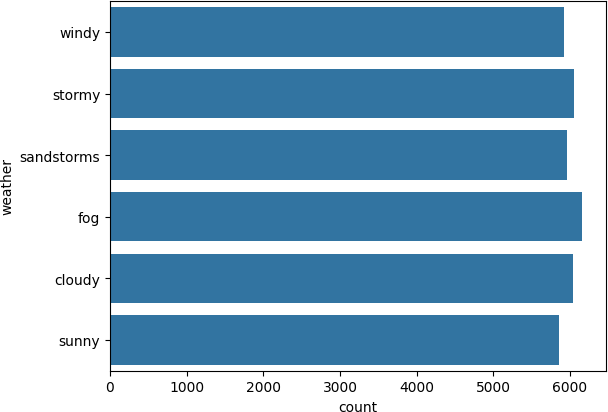
*# missing values in the column* X\_train['weather'].isna().sum() 421

*# countplot*

sns.countplot(X\_train['weather'])

<Axes: xlabel='count', ylabel='weather'>

|  |  |
| --- | --- |
| fog | 6160 |
| stormy | 6051 |
| cloudy | 6033 |
| sandstorms | 5958 |
| windy | 5928 |
| sunny | 5850 |



No dominant category to impute from

*# capture the missingness*

missing\_weather = MissingIndicator() missing\_weather.set\_output(transform="pandas")

pd.concat([X\_train['weather'],missing\_weather.fit\_transform(X\_train[[' weather']])],axis=1).sample(50)

{"summary":"{\n \"name\": \"pd\",\n \"rows\": 50,\n \"fields\": [\n

{\n \"column\": \"weather\",\n \"properties\": {\n

\"dtype\": \"category\",\n

\"samples\": [\n

\"num\_unique\_values\": 6,\n

\"fog\",\n

\"stormy\",\n

\"windy\"\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"missingindicator\_weather\",\n \"properties\": {\n

\"dtype\": \"boolean\",\n \"num\_unique\_values\": 2,\n

\"samples\": [\n true,\n

\"semantic\_type\": \"\",\n

false\n ],\n

\"description\": \"\"\n }\

n }\n ]\n}","type":"dataframe"}

## Traffic

*# value counts*

X\_train['traffic'].value\_counts() traffic

Name: count, dtype: int64

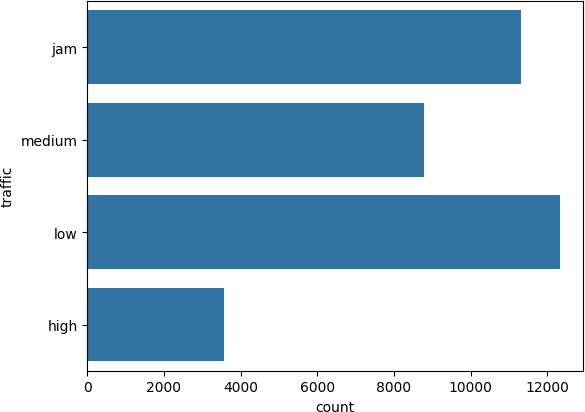
*# Missing values in column* X\_train['traffic'].isna().sum() 407

*# countplot*

sns.countplot(X\_train['traffic'])

<Axes: xlabel='count', ylabel='traffic'>

|  |  |
| --- | --- |
| low | 12323 |
| jam | 11320 |
| medium | 8773 |
| high | 3578 |



No dominant category

missing\_cols

Index(['age', 'ratings', 'weather', 'traffic', 'multiple\_deliveries',

'festival', 'city\_type', 'pickup\_time\_minutes', 'order\_time\_of\_day',

'distance', 'distance\_type'],

dtype='object')

## Multiple Deliveries

*# value counts*

X\_train['multiple\_deliveries'].value\_counts() multiple\_deliveries

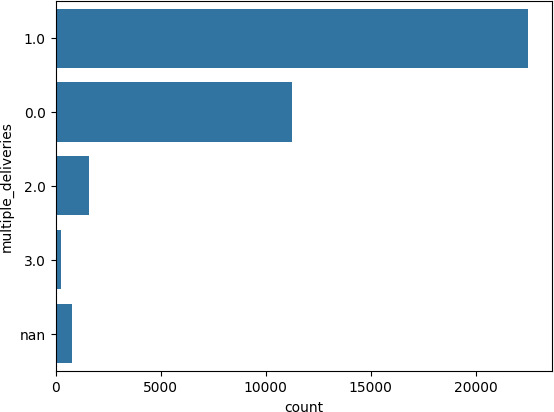
Name: count, dtype: int64

*# countplot*

sns.countplot(X\_train['multiple\_deliveries'].apply(str))

<Axes: xlabel='count', ylabel='multiple\_deliveries'>

|  |  |
| --- | --- |
| 1.0 | 22487 |
| 0.0 | 11252 |
| 2.0 | 1599 |
| 3.0 | 268 |



*# number of missing values* X\_train['multiple\_deliveries'].isna().sum() 795

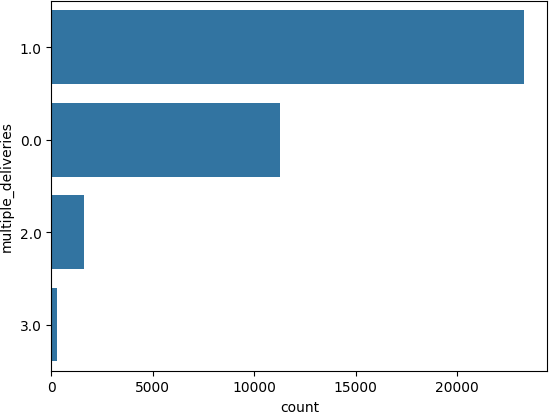
*# mode value*

multiple\_deliveries\_mode = X\_train['multiple\_deliveries'].mode()[0]

*# fill na values with mode*

sns.countplot(X\_train['multiple\_deliveries'].fillna(multiple\_deliverie s\_mode).apply(str))

<Axes: xlabel='count', ylabel='multiple\_deliveries'>



Mode can be used for this column as an imputation technique

Festival

no yes

35474

739

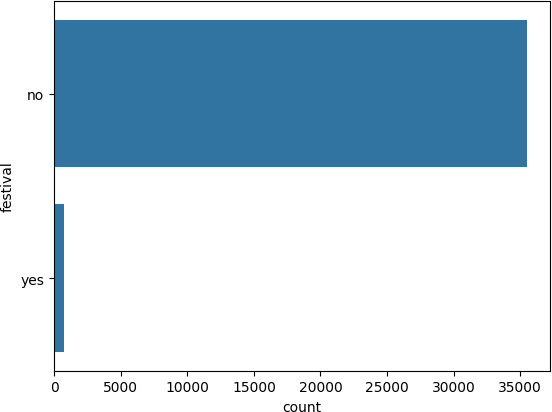
*# value counts* X\_train['festival'].value\_counts() festival

Name: count, dtype: int64

*# countplot*

sns.countplot(X\_train['festival'])

<Axes: xlabel='count', ylabel='festival'>



*# missing values in column* X\_train['festival'].isna().sum() 188

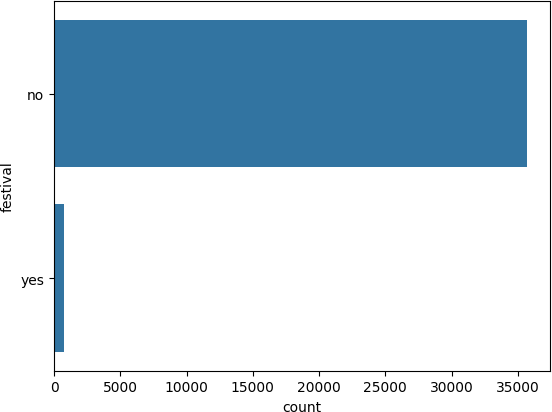
*# mode value*

festival\_mode = X\_train['festival'].mode()[0]

*# fill with mode*

sns.countplot(X\_train['festival'].fillna(festival\_mode))

<Axes: xlabel='count', ylabel='festival'>



City type

metropolitian urban

semi-urban

27245

8058

130

*# value counts* X\_train['city\_type'].value\_counts() city\_type

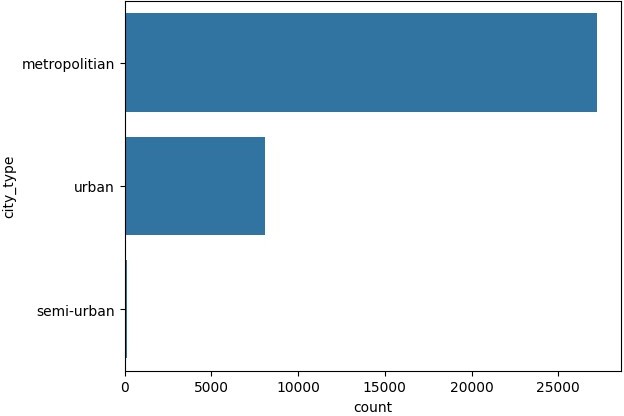
Name: count, dtype: int64

*# number of missing values* X\_train['city\_type'].isna().sum() 968

*# countplot*

sns.countplot(X\_train['city\_type'])

<Axes: xlabel='count', ylabel='city\_type'>



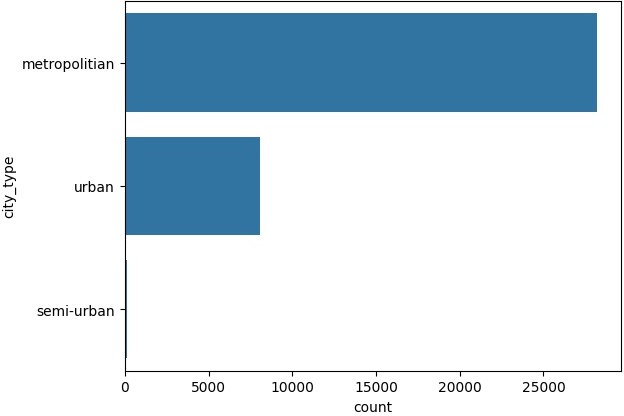
*# mode value*

city\_type\_mode = X\_train['city\_type'].mode()[0]

*# fill with mode*

sns.countplot(X\_train['city\_type'].fillna(city\_type\_mode))

<Axes: xlabel='count', ylabel='city\_type'>



*# statistical summary*

X\_train['pickup\_time\_minutes'].describe()

Name: pickup\_time\_minutes, dtype: float64

missing\_cols

Index(['age', 'ratings', 'weather', 'traffic', 'multiple\_deliveries',

'festival', 'city\_type', 'pickup\_time\_minutes', 'order\_time\_of\_day',

'distance', 'distance\_type'],

dtype='object')

## Pickup time minutes

|  |  |
| --- | --- |
| count | 35103.000000 |
| mean | 9.998718 |
| std | 4.082279 |
| min | 5.000000 |
| 25% | 5.000000 |
| 50% | 10.000000 |
| 75% | 15.000000 |
| max | 15.000000 |

*# missing values in the column* X\_train['pickup\_time\_minutes'].isna().sum() 1298

*# median value*

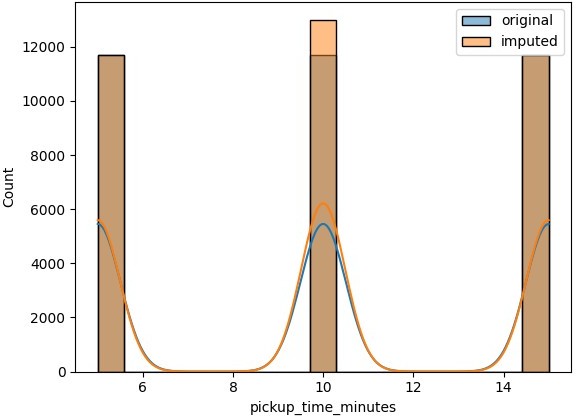
pickup\_time\_minutes\_median = X\_train['pickup\_time\_minutes'].median()

*# histplot*

sns.histplot(X\_train['pickup\_time\_minutes'],kde=True,label='original') sns.histplot(X\_train['pickup\_time\_minutes'].fillna(pickup\_time\_minutes

\_median),kde=True,label='imputed') plt.legend()

<matplotlib.legend.Legend at 0x7e50250e74c0>



Order time of day

*# value counts*

X\_train['order\_time\_of\_day'].value\_counts()

order\_time\_of\_day

Name: count, dtype: int64

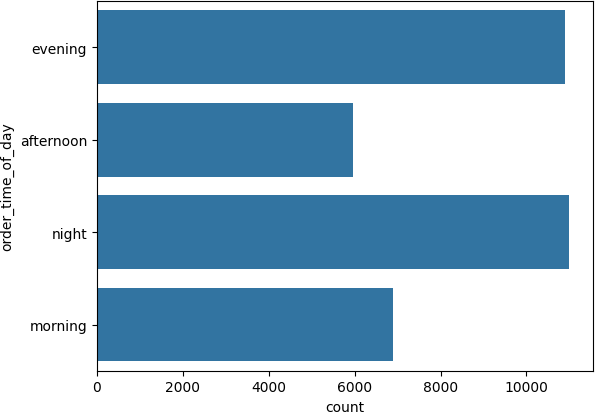
*# missing values* X\_train['order\_time\_of\_day'].isna().sum() 1646

*# countplot*

sns.countplot(X\_train['order\_time\_of\_day'])

<Axes: xlabel='count', ylabel='order\_time\_of\_day'>

|  |  |
| --- | --- |
| night | 10994 |
| evening | 10906 |
| morning | 6883 |
| afternoon | 5972 |



*# rows where the data is missing*

X\_train[X\_train['order\_time\_of\_day'].isna()]

{"repr\_error":"0","type":"dataframe"}

## Distance

*# statistical summary*

X\_train['distance'].describe()

Name: distance, dtype: float64

*# number of missing values* X\_train['distance'].isna().sum() 2931

*# avg distance*

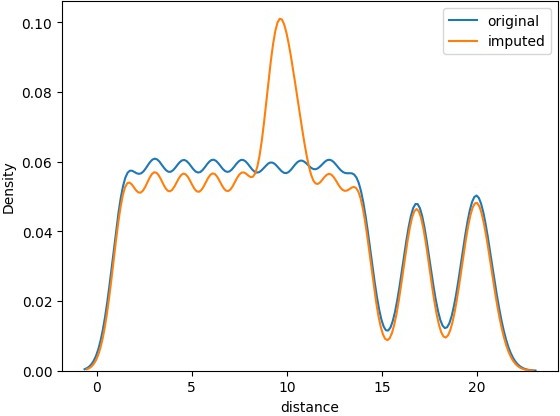
distance\_mean = X\_train['distance'].mean()

*# kdeplot*

sns.kdeplot(X\_train['distance'],label='original') sns.kdeplot(X\_train['distance'].fillna(distance\_mean),label='imputed') plt.legend()

<matplotlib.legend.Legend at 0x7e4fdeae4cd0>

|  |  |
| --- | --- |
| count | 33470.000000 |
| mean | 9.738154 |
| std | 5.608401 |
| min | 1.465067 |
| 25% | 4.657672 |
| 50% | 9.193421 |
| 75% | 13.681057 |
| max | 20.969489 |



*# value counts*

X\_train['distance\_type'].value\_counts() distance\_type

Name: count, dtype: int64

*# missing values*

X\_train['distance\_type'].isna().sum()

missing\_cols

Index(['age', 'ratings', 'weather', 'traffic', 'multiple\_deliveries',

'festival', 'city\_type', 'pickup\_time\_minutes', 'order\_time\_of\_day',

'distance', 'distance\_type'],

dtype='object')

Distance Type

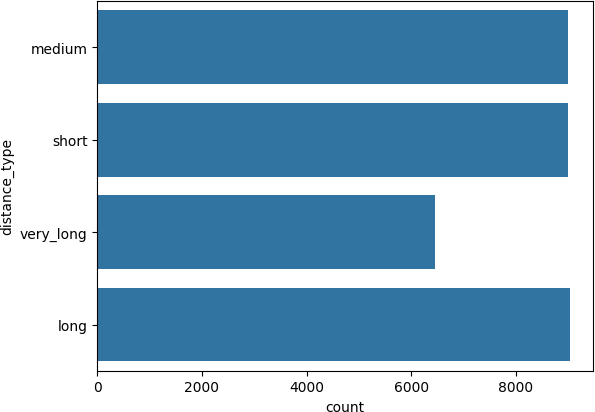
|  |  |
| --- | --- |
| long | 9025 |
| short | 8998 |
| medium | 8993 |
| very\_long | 6454 |

2931

*# countplot*

sns.countplot(X\_train['distance\_type'])

<Axes: xlabel='count', ylabel='distance\_type'>



Mode cannot be used here

Imputation Pipeline

nominal\_cat\_cols

['weather',

'type\_of\_order',

'type\_of\_vehicle',

'festival',

'city\_type',

'city\_name',

'order\_month',

'order\_day\_of\_week',

'is\_weekend',

age ratings weather traffic

vehicle\_condition type\_of\_order type\_of\_vehicle multiple\_deliveries festival

city\_type city\_name order\_month order\_day\_of\_week is\_weekend

pickup\_time\_minutes order\_time\_of\_day distance distance\_type dtype: int64

1470

1510

421

407

0

0

0

795

188

968

0

0

0

0

1298

1646

2931

2931

'order\_time\_of\_day'] X\_train.isna().sum()

*# features to fill values with mode*

features\_to\_fill\_mode = ['multiple\_deliveries','festival','city\_type'] features\_to\_fill\_missing = [col for col in nominal\_cat\_cols if col not in features\_to\_fill\_mode]

features\_to\_fill\_missing ['weather',

'type\_of\_order',

'type\_of\_vehicle',

'city\_name',

'order\_month',

'order\_day\_of\_week',

'is\_weekend',

'order\_time\_of\_day']

*# simple imputer to fill categorical vars with mode*

simple\_imputer = ColumnTransformer(transformers=[

("mode\_imputer",SimpleImputer(strategy="most\_frequent"),features\_to\_fi ll\_mode),

("missing\_imputer",SimpleImputer(strategy="constant",fill\_value="missi ng"),features\_to\_fill\_missing)

],remainder="passthrough",n\_jobs=- 1,force\_int\_remainder\_cols=False,verbose\_feature\_names\_out=False)

simple\_imputer ColumnTransformer(force\_int\_remainder\_cols=False, n\_jobs=-1,

remainder='passthrough',

transformers=[('mode\_imputer',

SimpleImputer(strategy='most\_frequent'),

['multiple\_deliveries', 'festival', 'city\_type']),

('missing\_imputer', SimpleImputer(fill\_value='missing',

strategy='constant'), ['weather', 'type\_of\_order',

'type\_of\_vehicle',

'city\_name', 'order\_month', 'order\_day\_of\_week', 'is\_weekend', 'order\_time\_of\_day'])],

verbose\_feature\_names\_out=False) simple\_imputer.fit\_transform(X\_train)

{"summary":"{\n \"name\": \"simple\_imputer\",\n \"rows\": 36401,\n

\"fields\": [\n {\n \"column\": \"multiple\_deliveries\",\n

\"properties\": {\n \"dtype\": \"date\",\n \"min\": 0.0,\n \"max\": 3.0,\n \"num\_unique\_values\": 4,\n

\"samples\": [\n 0.0,\n 3.0,\n 1.0\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"festival\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 2,\n \"samples\": [\n \"yes\",\ n \"no\"\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"city\_type\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 3,\n \"samples\": [\n \"metropolitian\",\n \"urban\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"weather\",\n \"properties\":

{\n \"dtype\": \"category\",\n \"num\_unique\_values\": 7,\n \"samples\": [\n \"windy\",\n

\"stormy\"\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"type\_of\_order\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 4,\n \"samples\": [\n \"drinks\",\n \"buffet\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"type\_of\_vehicle\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"motorcycle\",\n \"bicycle\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"city\_name\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 22,\n \"samples\": [\n

\"MUM\",\n \"GOA\"\n ],\n \"semantic\_type\":

\"\",\n \"description\": \"\"\n }\n },\n {\n

\"column\": \"order\_month\",\n \"properties\": {\n

\"dtype\": \"date\",\n \"min\": 2,\n \"max\": 4,\n

\"num\_unique\_values\": 3,\n \"samples\": [\n 3,\n 4\n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"order\_day\_of\_week\",\n \"properties\": {\n \"dtype\":

\"category\",\n \"num\_unique\_values\": 7,\n \"samples\": [\n \"wednesday\",\n \"thursday\"\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"is\_weekend\",\n

\"properties\": {\n \"dtype\": \"date\",\n \"min\": 0,\n

\"max\": 1,\n \"num\_unique\_values\": 2,\n \"samples\": [\n 1,\n 0\n ],\n \"semantic\_type\":

\"\",\n \"description\": \"\"\n }\n },\n {\n

\"column\": \"order\_time\_of\_day\",\n \"properties\": {\n

\"dtype\": \"category\",\n \"num\_unique\_values\": 5,\n

\"samples\": [\n \"afternoon\",\n \"morning\"\n

],\n \"semantic\_type\": \"\",\n \"description\": \"\"\n

}\n },\n {\n \"column\": \"age\",\n \"properties\": {\ n \"dtype\": \"number\",\n \"std\": 5.752868749688641,\n

\"min\": 20.0,\n \"max\": 39.0,\n \"num\_unique\_values\": 20,\n \"samples\": [\n 26.0,\n 35.0\

n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"ratings\",\n \"properties\": {\n \"dtype\": \"number\",\ n \"std\": 0.3140494198060681,\n \"min\": 2.5,\n

\"max\": 5.0,\n \"num\_unique\_values\": 26,\n

\"samples\": [\n 4.6,\n 2.6\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\

n },\n {\n \"column\": \"traffic\",\n \"properties\":

{\n \"dtype\": \"category\",\n \"num\_unique\_values\": 4,\n \"samples\": [\n \"medium\",\n \"high\"\ n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"vehicle\_condition\",\n \"properties\": {\n \"dtype\":

\"number\",\n \"std\": 0,\n \"min\": 0,\n

\"max\": 3,\n \"num\_unique\_values\": 4,\n \"samples\": [\n 0,\n 3\n ],\n \"semantic\_type\":

\"\",\n \"description\": \"\"\n }\n },\n {\n

\"column\": \"pickup\_time\_minutes\",\n \"properties\": {\n

\"dtype\": \"number\",\n \"std\": 4.082279167279904,\n

\"min\": 5.0,\n \"max\": 15.0,\n \"num\_unique\_values\": 3,\n \"samples\": [\n 10.0,\n 5.0\

multiple\_deliveries festival

city\_type weather type\_of\_order type\_of\_vehicle city\_name order\_month

order\_day\_of\_week is\_weekend order\_time\_of\_day age

ratings traffic

vehicle\_condition pickup\_time\_minutes distance distance\_type

0

0

0

0

0

0

0

0

0

0

0

1470

1510

407

0

1298

2931

2931

n ],\n \"semantic\_type\": \"\",\n

\"description\": \"\"\n }\n },\n {\n \"column\":

\"distance\",\n \"properties\": {\n \"dtype\":

\"number\",\n \"std\": 5.608400885706788,\n \"min\": 1.4650674052309467,\n \"max\": 20.969489380087342,\n

\"num\_unique\_values\": 4327,\n \"samples\": [\n 8.988056602876522,\n 7.448379823536196\n ],\n

\"semantic\_type\": \"\",\n \"description\": \"\"\n }\ n },\n {\n \"column\": \"distance\_type\",\n

\"properties\": {\n \"dtype\": \"category\",\n

\"num\_unique\_values\": 4,\n \"samples\": [\n

\"short\",\n \"long\"\n ],\n \"semantic\_type\":

\"\",\n \"description\": \"\"\n }\n }\n ]\ n}","type":"dataframe"}

simple\_imputer.fit\_transform(X\_train).isna().sum()

dtype: int64

*# knn imputer*

knn\_imputer = KNNImputer(n\_neighbors=5)

*# do basic preprocessing*

num\_cols = ["age","ratings","pickup\_time\_minutes","distance"]

nominal\_cat\_cols = ['weather','type\_of\_order',

'type\_of\_vehicle',"festival", "city\_type","city\_name","order\_month",

"order\_day\_of\_week", "is\_weekend", "order\_time\_of\_day"]

ordinal\_cat\_cols = ["traffic","distance\_type"] *# generate order for ordinal encoding* traffic\_order = ["low","medium","high","jam"]

distance\_type\_order = ["short","medium","long","very\_long"]

*# unique categories the ordinal columns*

for col in ordinal\_cat\_cols: print(col,X\_train[col].unique())

traffic ['jam' 'medium' 'low' 'high' nan] distance\_type ['medium' 'short' 'very\_long' 'long' nan]

*# build a preprocessor*

preprocessor = ColumnTransformer(transformers=[ ("scale", MinMaxScaler(), num\_cols), ("nominal\_encode",

OneHotEncoder(drop="first",handle\_unknown="ignore",

sparse\_output=False),

nominal\_cat\_cols), ("ordinal\_encode",

OrdinalEncoder(categories=[traffic\_order,distance\_type\_order],

encoded\_missing\_value=-999,

handle\_unknown="use\_encoded\_value",

ordinal\_cat\_cols)

],remainder="passthrough",n\_jobs=-

unknown\_value=-1),

1,force\_int\_remainder\_cols=False,verbose\_feature\_names\_out=False)

preprocessor ColumnTransformer(force\_int\_remainder\_cols=False, n\_jobs=-1,

remainder='passthrough',

transformers=[('scale', MinMaxScaler(),

['age', 'ratings', 'pickup\_time\_minutes',

'distance']),

('nominal\_encode',

OneHotEncoder(drop='first', handle\_unknown='ignore',

sparse\_output=False), ['weather', 'type\_of\_order',

'type\_of\_vehicle',

'festival', 'city\_type',

'city\_name',

'order\_month', 'order\_day\_of\_week', 'is\_weekend', 'order\_time\_of\_day']),

('ordinal\_encode', OrdinalEncoder(categories=[['low',

'medium',

'high',

'jam'],

['short',

'medium',

'long',

'very\_long']], encoded\_missing\_value=-999,

handle\_unknown='use\_encoded\_value',

unknown\_value=-1),

['traffic', 'distance\_type'])],

verbose\_feature\_names\_out=False) preprocessor.fit\_transform(X\_train)

{"type":"dataframe"}

preprocessor.fit\_transform(X\_train).isna().sum().loc[lambda ser : ser.ge(1)]

[age 1470](#_TOC_250003)

[ratings 1510](#_TOC_250002)

[pickup\_time\_minutes 1298](#_TOC_250001)

[distance 2931](#_TOC_250000)

multiple\_deliveries 795

dtype: int64

*# build the pipeline*

processing\_pipeline = Pipeline(steps=[

("simple\_imputer",simple\_imputer), ("preprocess",preprocessor), ("knn\_imputer",knn\_imputer)

])

processing\_pipeline

Pipeline(steps=[('simple\_imputer',

ColumnTransformer(force\_int\_remainder\_cols=False,

n\_jobs=-1,

remainder='passthrough',

transformers=[('mode\_imputer', SimpleImputer(strategy='most\_frequent'),

['multiple\_deliveries',

'festival',

'city\_type']),

('missing\_imputer',

SimpleImputer(fill\_value='missing',

strategy='constant'),

['weather',

'type\_of\_order',

'type\_of\_vehi... 'city\_name',

'order\_month',

'order\_day\_of\_week', 'order\_time\_of\_day']),

OrdinalEncoder(categories=[['low', 'medium',

'high',

'jam'],

['short',

'medium',

'long', 'very\_long']],

encoded\_missing\_value=-999,

handle\_unknown='use\_encoded\_value', unknown\_value=-1),

'is\_weekend', ('ordinal\_encode',

['traffic', 'distance\_type'])],

verbose\_feature\_names\_out=False)),

('knn\_imputer', KNNImputer())]) *# fit and transform the pipeline on X\_train* processing\_pipeline.fit\_transform(X\_train)

{"type":"dataframe"}

from sklearn.linear\_model import LinearRegression lr = LinearRegression()

model\_pipe = Pipeline(steps=[

model\_pipe

("preprocessing",processing\_pipeline), ("model",lr)

])

Pipeline(steps=[('preprocessing',

Pipeline(steps=[('simple\_imputer', ColumnTransformer(force\_int\_remainder\_cols=False,

remainder='passthrough', transformers=[('mode\_imputer', SimpleImputer(strategy='most\_frequent'), ['multiple\_deliveries',

'festival', 'city\_type']), ('missing\_imputer',

SimpleImputer(fill\_value='missing', strategy='constant'),

['weath...

'is\_weekend', 'order\_time\_of\_day']), ('ordinal\_encode',

n\_jobs=-1,

OrdinalEncoder(categories=[['low', 'medium',

'high',

'jam'],

['short',

'medium',

'long', 'very\_long']],

encoded\_missing\_value=-999, handle\_unknown='use\_encoded\_value', unknown\_value=-1),

['traffic',

'distance\_type'])], verbose\_feature\_names\_out=False)),

('knn\_imputer', KNNImputer())])),

('model', LinearRegression())]) *# fit the pipeline on data* model\_pipe.fit(X\_train,y\_train\_pt)

Pipeline(steps=[('preprocessing',

Pipeline(steps=[('simple\_imputer',

ColumnTransformer(force\_int\_remainder\_cols=False,

n\_jobs=-1,

remainder='passthrough', transformers=[('mode\_imputer', SimpleImputer(strategy='most\_frequent'), ['multiple\_deliveries',

'festival', 'city\_type']),

('missing\_imputer', SimpleImputer(fill\_value='missing', strategy='constant'),

['weath... 'is\_weekend', 'order\_time\_of\_day']), ('ordinal\_encode',

OrdinalEncoder(categories=[['low', 'medium',

'high',

'jam'],

['short',

'medium',

'long', 'very\_long']],

encoded\_missing\_value=-999, handle\_unknown='use\_encoded\_value', unknown\_value=-1),

['traffic',

'distance\_type'])], verbose\_feature\_names\_out=False)),

('knn\_imputer', KNNImputer())])),

('model', LinearRegression())])

*# get the predictions*

y\_pred\_train = model\_pipe.predict(X\_train) y\_pred\_test = model\_pipe.predict(X\_test)

*# get the actual predictions values*

y\_pred\_train\_org = pt.inverse\_transform(y\_pred\_train.reshape(-1,1)) y\_pred\_test\_org = pt.inverse\_transform(y\_pred\_test.reshape(-1,1))

from sklearn.metrics import mean\_absolute\_error, r2\_score

print(f"The train error is

{mean\_absolute\_error(y\_train,y\_pred\_train\_org):.2f} minutes") print(f"The test error is

{mean\_absolute\_error(y\_test,y\_pred\_test\_org):.2f} minutes")

The train error is 4.83 minutes The test error is 4.86 minutes

print(f"The train r2 score is

{r2\_score(y\_train,y\_pred\_train\_org):.2f}")

print(f"The test r2 score is {r2\_score(y\_test,y\_pred\_test\_org):.2f}")

The train r2 score is 0.58 The test r2 score is 0.58

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(random\_state=42,n\_jobs=-1) model\_pipe = Pipeline(steps=[

model\_pipe

("preprocessing",processing\_pipeline), ("model",rf)

])

Pipeline(steps=[('preprocessing',

Pipeline(steps=[('simple\_imputer', ColumnTransformer(force\_int\_remainder\_cols=False,

remainder='passthrough', transformers=[('mode\_imputer', SimpleImputer(strategy='most\_frequent'), ['multiple\_deliveries',

'festival', 'city\_type']), ('missing\_imputer',

SimpleImputer(fill\_value='missing',

n\_jobs=-1,

strategy='constant'), ['weath... ('ordinal\_encode',

OrdinalEncoder(categories=[['low', 'medium',

'high',

'jam'],

['short',

'medium',

'long', 'very\_long']],

encoded\_missing\_value=-999, handle\_unknown='use\_encoded\_value', unknown\_value=-1),

['traffic',

'distance\_type'])], verbose\_feature\_names\_out=False)),

('knn\_imputer', KNNImputer())])),

('model', RandomForestRegressor(n\_jobs=-1, random\_state=42))])

*# fit the pipeline on data*

model\_pipe.fit(X\_train,y\_train\_pt.values.ravel()) Pipeline(steps=[('preprocessing',

Pipeline(steps=[('simple\_imputer',

ColumnTransformer(force\_int\_remainder\_cols=False,

n\_jobs=-1,

remainder='passthrough', transformers=[('mode\_imputer',

SimpleImputer(strategy='most\_frequent'), ['multiple\_deliveries',

'festival', 'city\_type']), ('missing\_imputer',

SimpleImputer(fill\_value='missing', strategy='constant'),

['weath...

('ordinal\_encode', OrdinalEncoder(categories=[['low', 'medium',

'high',

'jam'],

['short',

'medium',

'long', 'very\_long']],

encoded\_missing\_value=-999, handle\_unknown='use\_encoded\_value', unknown\_value=-1),

['traffic',

'distance\_type'])], verbose\_feature\_names\_out=False)),

('knn\_imputer', KNNImputer())])),

('model',

RandomForestRegressor(n\_estimators=50, n\_jobs=-1,

random\_state=42))])

*# get the predictions*

y\_pred\_train = model\_pipe.predict(X\_train) y\_pred\_test = model\_pipe.predict(X\_test)

*# get the actual predictions values*

y\_pred\_train\_org = pt.inverse\_transform(y\_pred\_train.reshape(-1,1)) y\_pred\_test\_org = pt.inverse\_transform(y\_pred\_test.reshape(-1,1))

from sklearn.metrics import mean\_absolute\_error, r2\_score

print(f"The train error is

{mean\_absolute\_error(y\_train,y\_pred\_train\_org):.2f} minutes") print(f"The test error is

{mean\_absolute\_error(y\_test,y\_pred\_test\_org):.2f} minutes")

The train error is 1.22 minutes The test error is 3.28 minutes

print(f"The train r2 score is

{r2\_score(y\_train,y\_pred\_train\_org):.2f}")

print(f"The test r2 score is {r2\_score(y\_test,y\_pred\_test\_org):.2f}")

The train r2 score is 0.97 The test r2 score is 0.80