VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

"Delivery time prediction using Machine Learning Techniques"

Reg No:	**Name:**
20BIT0095	Routhu sai praneeth

Winter semester 2022-2023

Machine Learning

School of Information Technology and Engineering

Winter Semester 2022-23

Under the Guidance of:

Prof: Durai Raj Vincent

Introduction:

Delivery time prediction is an important task in the field of logistics and transportation, as it helps businesses to optimize their operations and provide better services to their customers. With the advancement of machine learning techniques, it has become possible to predict delivery times more accurately by analyzing various factors that affect the delivery process.

Machine learning techniques like regression, decision trees, and neural networks can be used to build predictive models that take into account parameters such as distance, traffic conditions, weather, and past delivery data. These models can learn patterns and relationships from historical data and use them to make predictions about future deliveries.

In this project, we will explore how machine learning techniques can be used to predict delivery times for a given set of parameters. We will use a dataset of past delivery records to train and test various machine learning models and evaluate their performance. The goal of this project is to build an accurate and robust delivery time prediction model that can help businesses optimize their logistics operations and improve customer satisfaction.

Abstract:

Delivery time prediction is a challenging task for food delivery services like Zomato and Swiggy. To accurately predict delivery time, these companies rely on calculating the distance between the point of order pick-up and delivery, as well as analyzing past delivery times for the same distance. Food delivery involves the transportation of items like entrees, sides, drinks, and groceries from a restaurant or store to a customer's location. Accurate delivery time prediction is crucial for maintaining transparency and customer satisfaction. Machine learning algorithms are commonly used to predict delivery times based on historical data. This project mainly focuses on explores how to use machine learning for food delivery time prediction using Python, including finding relationships between past delivery times and distance, and utilizing relevant datasets to improve accuracy. The dataset is taken from kaggle and the various algorithms will be applied. Our results show that machine learning models can be used to accurately predict delivery times, with neural networks performing the best in terms of prediction accuracy. We also found that including additional factors such as weather conditions and traffic data can significantly improve the performance of the models.

Literature Survey:

1. **Unboxing Deep Learning Model of Food Delivery Service Reviews Using Explainable Artificial Intelligence (XAI) Technique.**
   - **Anirban Adak, Biswajeet Pradhan, Nagesh Shukla, and Abdullah Alamri.**

Abstract:

Customers who would rather purchase meals online and have them delivered to their door than wait in line at a restaurant have helped to drive the demand for food delivery services (FDSs) amid the COVID-19 crisis. Customer reviews on internet platforms have developed into an important source of data regarding a company's performance as more restaurants have gone online and joined FDSs like Uber Eats, Menulog, and Deliveroo. FDS organisations work hard to compile customer complaints and use the data wisely to determine what needs to be improved to raise customer happiness. Due to the volume of consumer feedback data and dearth of customer service experts, only a small number of client opinions are addressed. Organizations can employ artificial intelligence (AI) to solve problems rather of depending on customer service specialists.

Algorithms:

Deep Learning techniques (LSTM, Bi-LSTM, Bi-GRU-LSTM-CNN), Shapley Additive exPlanations (SHAP), Local Interpretable Model-Agnostic Explanations (LIME).

Conclusion:

The goal of this study was to forecast the FDS industry's customer evaluations' tone and to explain the choices. The use of FDS will continue to rise even after the COVID-19 outbreak has subsided. AI can enable FDS organisations to solve problems and save money given the volume of review data scattered across various platforms and the absence of customer service people to assess and reply to each comment. A false positive result in the FDS domain denotes increased operational efforts, whereas a false negative result raises the possibility of an organisation missing significant customer complaints.

2. **A Procedure for Tracing Supply Chains for Perishable Food Based on Blockchain, Machine Learning and Fuzzy Logic.**
   - **Zeinab Shahbazi and Yung-Cheol Byun.**

Abstract:

Improving the food traceability system is one of the most important aspects of food production in the industry and of product shelf life. In order to raise the standard of the anti-

counterfeiting sector, the food traceability mechanism has emerged as one of the new blockchain applications. Several systems used in the food processing industry have poor readability, scalability, and data accuracy. Similar to that process, this one is difficult to complete and takes a long time to process. To address these problems, the blockchain system develops a new ontology in the supply chain traceability system. In order to merge the new blockchain extension, Machine Learning technology (ML), and fuzzy logic, a blockchain machine learning-based food traceability system (BMLFTS) is proposed in this study.

Algorithms:

Block chain, fuzzy logic, machine learning algorithms.

Conclusion:

One of the crucial and significant steps in the PFSC is food traceability, which displays data on the food's entire life cycle. In order to make the best choice, customers and supply chain participants also adhere to this process. Based on the rapid growth of the e-commerce industry, purchasing perishable foods online has become more popular than visiting wet markets. The lack of information about the goods gives the client confidence that the item is fresh. From a different angle, the perishable food industry is delicate and important because of food variation and deterioration.

3. Prototyping Machine-Learning-Supported Lead Time prediction Using AutoML.

- **Janek Bender, Jivka Ovtcharova.**

Abstract:

Many small and medium-sized businesses that specialise in make-to-order and small-series production find it difficult to forecast the lead times for orders that can be highly customised. In order to allow Lead Time Prediction based on Machine Learning models, this article explores a method employing AutoML integrated into current corporate systems. This forecast is based on real-time factory state data provided by an IIoT platform as well as order data from an ERP system. We created a lightweight web-based microservice around the AutoML model generated from simulation data to estimate lead times of incoming orders during live production. This microservice can be easily integrated into current system landscapes using industry standards.

Algorithms:

Lead Time Prediction, AutoML, Machine Learning Algorithms

Conclusion:

In this research, we demonstrated a straightforward method for incorporating AutoML-supported ML-supported LTP into a larger business architecture in a manufacturing SME. Using H20.ai to infer LTs on fresh orders, a microservice architecture was created to imitate the order scheduling functionality of a MES. Using H20.ai's AutoML features, the corresponding models were trained using simulation data. The idea of ML-supported LTP is technically viable and advantageous for order scheduling in SMEs, we preliminarily conclude.

4. Boosting Algorithms for Delivery Time Prediction in Transportation Logistics

- **Jihed Khiari, Cristina Olaverri-Monreal.**

Abstract:

A key factor in transportation is travel time. For operational and sophisticated information systems to function, accurate travel time prediction is also essential. There are many ways to anticipate short-term trip times, including solutions that follow a vehicle's journey using real-time GPS data and optimization techniques. Long-term projections that can be trusted, however, are still difficult. In this study, we demonstrate the usefulness and applicability of journey time, or delivery time prediction, for postal services. By performing comprehensive tests and taking into account a variety of usability scenarios, we study a number of approaches, including linear regression models and tree-based ensembles, such as random forest, bagging, and boosting, that enable delivery time prediction. The findings show that travel time prediction can reduce significant postal service delays.

Algorithms:

Transportation logistics, Ensemble Learning, Boosting

Conclusion:

The outcomes support the possibility of forecasting postal service journey times. The thorough results demonstrate that by ensuring that mistakes do not exceed one hour in terms of MAE and three hours in terms of RMSE, it is possible to reduce significant delivery delays, such as delays of more than five hours. We point out that anticipating the delay may result in relatively small errors, and that even retraining on a limited basis, such as once a month, may result in small errors and be sufficient to enhance operations and customer information.

5. **Online lead time prediction supporting situation-aware production control**

   - **David Gyulai, Andras Pfeiffer, Julia Bergmann, Viola Gallina**

Abstract:

Recent industrial environments have seen the emergence of Cyber-Physical Production Systems (CPPS), which are able to provide in-depth information about the resources, processes, and products in almost real-time. There are several analytics techniques available to use this technology-related data in decision-making, however these tools usually operate in the maintenance and quality fields. Only a few strategies focus on production control, although from the perspective of overall performance, the success of linked processes is vital. The study introduces a new production data analytics tool that uses machine learning to estimate manufacturing lead times proactively so that decisions may be made by creating a closed-loop production control. Regression approaches are used in the suggested method, and as a result, it facilitates the use of work prioritisation in dispatching decisions.

Algorithms:

ML Regression approaches.

Conclusion:

The experiments produced some extremely encouraging outcomes, but the authors only see this work as a first step towards a comprehensive, real-time, and situation-aware production control framework. The tests show that there is potential for ML methods to be used in PPC since they can respond to dynamic changes in the production environment while still being able to make precise predictions that can be immediately used in control-related decision-making processes. Candidate test environments for the method's implementation in a real industrial setting come from the field of mass customization.

6. **Application of Random Forest Algorithm for the prediction of Online Food Delivery Service Delay**

   - **H.Sahin, D.Icen.**

Abstract:

With the advancement of technology, the online shopping sector has been expanding quickly recently. With the growth of the internet retail industry, consumers have begun to shop based on certain criteria. The online food industry is one of the areas that informs potential customers about service quality by collecting feedback from purchases (comments or ratings). One of the key requirements in the online food sector is fast delivery, and in this study a

classification analysis is done to look into how well that standard is being followed. Application of the Random Forest (RF) method is used for categorization.

Algorithms:

Random Forest Algorithm.

Conclusion:

Customers who place an online meal order can promptly learn about the performance of the restaurants thanks to the outcomes of this application. Since various significant characteristics are taken into account in this study to forecast the "late" or "early" or "on time" arrival of online food delivery using the RF algorithm. These variables include the distance between the customer and the restaurant, the current traffic conditions at that location, the cost of delivery, the time of day, and the volume of customer comments that have been made to the restaurant.

7. Benchmarking AutoML-Supported Lead Time Prediction

- **Janek Bender, Martin Trat, Jivka Ovtcharova**

Abstract:

Predicting precise lead times for highly customizable items poses a challenge for manufacturing organisations, particularly for small and medium enterprises (SME) in the make-to-order arena. Creating data-driven models using machine learning is one approach to solving this problem. With the introduction of Automatic Machine Learning (AutoML), it is now possible for SMEs to employ machine learning independently of data scientists. In this study, we compared the models of three AutoML solutions to manual predictions already in use in the workplace and to basic mean models using data from two SMEs. We also looked into how these solutions aid in the creation of models.

Algorithms:

Automated Machine Learning algorithms, SVM, ANN

Conclusion:

In this study, we train and assess automated machine learning models for lead time prediction using enterprise resource planning (ERP) data from two real-world organisations A and B in the make-to-order (MTO) and small-series manufacturing domains (AutoML). Three AutoML approaches have been compared against the current manual forecasts as well as a straightforward mean model that outputs mean values grouped by production phase (step id).

For Company A, the benchmarked AutoML solutions overall produced better models. Unfortunately, the models had rather large error rates on the test data for some industrial processes, like as sawing or preturning..

8. A REVIEW OF TRAVEL-TIME PREDICTION IN TRANSPORT AND LOGISTICS.

-Hong-En LIN, Rocco ZITO

Abstract:

Information on travel times could be used for a variety of things. From the perspective of the traveller, the travel-time information aids in reducing travel time and increasing dependability through the selection of travel routes before to departure and while on the road. With regard to the use of Delivery costs, delivery reliability, and service quality could all be improved by using logistics and travel-time data. According to a review of the literature, travel-time can be predicted using a variety of alternative techniques and a range of input data. The area of travel-time prediction was reviewed and comparisons made systematically in this work.

Algorithms:

Artificial neural networks

Fuzzy Logic

Conclusion:

The provision of predicted travel-time information offers road users the ability to plan their travel schedules both before embarking on a trip and while en-route. This has the potential to reduce transport operation costs and mitigate environmental impacts. Accurate travel time information also enables delivery industries to enhance their service quality by ensuring timely deliveries. However, the development of travel time estimation and prediction has been hindered by limited traffic data sets and substantial interference from transport environments.

9. Delivery Time Prediction Using Support Vector Machine Combined with Look-back Approach.

- Oğuz Erdi Erkmen1 , Ece Nigiz1 , Z. Sude Sarı 2 , H. Şebnem Arlı 2 , M. Fatih Akay2

Abstract:

The time between order creation and product delivery to the final consumer is known as delivery time. Predicting delivery time is crucial to ensure customer satisfaction and efficient logistics processes. To achieve this goal, the study aimed to utilize Support Vector Machine (SVM) models with and without consecutive look-back and periodic look-back approaches to predict

delivery time. The sample dataset used was obtained from Kaggle. The prediction model's performance was evaluated using Mean Absolute Error (MAE). The results revealed that the look-back approach produced an average MAE, which was 59.12% lower than the model's performance without the look-back approach.

Algorithms:

Support vector machine

Look Back approach

Conclusion:

This study focused on the development of delivery time prediction models using the with and without look-back approach. A dataset sourced from Kaggle was utilized for generating the predictions with the aid of SVM. Initially, the models were generated without the look-back approach, and then consecutive and periodic look-back approaches were applied to enhance the models' performance. The models' performance was assessed using the MAE metric. The findings revealed that the use of the look-back approach resulted in a 59.12% decrease in MAE compared to that obtained without the approach. These results highlight the significant reduction in MAE's achieved by incorporating the look-back approach in the models.

10. End-to-End Prediction of Parcel Delivery Time with Deep Learning for Smart-City Applications.

- Arthur Cruz de Araujo and Ali Etemad, Senior Member, IEEE

Abstract:

The need for improving customer service and predicting delivery times has motivated postal operators to acquire massive data on parcel delivery. One of the biggest challenges in this domain is last-mile prediction, which deals with factors such as traffic, drivers' behaviors, and weather. This study aims to solve a real-world case of last-mile parcel delivery time prediction using deep learning under the IoT paradigm. The solution is designed as a cloud-based architecture for smart city application. The dataset used in this study is a large-scale parcel dataset covering the Greater Toronto Area (GTA) provided by Canada Post.

Algorithms:

Adam Optimizer

Mean Squared error

Recurrent Neural networks

Conclusion:

The research conducted was focused on the estimation of delivery times for last-mile parcels. To achieve this, a real-world large-scale dataset of parcels that were delivered in the GTA during the first six months of 2017 was used. The dataset was provided by Canada Post, the leading postal operator in Canada. The solution presented in this research is a smart city application under the IoT paradigm. A cloud-based architecture was explored to make the system feasible for real-world usage.

11. Lead time prediction in a flow-shop environment with analytical and machine learning approaches

- D´avid Gyulai Andr´as Pfeiffer G´abor Nick Viola Gallina Wilfried Sihn L´aszl´o Monostori

Abstract:

Manufacturing lead time is a critical performance indicator that companies strive to minimize in order to satisfy customer demands by delivering products promptly. It is a crucial element in most production planning and scheduling methods, and its accuracy is vital in ensuring the efficiency of these processes. However, achieving high accuracy in predicting manufacturing lead time is often a challenge due to the complexity of processes and the variety of products involved. This paper examines various analytical and machine learning prediction techniques and compares their effectiveness in a real flow-shop environment.

Algorithms:

Statistical Interference

Linear Regression

Support Vector Regression

Conclusion:

The test results demonstrate that data analytics and machine learning prediction models are more effective in lead time prediction tasks for non-stationary processes. ML tools offer an advantage in considering job features that vary in a diverse manufacturing environment. Selecting the appropriate prediction model depends on the parameter and process under consideration. Random forests are the best choice for achieving accuracy, while linear methods can provide accurate predictions with well-tuned parameters.

12. A combined order selection and time-dependent vehicle routing problem with time widows for perishable product delivery.

- Zu-Jun Ma, Yao Wu, Ying Dai

Abstract:

The issue of failed delivery is a common problem faced by urban perishable product deliverers, which results in losses such as product deterioration or violation of customers' delivery time windows. This is especially challenging when delivery orders exceed the providers' delivery capacity, and traditional delivery models are not applicable. There are limited studies related to this problem, thus the authors propose a new model that combines order selection and time-dependent vehicle routing problem with time windows in the same framework of perishable product delivery.

Algorithms:

TDTOP, COSTDVRPTW

Framework of HACO.

Local Search Operatons.

Conclusion:

The combined order selection and time-dependent vehicle routing problem with time windows in delivering highly perishable products is examined in this paper. Linearization of the time-dependent travel times of arcs is performed to develop a MIP model for joint decisions on order selection and vehicle routing. The objective is to maximize the total profits of served orders while considering the potential loss from order rejection. A hybrid ant colony algorithm (HACO) is proposed to solve the problem. To evaluate the search efficiency of HACO, problems of different sizes are generated and compared with results from HACO without LNS operator and CPLEX software with fixed run durations. The proposed HACO is found to have relative advantages in terms of solution quality and searching time.

13. Review of Online Food Delivery Platforms and their Impacts on Sustainability.

- Charlene Li, Miranda Mirosa and Phil Bremer 1,2,

Abstract:

The COVID-19 pandemic of 2020 highlighted the benefits of online food delivery (FD), which allowed consumers to easily access prepared meals and kept food providers in operation.

However, concerns have been raised about the negative effects of online FD, leading to consumer and restaurant boycotts. This necessitates an examination of the broader impact of online FD on all stakeholders. By using sustainability's three pillars as a framework, this review analyzes the latest research to identify positive and negative effects. Economically, online FD offers employment and sales opportunities, but restaurant commissions and delivery person working conditions have been criticized.

Algorithms:

It is the review of Food Delivery Platforms so the algorithms are not there.

Conclusion:

The impacts of online food delivery (FD) have been extensively discussed in various studies, and this review has summarized these impacts as presented in Table 3. While the impacts have been classified as positive or negative, it is noteworthy that each impact could be viewed differently. However, it also increased exposure to risks for delivery personnel, a negative impact for them. In conclusion, this interdisciplinary review has three significant contributions. Firstly, it is the first review that has compiled academic research on the wide range of areas impacted by online FD.

14. Supply chain B2B e-commerce and time-based delivery performance.

- Karthik N.S. Iyer

Abstract:

The study aims to explore the relationships between B2B e-commerce in supply chain, environmental uncertainty, organizational structure, and time-based delivery performance. Based on empirical evidence, it was found that B2B e-commerce significantly contributes to time-based delivery performance. The research also indicates that the turbulence of the business process, which is a component of environmental uncertainty, influences the implementation of B2B e-commerce directly and indirectly. The mediating factor in this relationship is the integration dimension of organizational structure.

Algorithms:

Confirmatory factor analysis (CFA), using LISREL.

Conclusion:

The impacts of online food delivery (FD) have been extensively discussed in various studies, and this review has summarized these impacts as presented in Table 3. While the impacts have been classified as positive or negative, it is noteworthy that each impact could be viewed differently. However, it also increased exposure to risks for delivery personnel, a negative impact for them. In conclusion, this interdisciplinary review has three significant contributions. Firstly, it is the first review that has compiled academic research on the wide range of areas impacted by online FD.

15. Route Prediction for Instant Delivery

Abstract:

Customers place orders via the Ele.me website in the normal fast delivery service method.

The platform would alert the businesses to get the takeout packaging ready. To pick up the packages and deliver them to the clients, couriers will be sent out.

Typically, delivery packages include food or necessities. There is frequently a Time-of-Delivery (ToD), which is a deadline for each order, in order to enhance user experiences and boost service quality. For instance, Ele.me guarantees that the food will be delivered within 30 minutes. The one-hour delivery time is a promise made by Amazon Prime Now.

Algorithms used:

Route plan algorithm in ride-sharing, an distance-based algorithm with no courier model, Optimal Route algorithm.

Conclusion:

In this research, we looked at route prediction in an instant delivery scenario. The main goal of route prediction is to create the Perceived Distance Model and to represent the courier's decision-making psychology. The XGBoost model is utilised to create a route prediction algorithm, and the suitable features are chosen using the 2 test. One of the biggest platforms for quick delivery in the world, Ele.me, is where we put our concept into practise. The experimental findings indicate a 16.5% reduction in route prediction inaccuracy and a considerable decrease in late rates from 2.52% to 1.31%.

16. Implementation of Data Mining Prediction Delivery Time Using Linear Regression Algorithm

Abstract:

Online shopping has become commonplace in the current period of modernity. It is also directly tied to freight forwarding services, which are in charge of transporting things purchased online from the vendor to the consumer. In order to ensure that the goods are delivered on time to their destination, purchasers require a quick and secure delivery service. In the transportation industry, one of the most crucial components is customer happiness. Unfortunately, there are a number of roadblocks in the way that slow down the delivery of goods. Thus, one approach to solving this issue is to use data mining technology to forecast delivery timelines.

Algorithm used:

Linear regression algorithm.

Conclusion:

Data processing will be done with prediction methods using the Linear Regression algorithm employing 1,000 datasets made up of 4 Attributes. By using data from the time the items are taken, from the time they are on their way, till they reach the buyer, they may make forecasts, projections, and several analyses so that there won't be any delivery delays in the future. The error of the forecast obtained using this method has an RMSE value of 0.370% based on the RMSE (Root Mean Square Error) value, which is used to calculate the level value. Conclusion: The linear regression algorithm has been shown to be reliable for estimating delivery timeframes.

17. Integration of machine learning prediction and heuristic optimization for mask delivery in COVID-19

Abstract:

In order to safeguard the public, there is a high demand for medical masks due to the novel coronavirus pneumonia (COVID-19). These masks must be distributed to several demand sites. The effectiveness of distribution is essential for epidemic prevention and management. The situation is extremely complicated, nevertheless, due to the enormous demand for masks and the vast number of dispersed demand sites. Also, the time available for solution computation and mask delivery is frequently very short because the exact demands are frequently discovered only after it is too late.

Algorithms used:

A hybrid multi-start genetic algorithm, VNS algorithm, local search algorithm, Heuristic optimization.

Conclusion:

In order to increase the effectiveness and quality of the solutions, we suggest in this work a new two-echelon delivery method based on a mix of machine learning and heuristic optimization. The approach effectively delivers masks to the demand points in each location after pre-distributing predicted demands from depots to regional facilities, reassigning demand points among various regions to balance the variations of predicted demands from real demands, and ultimately. An effective heuristic algorithm for reassigning demand points among various regions and an enhanced heuristic algorithm for two-batch delivery in a region with scarcity are among the important achievements. The effectiveness of the suggested strategy is demonstrated by its application to the supply of emergency medical masks in three Chinese megacities at the height of COVID-19.

18. Travel speed prediction based on learning methods for home delivery.

Abstract:

In many urban transportation scenarios, from the design of delivery routes in freight transportation to the identification of shortest itineraries in cutting-edge traveller information systems, the trip time to move from one site to another in a network is a crucial factor. Therefore, it is crucial to anticipate journey times accurately. Due to congestion brought on, for example, by accidents or severe weather, travel times and vehicle speeds can vary greatly in an urban setting. At a deeper level, one also notices seasonal trends, weekly patterns (such as weekdays versus weekends), and daily patterns (such as rush hours).

Algorithms used:

Affinity propagation algorithm, Clustering, K-means, Random Forest, LSTM.

Conclusion:

This study makes use of a sizable collection of GPS traces gathered from mobile devices deployed inside delivery vans to develop a system for estimating journey speeds. The methodology consists of the following macro-steps: (1) data preparation and representation; (2) size reduction and classification of arcs using unsupervised learning; (3) imputation of missing data; and (4) speed prediction for the previously defined classes of arcs using supervised learning.

19. Delivery Time Estimation for Space Bundles

Abstract:

Recently, internetworking in space has become more popular, mostly for two reasons. Secondly, better resource exploitation made possible by space internetworking makes it easier and safer for space engineers to communicate with deep space. Second, the term "space internetworking" refers to a new era of space networking in which interoperability, interagency collaboration, and the merger of space and terrestrial networking are all conceivable.

Algorithms used:

BDTEalgorithm,CGRalgorithm.

Conclusion:

Using the CGR route computing algorithm, we have provided a novel technique for calculating bundle delivery time in space communications. Our approach is based on a database used for instrumentation that contains statistics for each network node. Previous BER values are determined using some straightforward metrics that are taken out of the database. The Holt-Winters time series forecasting method is used to predict future BER values, which are then used to calculate the approximate total number of transmission rounds for a bundle transmission..

20. Machine Learning based Batching Prediction System for Food Delivery.

Abstract:

Estimates of delivery times are crucial for platforms that supply meals online. These platforms also rely on batching, or combining two orders into one, to boost productivity and cut costs.We suggest a unique system for improved delivery time estimations for batched orders in this study. The system bases its predictions on a number of machine learning algorithms that cooperate. We note that the approach increases the frequency of food deliveries within the predicted delivery periods by roughly 6%.

1. K-Nearest Neighbour

K-Nearest Neighbors (KNN) is a popular machine learning algorithm used for classification and regression tasks. It is a non-parametric algorithm that does not make any assumptions about the underlying distribution of the data. Instead, it uses a simple concept of finding the k-nearest data points to a given query point and making predictions based on the majority class or average value of the k-nearest neighbors.

In the classification task, KNN works by finding the k-nearest neighbors of the query point and assigning the class label based on the majority class of those neighbors. For instance, if the majority of the k-nearest neighbors of a query point belong to class A, then the query point is classified as class A. In the regression task, KNN works by finding the k-nearest neighbors of the query point and averaging their values to make the prediction for the query point. For instance, if the k-nearest neighbors of a query point have values [10, 12, 14], then the prediction value for the query point will be (10+12+14)/3 = 12.

The choice of k is a hyperparameter that needs to be tuned to get optimal performance. A small value of k may lead to overfitting, while a large value of k may lead to underfitting. The distance metric used to calculate the distance between the query point and the neighbors is an important consideration. The most commonly used distance metrics are Euclidean distance and Manhattan distance. KNN is a lazy learning algorithm, which means it does not require a training phase. Instead, it stores the training dataset in memory and uses it during the prediction phase.

KNN suffers from the curse of dimensionality, which means that its performance deteriorates as the number of features or dimensions in the dataset increases. KNN can be used for both binary and multiclass classification tasks, as well as for regression tasks. KNN has a number of advantages, including its simplicity, flexibility, and ease of implementation. It is also a non-parametric algorithm, which means it can capture complex relationships between variables.

However, KNN has a number of disadvantages as well. One major disadvantage is that it can be computationally expensive, especially for large datasets. Another disadvantage is that it can be sensitive to the choice of distance metric used. To improve the performance of KNN, various techniques such as feature selection, dimensionality reduction, and distance metric learning can be used.

Overall, KNN is a powerful and widely used algorithm in the field of machine learning, and it has numerous applications in areas such as image recognition, text classification, and medical diagnosis.

2. Support Vector Machine (SVM)

Support Vector Machines (SVM) is a popular machine learning algorithm used for classification, regression, and outlier detection tasks. SVM is a powerful algorithm that works by finding the best hyperplane in a high-dimensional space that separates the different classes.

In the classification task, SVM works by finding the hyperplane that maximizes the margin between the two classes. The margin is defined as the distance between the hyperplane and the closest data points from each class. The hyperplane that maximizes this margin is known as the maximum margin hyperplane.

SVM can handle both linearly separable and non-linearly separable data by using the kernel trick. The kernel function maps the data to a higher-dimensional space where it becomes linearly separable, thus allowing SVM to find the maximum margin hyperplane.

SVM can also handle multi-class classification by using one-vs-one or one-vs-all strategies. In the one-vs-one strategy, SVM trains a binary classifier for each pair of classes, while in the one-vs-all strategy, SVM trains a binary classifier for each class against all the other classes. In the regression task, SVM works by finding the hyperplane that best fits the data. This hyperplane is known as the maximum margin hyperplane for regression.

SVM has a number of advantages, including its ability to handle high-dimensional data, its robustness to outliers, and its effectiveness in handling non-linearly separable data. SVM can also handle small and noisy datasets with high accuracy. However, SVM has a number of disadvantages as well. One major disadvantage is that it can be computationally expensive, especially for large datasets. Another disadvantage is that it is sensitive to the choice of kernel function and its parameters.

To improve the performance of SVM, various techniques such as feature selection, dimensionality reduction, and parameter tuning can be used.

Overall, SVM is a powerful and widely used algorithm in the field of machine learning, and it has numerous applications in areas such as image recognition, text classification, and bioinformatics.

3. Decision Tree

Decision Tree is a popular machine learning algorithm used for classification and regression tasks. It is a tree-based model that partitions the data into subsets based on a set of rules learned from the data.

In the classification task, decision tree works by recursively partitioning the data into subsets based on the most discriminative feature. The feature with the highest information gain or Gini index is selected as the splitting criterion. Information gain measures the reduction in entropy or disorder in the data, while Gini index measures the probability of misclassifying a randomly chosen sample from a given class.

In the regression task, decision tree works by recursively partitioning the data into subsets based on the most discriminative feature. The feature with the highest reduction in variance is selected as the splitting criterion. Reduction in variance measures the decrease in variance of the target variable after partitioning the data based on the selected feature. Decision tree is a simple and interpretable algorithm that can handle both categorical and continuous data. It is also robust to outliers and missing data.

However, decision tree has a tendency to overfit the data, which means it can memorize the training data and fail to generalize to new data. To prevent overfitting, various techniques such as pruning, setting a minimum number of samples per leaf, and limiting the depth of the tree can be used.

Decision tree can also be used for ensemble learning, where multiple decision trees are trained on different subsets of the data and combined to make the final prediction. Random Forest and Gradient Boosting are two popular ensemble methods based on decision trees.

Overall, decision tree is a powerful and widely used algorithm in the field of machine learning, and it has numerous applications in areas such as finance, healthcare, and natural language processing.

4. Random Forest

Random forest is a popular machine learning algorithm that is used for both classification and regression tasks. It belongs to the family of ensemble methods, which combines multiple models to improve the accuracy of predictions. The basic idea behind random forest is to create multiple

decision trees on different subsets of the data, and then combine their outputs to make a final prediction.

Random forest is an extension of the decision tree algorithm, which is a simple but powerful machine learning algorithm. A decision tree works by recursively splitting the data into smaller subsets based on the values of the features, until the subsets are homogeneous or a stopping criterion is met. Each subset corresponds to a node in the tree, and the final predictions are made by following the path from the root to a leaf node that corresponds to the input features.

While decision trees are easy to understand and interpret, they suffer from several limitations. One of the main problems is overfitting, which occurs when the tree is too complex and captures the noise in the data rather than the underlying pattern. Another problem is that decision trees are sensitive to the choice of the splitting criteria, and small changes in the data or the algorithm can result in different trees.

Random forest overcomes these limitations by creating multiple decision trees on different subsets of the data, and then combining their outputs to make a final prediction. The key idea is to introduce randomness in the creation of the trees, so that each tree is different and captures different aspects of the data. The randomness can be introduced in two ways: by randomly selecting a subset of the features at each split, and by randomly sampling the data with replacement.

The first step in the random forest algorithm is to create a bootstrap sample of the data, which is a random sample of the same size as the original data but with replacement. This means that some instances may be selected multiple times, while others may not be selected at all. The bootstrap sample is used to create a decision tree, which is trained on a subset of the features selected randomly at each split.

This process is repeated multiple times to create a forest of decision trees. The number of trees is a hyper parameter that can be tuned to optimize the performance of the algorithm. Once the trees are created, they can be used to make predictions on new data by aggregating their outputs. In classification tasks, the most common aggregation method is to use a majority vote, where each tree assigns a class to the input data, and the final prediction is the class that receives the most votes. In regression tasks, the most common aggregation method is to use the mean or median of the outputs of the trees.

5. XG Boost

XGBoost is a popular machine learning algorithm used for classification, regression, and ranking tasks. It is an optimized implementation of gradient boosting algorithm that utilizes decision trees as base models.

Gradient boosting is an ensemble method that combines weak learners, such as decision trees, to create a stronger and more accurate predictor. The key idea of gradient boosting is to iteratively add new trees to the ensemble, where each new tree corrects the mistakes made by the previous trees. This way, the ensemble can gradually learn from the data and improve its performance.

XGBoost improves upon the traditional gradient boosting algorithm by introducing a number of optimizations and regularization techniques that enhance the speed, accuracy, and scalability of the algorithm. Some of the key features of XGBoost include:

1. Handling of missing values: XGBoost can handle missing values in the data by assigning the most common value of the feature to the missing values.
2. Regularization: XGBoost employs two types of regularization techniques: L1 regularization (Lasso) and L2 regularization (Ridge). These techniques help prevent overfitting and improve the generalization of the model.
3. Weighted quantile sketch: XGBoost uses a weighted quantile sketch algorithm to approximate the gradient statistics for each split, which significantly improves the speed and accuracy of the algorithm.
4. Early stopping: XGBoost allows early stopping of the training process when the performance on the validation set stops improving. This helps prevent overfitting and saves computation time.
5. Parallel processing: XGBoost can run in parallel on multiple cores or machines, which makes it highly scalable and able to handle large datasets.
6. Handling imbalanced data: XGBoost has built-in functionality to handle imbalanced datasets by weighting the samples in each class according to their frequency.
7. Custom objective function: XGBoost allows the user to define their own objective function, which can be useful for tackling specific machine learning problems.

XGBoost has been successful in numerous Kaggle competitions and has become one of the most widely used algorithms in the field of machine learning. It has numerous applications in areas such as finance, healthcare, and natural language processing.

To use XGBoost, the user needs to define the objective function, the loss function, and the hyperparameters of the algorithm. The objective function specifies the goal of the model, such as minimizing the mean squared error or maximizing the log-likelihood. The loss function measures the error between the predicted and actual values. The hyperparameters of the algorithm, such as the learning rate, the number of trees, and the maximum depth of the trees, can be tuned using techniques such as grid search or randomized search.

Overall, XGBoost is a powerful and versatile algorithm that can handle a wide range of machine learning problems. Its speed, accuracy, and scalability make it a popular choice for many data scientists and machine learning practitioners.

6. Logistic Regression

Logistic regression is a popular machine learning algorithm used for classification tasks. It is a simple yet powerful algorithm that is widely used in a variety of applications, such as medical diagnosis, credit scoring, and marketing analytics. The main idea behind logistic regression is to model the probability of a binary response variable based on one or more predictor variables.

In logistic regression, the response variable is a binary variable that takes on one of two values, typically 0 or 1. The predictor variables can be either continuous or categorical, and their relationship with the response variable is modeled using a logistic function, which maps the predictor variables to the probability of the response variable being 1.

The logistic function is a sigmoid curve that takes on values between 0 and 1. It is defined as:

$p(x) = 1 / (1 + \exp(-z))$

where $p(x)$ is the probability of the response variable being 1, x is a vector of predictor variables, and z is a linear combination of the predictor variables and their coefficients:

$z = \beta 0 + \beta 1 x1 + \beta 2 x2 + ... + \beta p xp$

where $\beta 0$ is the intercept term, $\beta 1$ to βp are the coefficients of the predictor variables, and x1 to xp are the values of the predictor variables.

The logistic regression model is trained on a labeled dataset, where each observation has a value of the response variable and a set of values for the predictor variables. The goal is to estimate the coefficients of the logistic function that best fit the data, so that the predicted probabilities of the response variable are as close as possible to the actual probabilities.

The coefficients are estimated using maximum likelihood estimation, which is a statistical method that finds the values of the coefficients that maximize the likelihood of the observed data. The likelihood is the probability of the observed data given the values of the coefficients, and it is defined as the product of the probabilities of the individual observations:

$L(\beta) = \Pi i=1n\ p(xi)yi\ (1 - p(xi))1-yi$

where n is the number of observations, xi is the vector of predictor variables for observation i, yi is the value of the response variable for observation i, and p(xi) is the predicted probability of the response variable being 1 for observation i.

The logistic regression model can be evaluated using various metrics, such as accuracy, precision, recall, and F1-score. These metrics depend on the choice of a threshold value for the predicted probabilities, which determines the classification of the observations as 0 or 1. The threshold value can be chosen based on the specific needs of the application, such as maximizing the true positive rate or minimizing the false positive rate.

Logistic regression has several advantages over other machine learning algorithms. One of the main advantages is that it is simple and easy to interpret, since the coefficients of the model can be directly related to the odds of the response variable being 1. For example, if the coefficient of a predictor variable is positive, it means that an increase in the value of the predictor variable increases the odds of the response variable being 1, while if the coefficient is negative, it means that an increase in the value of the predictor variable decreases the odds of the response variable being 1.

Another advantage of logistic regression is that it can handle both continuous and categorical variables, and can model their nonlinear relationships with the response variable using polynomial or interaction terms. Moreover, logistic regression is a parametric algorithm, which means that it makes assumptions about the distribution of the data and the functional form of the model. This can be an advantage when the assumptions are met, since it can result in a more efficient and stable model.

However, logistic regression also has some limitations. One of the main limitations is that it assumes that the relationship

Dataset Description:

Size of training set: 11,094 records

Size of test set: 2,774 records

FEATURES:

- Restaurant: A unique ID that represents a restaurant.
- Location: The location of the restaurant.
- Cuisines: The cuisines offered by the restaurant.
- Average_Cost: The average cost for one person/order.
- Minimum_Order: The minimum order amount.
- Rating: Customer rating for the restaurant.
- Votes: The total number of customer votes for the restaurant.
- Reviews: The number of customer reviews for the restaurant.
- Delivery_Time: The order delivery time of the restaurant. (Target Classes)

Code and Output:

Data Cleaning

We will consider the following features of training and test datasets. Here I have grouped the columns based on similarity of values which will make it easy for cleaning.Cleaning Average_Cost & Minimum_order columns to remove special characters and make them float variables.Location & Cuisines are categoriacal variables that needs to be cleaned before encoding. Rating, Votes & Reviews needs to be cleaned and converted to respective types.We will delve deeper in the coming sections.

Average_Cost & Minimum_order

```
train.head()
```

	Restaurant	Location	Cuisines	Average_Cost	Minimum_Order	Rating	Votes	Reviews	Delivery_Time
0	ID_3744	Dockyard Road, Mumbai CST Area	Fast Food, Chinese	₹200	₹50	3.8	91	44	45 minutes
1	ID_5984	Delhi Cantt.	North Indian	₹100	₹50	2.9	37	16	30 minutes
2	ID_13	MG Road, Pune	Continental, North Indian, Seafood, Cafe, Sala...	₹650	₹50	4.1	3860	1962	45 minutes
3	ID_643	Delhi Administration Flats, Timarpur	Naga	₹250	₹50	3.9	313	119	45 minutes
4	ID_5754	Pune University	Pizza, Fast Food	₹200	₹50	3.3	86	48	45 minutes

```
#Finding the unique values in Average_Cost
train['Average_Cost'].unique()
```

```
array(['₹200', '₹100', '₹650', '₹250', '₹400', '₹150', '₹50', '₹600',
       '₹300', '₹350', '₹500', '₹450', '₹850', '₹750', '₹1,000', '₹700',
       '₹800', '₹550', '₹1,200', '₹900', 'for', '₹950', '₹1,150'],
      dtype=object)
```

The Average_Cost has an invalid value in one of its rows. We will replace it with 200 which is the most frequent value in the column.(Check the train.describe() method). We will add 200 as a string and not as an integer as the column is of type object and all its values are strings.

```
train[train['Average_Cost'] == 'for']
```

	Restaurant	Location	Cuisines	Average_Cost	Minimum_Order	Rating	Votes	Reviews	Delivery_Time
2102	ID_6472	Pune University	Fast Food	for	₹50	NEW	-	-	30 minutes

```
#replacing 'for' with 200
train['Average_Cost'].replace('for', '200', inplace = True)
```

Now we will clean all the values and will convert it in to integer.

```
train['Average_Cost_Cleaned'] = train['Average_Cost'].apply(lambda x: int(re.sub("[^0-9]", "", x)))
```

```
train['Average_Cost_Cleaned'].unique()
```

```
array([ 200,  100,  650,  250,  400,  150,   50,  600,  300,  350,  500,
        450,  850,  750, 1000,  700,  800,  550, 1200,  900,  950, 1150])
```

We can see that all the special characters ahave been removed and strings have been converted to integers.

Let's perform the same operations on the test set

```
test['Average_Cost'].unique()
```

```
array(['₹100', '₹200', '₹350', '₹50', '₹150', '₹650', '₹600', '₹300',
       '₹500', '₹1,000', '₹400', '₹250', '₹450', '₹550', '₹800', '₹750',
       '₹900', '₹1,100', '₹850', '₹1,200', '₹1,400', '₹2,050', '₹700'],
      dtype=object)
```

```
test['Average_Cost_Cleaned'] = test['Average_Cost'].apply(lambda x: int(re.sub("[^0-9]", "", x)))
```

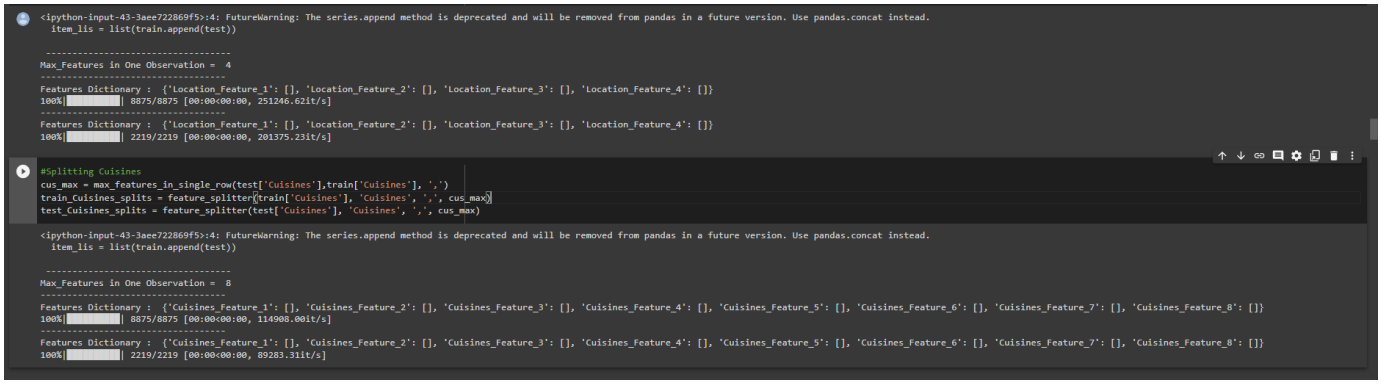
```
test['Average_Cost_Cleaned'].unique()
```

```
array([ 100,  200,  350,   50,  150,  650,  600,  300,  500, 1000,  400,
        250,  450,  550,  800,  750,  900, 1100,  850, 1200, 1400, 2050,
        700])
```

LOCATION & CUISINES:

Location and Cuisines are categorical variables that need to be encoded later. By looking at the dataset, we can see that each of these columns have multiple values in them. We can use each of them as a feature by splitting each column into n number of features. To do that we will first find the maximum number(n) of features a column has in the entire dataset including both test and train data. Once the maximum number of features within a cell is found, we will split all the rows in the dataset for that specific column into n features.

```
#A function to find the maximun number of features in a single cell
def max_features_in_single_row(train, test, delimiter):
  max_info = 0
  item_lis = list(train.append(test))
  for i in item_lis:
    if len(i.split("{}".format(delimiter))) > max_info:
      max_info = len(i.split("{}".format(delimiter)))
  print("\n","-"*35)
  print("Max_Features in One Observation = ", max_info)
  return max_info
#This function splits a column in to n features where n is the maximum number
of features in a single cell
def feature_splitter(feat, name, delimiter, max_info):
  item_lis = list(feat)
  extracted_features = {}

  for i in range(max_info):
    extracted_features['{}_Feature_{}'.format(name, i+1)] = []

  print("-"*35)
  print("Features Dictionary : ", extracted_features)

  for i in tqdm(range(len(item_lis))):
    for j in range(max_info):
      try:
        extracted_features['{}_Feature_{}'.format(name,j+1)].append(item_lis[i
].split("{}".format(delimiter))[j].lower().strip())
      except:
        extracted_features['{}_Feature_{}'.format(name, j+1)].append(np.nan)


  return extracted_features

#Splitting Location
loc_max = max_features_in_single_row(test['Location'],train['Location'], ',')
train_Location_splits = feature_splitter(train['Location'], 'Location', ',', l
oc_max)
test_Location_splits = feature_splitter(test['Location'], 'Location', ',', loc
_max)
```



Rating, Votes & Reviews

We will now clean Rating, Votes & Reviews columns to remove invalid values and to convert them to the right type.

```
[ ] train.describe(include = 'all')
```

	Restaurant	Location	Cuisines	Average_Cost	Minimum_Order	Rating	Votes	Reviews	Delivery_Time	Average_Cost_Cleaned	Minimum_Order_Cleaned
count	8875	8875	8875	8875	8875	8875	8875	8875	8875	8875.000000	8875.000000
unique	6363	35	1920	23	15	33	1024	711	7	NaN	NaN
top	ID_4625	Mico Layout, Stage 2, BTM Layout,Bangalore	North Indian	₹200	₹50	-	-	-	30 minutes	NaN	NaN
freq	16	753	681	2594	8115	961	1676	1859	5919	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	203.059155	53.219155
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	128.385830	17.803072
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	50.000000	0.000000
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	100.000000	50.000000
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	200.000000	50.000000
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	200.000000	50.000000
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1200.000000	450.000000

```
#A function to find all the non numeric values
def non_numerals(series):
  non_numerals = []
  for i in series.unique():
    try :
      i = float(i)
    except:
      non_numerals.append(i)
  return non_numerals
# A function to replace the non-numeric values
def replace_nn_with(series, type_, fill_with = None, method = 'mean'):

  nn = non_numerals(series)
  print('-'*30)
  print('-'*30)
  print("Non Numerals in column ",series.name," : ",nn)

  series = series.replace(nn, np.nan, inplace = False)
  nulls = series.isnull().sum()
  if fill_with:
    series.fillna(fill_with, inplace = True)
    print("Filling Non Numerals with {}".format(fill_with))

  else:
    series = series.replace(nn, np.nan, inplace = False)
```

```
    if method == 'mean' :
      rep = series.astype(float).mean()
      print("Filling Non Numerals with MEAN = ", rep)

    elif method == 'median' :
      rep = series.astype(float).median()
      print("Filling Non Numerals with MEDIAN = ", rep)

    elif method == 'min' :
      rep = series.astype(float).min()
      print("Filling Non Numerals with MINIMUM = ", rep)

    else:
      print('Please pass a valid method as a string --
 ("mean" or "median" or "min")')
      return 0

    series.fillna(rep, inplace = True)

  try:
    series = series.astype(type_)
    print(nulls, ": observations replaced")
    return series
  except:
    # Since type conversion of a string containting decimals to int is not pos
sible, it is first converted to float
    series =  series.astype(float)
    print(nulls, ": observations replaced")
    series =  series.astype(type_)
    return series
```

```
[ ] train.head()
```

	Restaurant	Location	Cuisines	Average_Cost	Minimum_Order	Rating	Votes	Reviews	Delivery_Time	Average_Cost_Cleaned	Minimum_Order_Cleaned
0	ID_3744	Dockyard Road, Mumbai CST Area	Fast Food, Chinese	₹200	₹50	3.8	91	44	45 minutes	200	50
1	ID_5984	Delhi Cantt.	North Indian	₹100	₹50	2.9	37	16	30 minutes	100	50
2	ID_13	MG Road, Pune	Continental, North Indian, Seafood, Cafe, Sala...	₹650	₹50	4.1	3860	1962	45 minutes	650	50
3	ID_643	Delhi Administration Flats, Timarpur	Naga	₹250	₹50	3.9	313	119	45 minutes	250	50
4	ID_5754	Pune University	Pizza, Fast Food	₹200	₹50	3.3	86	48	45 minutes	200	50

Lets Clean the columns

```
[ ] train['Rating_Cleaned'] = replace_nn_with(train['Rating'],float, method = 'mean')

    ------------------------------
    ------------------------------
    Non Numerals in column  Rating  :  ['NEW', '-', 'Temporarily Closed', 'Opening Soon']
    Filling Non Numerals with MEAN =  3.618973796131156
    1586 : observations replaced

[ ] test['Rating_Cleaned'] = replace_nn_with(test['Rating'],float, fill_with = 3.6134596429744668)

    ------------------------------
    ------------------------------
    Non Numerals in column  Rating  :  ['-', 'NEW', 'Opening Soon']
    Filling Non Numerals with 3.6134596429744668
    377 : observations replaced

[ ] train['Votes_Cleaned'] = replace_nn_with(train['Votes'],int,method =  'mean')

    ------------------------------
    ------------------------------
    Non Numerals in column  Votes  :  ['-']
    Filling Non Numerals with MEAN =  253.60161133490763
    1676 : observations replaced
```

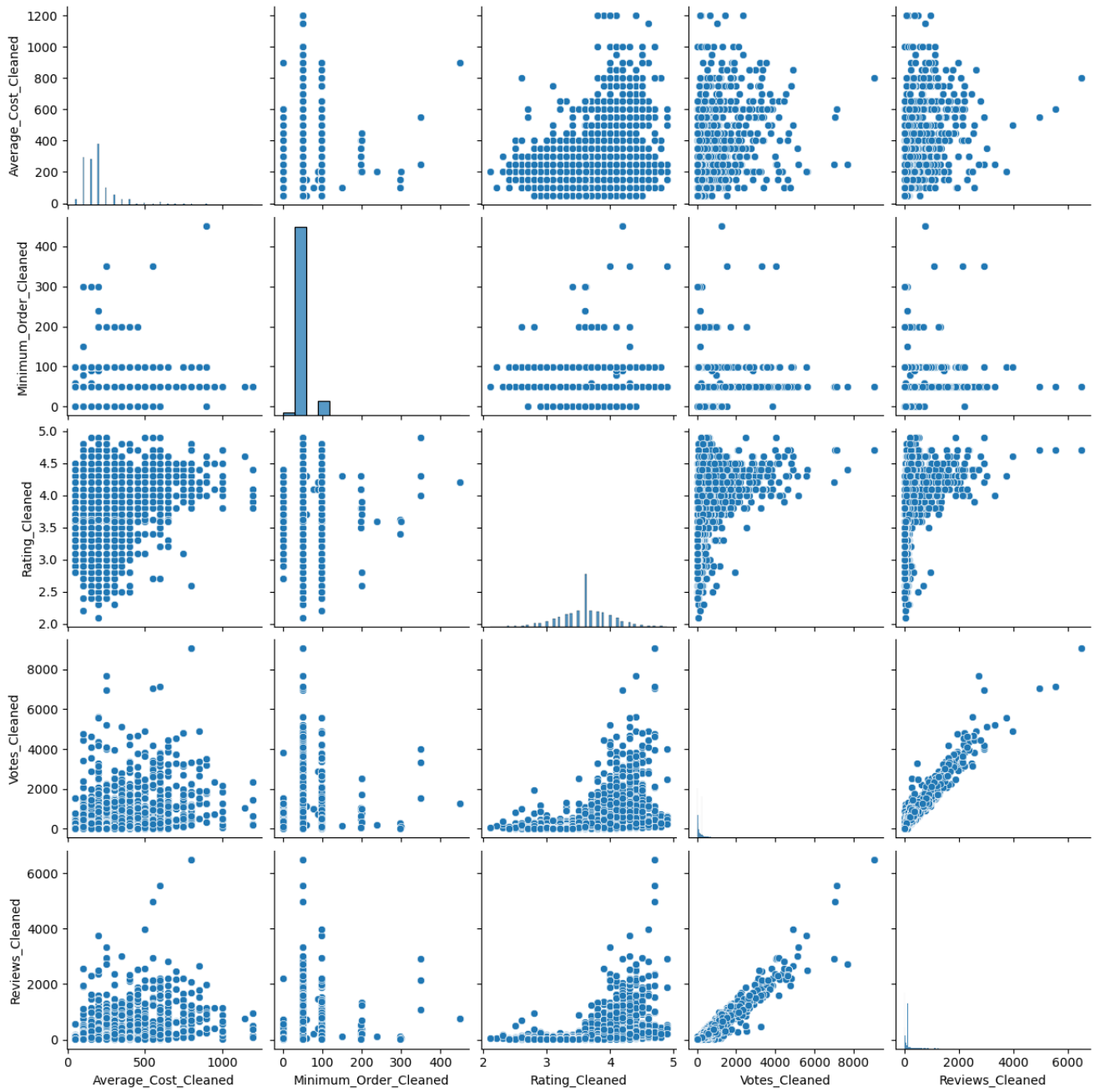
```
[ ] test['Votes_Cleaned'] = replace_nn_with(test['Votes'],int,fill_with = 244.54445676274943)

    ------------------------------
    ------------------------------
    Non Numerals in column  Votes  :  ['-']
    Filling Non Numerals with 244.54445676274943
    398 : observations replaced

[ ] train['Reviews_Cleaned'] = replace_nn_with(train['Reviews'],int, method =  'mean')

    ------------------------------
    ------------------------------
    Non Numerals in column  Reviews  :  ['-']
    Filling Non Numerals with MEAN =  127.1063283922463
    1859 : observations replaced

[ ] test['Reviews_Cleaned'] = replace_nn_with(test['Reviews'],int, method =  'mean',fill_with = 123.247893 )

    ------------------------------
    ------------------------------
    Non Numerals in column  Reviews  :  ['-']
    Filling Non Numerals with 123.247893
    453 : observations replaced

    train.head(5)
```

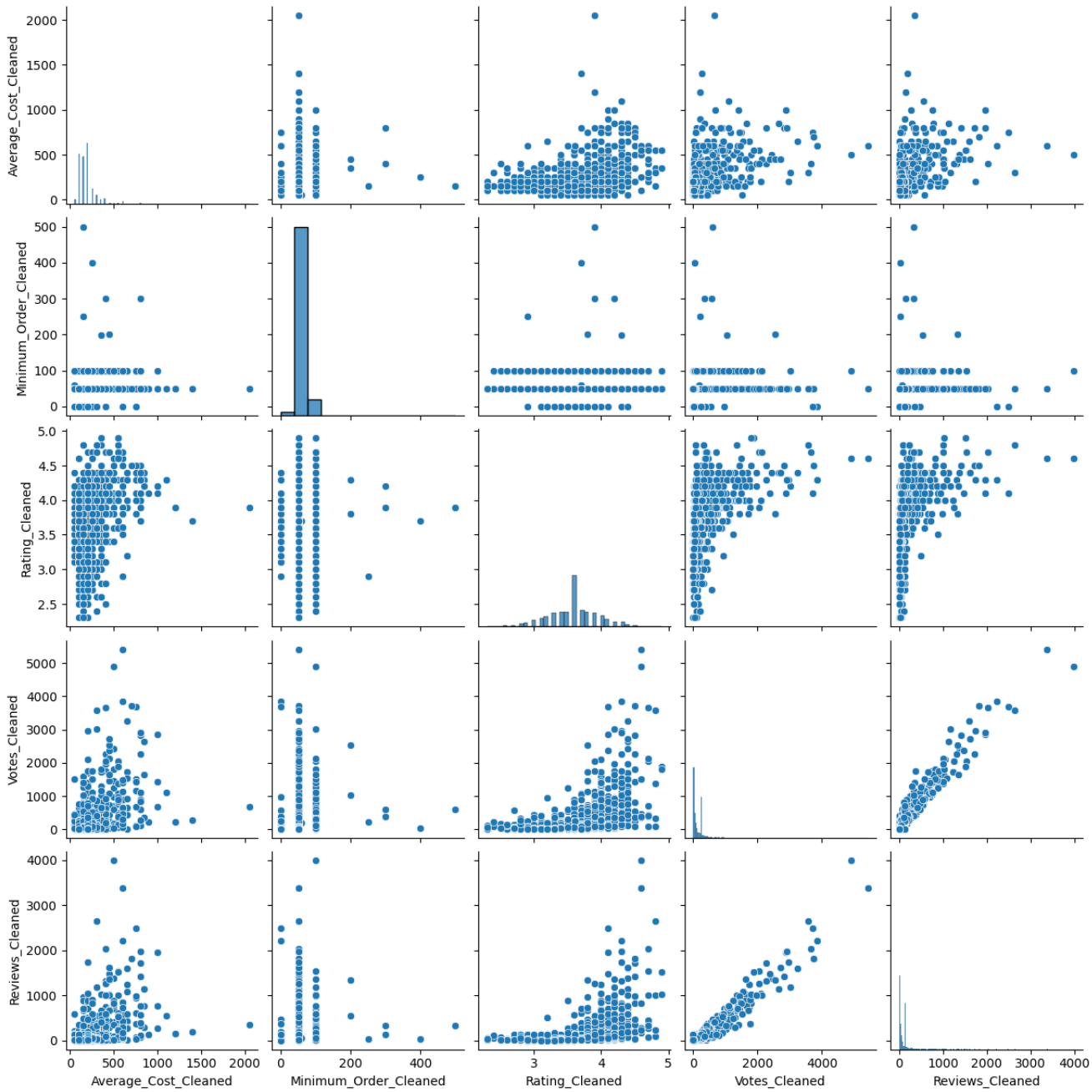
	Restaurant	Location	Cuisines	Average_Cost	Minimum_Order	Rating	Votes	Reviews	Delivery_Time	Average_Cost_Cleaned	Minimum_Order_Cleaned	Rating_Cleaned	Votes_Cleaned	Reviews_Cleaned
0	ID_3744	Dockyard Road, Mumbai CST Area	Fast Food, Chinese	₹200	₹50	3.8	91	44	45 minutes	200	50	3.8	91	44
1	ID_5984	Delhi Cantt.	North Indian	₹100	₹50	2.9	37	16	30 minutes	100	50	2.9	37	16
2	ID_13	MG Road, Pune	Continental, North Indian, Seafood, Cafe, Sala...	₹650	₹50	4.1	3860	1962	45 minutes	650	50	4.1	3860	1962
3	ID_643	Delhi Administration Flats, Timarpur	Naga	₹250	₹50	3.9	313	119	45 minutes	250	50	3.9	313	119
4	ID_5754	Pune University	Pizza, Fast Food	₹200	₹50	3.3	86	48	45 minutes	200	50	3.3	86	48

We are done with the cleaning part and now we will select only the columns we need for further stages.

We will now merge all the cleaned features to form a perfect data frame

```
train_sample = pd.concat([pd.DataFrame(train_Location_splits), pd.DataFrame(tr
ain_Cuisines_splits),train_sample],sort=False,axis = 1)
test_sample = pd.concat([pd.DataFrame(test_Location_splits), pd.DataFrame(test
_Cuisines_splits), test_sample],sort=False,axis = 1)
#lets take a look at the relation between the numeric features in the dataset
import seaborn
seaborn.pairplot(train_sample)
```


Average_Cost_Cleaned
Minimum_Order_Cleaned
Rating_Cleaned
Votes_Cleaned
Reviews_Cleaned



Data Preprocessing:

After cleaning, there are still some Data-Preparation tasks left. We still have some missing values that we allotted during the feature generation.

In the Preprocessing stage we will perform the following :

Dealing with Nulls/empty cells

Encoding Categorical variables

Scaling the features.

Removing Nulls

```
train_sample.isnull().sum()
```

```
Location_Feature_1          0
Location_Feature_2       1566
Location_Feature_3       5103
Location_Feature_4       8058
Cuisines_Feature_1          0
Cuisines_Feature_2       2383
Cuisines_Feature_3       5521
Cuisines_Feature_4       7502
Cuisines_Feature_5       8368
Cuisines_Feature_6       8687
Cuisines_Feature_7       8787
Cuisines_Feature_8       8836
Restaurant                  0
Average_Cost_Cleaned        0
Minimum_Order_Cleaned       0
Rating_Cleaned              0
Votes_Cleaned               0
Reviews_Cleaned             0
Delivery_Time               0
dtype: int64
```

Since the nulls are present only in the categorical features, in this approach, to make it simple I will just replace all the NaNs with a string 'NAN' and will use it as a added category or class.

```
[ ] train_sample.fillna('NAN', inplace = True)
```

```
[ ] train_sample.isnull().sum()
```

```
Location_Feature_1      0
Location_Feature_2      0
Location_Feature_3      0
Location_Feature_4      0
Cuisines_Feature_1      0
Cuisines_Feature_2      0
Cuisines_Feature_3      0
Cuisines_Feature_4      0
Cuisines_Feature_5      0
Cuisines_Feature_6      0
Cuisines_Feature_7      0
Cuisines_Feature_8      0
Restaurant              0
Average_Cost_Cleaned    0
Minimum_Order_Cleaned   0
Rating_Cleaned          0
Votes_Cleaned           0
Reviews_Cleaned         0
Delivery_Time           0
dtype: int64
```

```
train_sample.isnull().sum()
```

```
Location_Feature_1      0
Location_Feature_2      0
Location_Feature_3      0
Location_Feature_4      0
Cuisines_Feature_1      0
Cuisines_Feature_2      0
Cuisines_Feature_3      0
Cuisines_Feature_4      0
Cuisines_Feature_5      0
Cuisines_Feature_6      0
Cuisines_Feature_7      0
Cuisines_Feature_8      0
Restaurant              0
Average_Cost_Cleaned    0
Minimum_Order_Cleaned   0
Rating_Cleaned          0
Votes_Cleaned           0
Reviews_Cleaned         0
```

Encoding Categories:

Here we will use a simple Label Encoder to transform all the strings or categories.

Locations & Cuisines:

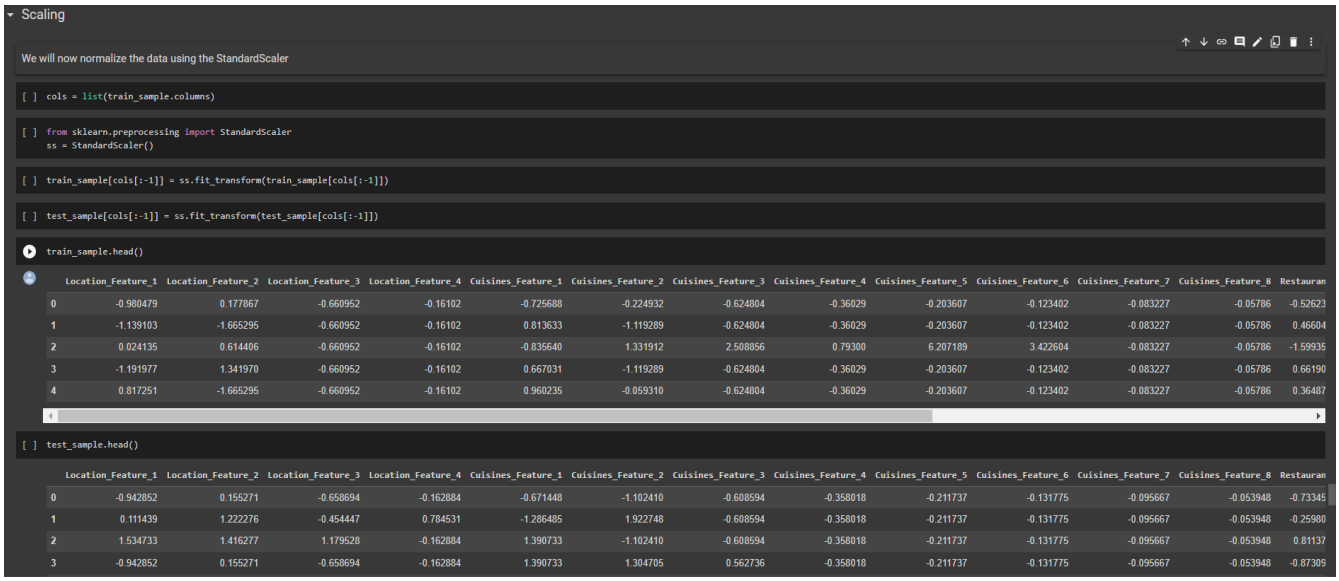
We will first find the unique values or categories in each of the categorical features and fit the label encoder with the unique values. The encoder eill assign an integer code to each of the categories which can be used to tranform the entire categorical feature column.

```
temp1 = []
for i in train_Cuisines_splits.keys():
  for j in train_Cuisines_splits.get(i):
    temp1.append(j)

temp2 = []
for i in test_Cuisines_splits.keys():
  for j in test_Cuisines_splits.get(i):
    temp2.append(j)

temp1.extend(temp2)

unique_cuisines = list(pd.Series(temp1).unique())
unique_cuisines.append('NAN')
temp1 = []
for i in train_Location_splits.keys():
  for j in train_Location_splits.get(i):
    temp1.append(j)

temp2 = []
for i in test_Location_splits.keys():
  for j in test_Location_splits.get(i):
    temp2.append(j)

temp1.extend(temp2)

unique_locations = list(pd.Series(temp1).unique())
unique_locations.append('NAN')
#encoding the categorical Features
from sklearn.preprocessing import LabelEncoder
le_c = LabelEncoder().fit(unique_cuisines)
le_l = LabelEncoder().fit(unique_locations)
for i in train_Location_splits.keys():
  train_sample[i] = le_l.transform(train_sample[i])

for i in train_Cuisines_splits.keys():
  train_sample[i] = le_c.transform(train_sample[i])
for i in train_Location_splits.keys():
  test_sample[i] = le_l.transform(test_sample[i])
for i in test_Cuisines_splits.keys():
  test_sample[i] = le_c.transform(test_sample[i])
```

Restaurant IDs

We will follow a similar approach for encoding the Restaurant IDs

```
t1 = list(train_sample['Restaurant'])
t2 = list(test_sample['Restaurant'])

t1.extend(t2)
unique_ids = list(set(t1))
len(unique_ids)
le_id = LabelEncoder().fit(unique_ids)
le_id = LabelEncoder().fit(unique_ids)
```



Scaling

We will now normalize the data using the StandardScaler

```
[ ] cols = list(train_sample.columns)

[ ] from sklearn.preprocessing import StandardScaler
    ss = StandardScaler()

[ ] train_sample[cols[:-1]] = ss.fit_transform(train_sample[cols[:-1]])

[ ] test_sample[cols[:-1]] = ss.fit_transform(test_sample[cols[:-1]])

train_sample.head()
```

	Location_Feature_1	Location_Feature_2	Location_Feature_3	Location_Feature_4	Cuisines_Feature_1	Cuisines_Feature_2	Cuisines_Feature_3	Cuisines_Feature_4	Cuisines_Feature_5	Cuisines_Feature_6	Cuisines_Feature_7	Cuisines_Feature_8	Restauran
0	-0.980479	0.177867	-0.660952	-0.16102	-0.725688	-0.224932	-0.624804	-0.36029	-0.203607	-0.123402	-0.083227	-0.05786	-0.52623
1	-1.139103	-1.665295	-0.660952	-0.16102	0.813633	-1.119289	-0.624804	-0.36029	-0.203607	-0.123402	-0.083227	-0.05786	0.46604
2	0.024135	0.614406	-0.660952	-0.16102	-0.835640	1.331912	2.508856	0.79300	6.207189	3.422604	-0.083227	-0.05786	-1.59935
3	-1.191977	1.341970	-0.660952	-0.16102	0.667031	-1.119289	-0.624804	-0.36029	-0.203607	-0.123402	-0.083227	-0.05786	0.66190
4	0.817251	-1.665295	-0.660952	-0.16102	0.960235	-0.059310	-0.624804	-0.36029	-0.203607	-0.123402	-0.083227	-0.05786	0.36487

```
[ ] test_sample.head()
```

	Location_Feature_1	Location_Feature_2	Location_Feature_3	Location_Feature_4	Cuisines_Feature_1	Cuisines_Feature_2	Cuisines_Feature_3	Cuisines_Feature_4	Cuisines_Feature_5	Cuisines_Feature_6	Cuisines_Feature_7	Cuisines_Feature_8	Restauran
0	-0.942852	0.155271	-0.658694	-0.162884	-0.671448	-1.102410	-0.608594	-0.358018	-0.211737	-0.131775	-0.095667	-0.053948	-0.73345
1	0.111439	1.222276	-0.454447	0.784531	-1.286485	1.922748	-0.608594	-0.358018	-0.211737	-0.131775	-0.095667	-0.053948	-0.25980
2	1.534733	1.416277	1.179528	-0.162884	1.390733	-1.102410	-0.608594	-0.358018	-0.211737	-0.131775	-0.095667	-0.053948	0.81137
3	-0.942852	0.155271	-0.658694	-0.162884	1.390733	1.304705	0.562736	-0.358018	-0.211737	-0.131775	-0.095667	-0.053948	-0.87309

Modelling:

Finally, we are ready for modelling. We will split the training set into training and validation sets. We will then use the training set to train and validation set to test the performance of the model. Finally we will use the given test set for predicting.

XGBOOST:

```
from sklearn.model_selection import train_test_split

train, val = train_test_split(train_sample, test_size = 0.1, random_state = 123)
```

```
X_train = train[cols[:-1]]
Y_train = train[cols[-1]]

X_Val = val[cols[:-1]]
Y_Val = val[cols[-1]]

X_test = test_sample[cols[:-1]]
```

```
[ ] from xgboost import XGBClassifier

    xgb = XGBClassifier()

    xgb.fit(X_train,Y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='multi:softprob', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

```
[ ] xgb.score(X_Val,Y_Val)
```

```
0.7216216216216216
```

This is a baseline model. Fine tuning the model can give better results. Also, tryout different algorithms to find the best.

NAÏVE-BAYES, KNN, SVM, DECISION TREE CLASSIFIER, RANDOM FOREST CLASSIFIER:

```
[ ] # NAIVE - BAYES
    from sklearn.naive_bayes import GaussianNB
    nb_model = GaussianNB()
    nb_model.fit(X_train, Y_train)
    nb_model.score(X_Val,Y_Val)
```

```
0.01463963963963964
```

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
KNN_model=KNeighborsClassifier()
KNN_model.fit(X_train, Y_train)
KNN_model.score(X_Val,Y_Val)
```

```
0.6745495495495496
```

```
[ ] #SVM
    from sklearn.svm import SVC
    svm_model = SVC()
    svm_model.fit(X_train, Y_train)
    svm_model.score(X_Val,Y_Val)
```

```
0.7105855855855856
```

```
[ ] #DecisionTreeClassifier
    from sklearn.tree import DecisionTreeClassifier
    clf = DecisionTreeClassifier(random_state=0)
    clf.fit(X_train,Y_train)
    clf.score(X_Val,Y_Val)
```

```
0.6801801801801802
```

```
[ ] #RandomForestClassifier
    from sklearn.ensemble import RandomForestClassifier
    Rforest = RandomForestClassifier()
    Rforest.fit(X_train,Y_train)
    Rforest.score(X_Val,Y_Val)
```

```
0.7635135135135135
```

ENSEMBLE LEARNING FOR CLASSIFIER:

```
[ ] #Ensemble learning for classifier
    from sklearn.ensemble import VotingClassifier
    cl1= DecisionTreeClassifier()
    cl2 = RandomForestClassifier()
    cl3 = SVC()
    cl4 = KNeighborsClassifier()
    cl5 = GaussianNB()
    cl_voting =VotingClassifier(estimators=[('DT', cl1),('RF', cl2),('svm',cl3),('KNN', cl4),('NB',cl5)],voting='hard')
    cl_voting.fit(X_train, Y_train)
    cl_voting.score(X_Val,Y_Val)
```

```
0.7297297297297297
```

```
import warnings
warnings.filterwarnings('ignore')
Predictions = Rforest.predict(X_test)
pd.DataFrame(Predictions, columns = ['Delivery_Time']).to_excel("/content/Submission.xlsx", index = False)
```

Conclusion:

For more accurate delivery predictions for orders that were batched, the Batching Prediction System was developed. The likelihood of batching for each order placed in the cart is predicted by the system's initial module. The second module receives orders with a high likelihood of batching, and it forecasts the buffer value to be placed on top of the initial delivery time estimations. In the paper, various machine learning algorithms and modelling approaches were examined for each of these modules. The overall effect of the system was also underlined in the report. For batched orders, the BPS results in a 5.5% gain in compliance. Although it lengthens estimates for average delivery times, it does not significantly reduce conversion rates. One of the many data science algorithms used by our company to maintain a positive client experience while increasing delivery effectiveness is BPS.

References:

1. Adak, A., Pradhan, B., Shukla, N., & Alamri, A. (2022). Unboxing deep learning model of food delivery service reviews using explainable artificial intelligence (XAI) technique. *Foods*, *11*(14), 2019.
2. Shahbazi, Z., & Byun, Y. C. (2020). A procedure for tracing supply chains for perishable food based on blockchain, machine learning and fuzzy logic. *Electronics*, *10*(1), 41.
3. Bender, J., & Ovtcharova, J. (2021). Prototyping machine-learning-supported lead time prediction using AutoML. *Procedia Computer Science*, *180*, 649-655.
4. Khiari, J., & Olaverri-Monreal, C. (2020, November). Boosting algorithms for delivery time prediction in transportation logistics. In *2020 International Conference on Data Mining Workshops (ICDMW)* (pp. 251-258). IEEE.
5. Gyulai, D., Pfeiffer, A., Bergmann, J., & Gallina, V. (2018). Online lead time prediction supporting situation-aware production control. *Procedia CIRP*, *78*, 190-195.
6. ŞAHİN, H., & Duygu, İ. Ç. E. N. Application of Random Forest Algorithm for the Prediction of Online Food Delivery Service Delay. *Turkish Journal of Forecasting*, *5*(1), 1-11.
7. Bender, J., Trat, M., & Ovtcharova, J. (2022). Benchmarking automl-supported lead time prediction. *Procedia Computer Science*, *200*, 482-494.
8. Lin, H. E., Zito, R., & Taylor, M. (2005, September). A review of travel-time prediction in transport and logistics. In *Proceedings of the Eastern Asia Society for transportation studies* (Vol. 5, pp. 1433-1448).

9. Erkmen, O. E., Nigiz, E., Sarı, Z. S., Arlı, H. Ş., & Akay, M. F. (2022). Delivery Time Prediction Using Support Vector Machine Combined with Look-back Approach.
10. de Araujo, A. C., & Etemad, A. (2021). End-to-end prediction of parcel delivery time with deep learning for smart-city applications. *IEEE Internet of Things Journal*, *8*(23), 17043-17056.
11. Gyulai, D., Pfeiffer, A., Nick, G., Gallina, V., Sihn, W., & Monostori, L. (2018). Lead time prediction in a flow-shop environment with analytical and machine learning approaches. *IFAC-PapersOnLine*, *51*(11), 1029-1034.
12. Ma, Z. J., Wu, Y., & Dai, Y. (2017). A combined order selection and time-dependent vehicle routing problem with time widows for perishable product delivery. *Computers & Industrial Engineering*, *114*, 101-113.
13. Li, C., Mirosa, M., & Bremer, P. (2020). Review of online food delivery platforms and their impacts on sustainability. *Sustainability*, *12*(14), 5528.
14. Iyer, K. N., Germain, R., & Frankwick, G. L. (2004). Supply chain B2B e-commerce and time-based delivery performance. *International Journal of Physical Distribution & Logistics Management*, *34*(8), 645-661..
15. Zhang, Y., Liu, Y., Li, G., Ding, Y., Chen, N., Zhang, H., ... & Zhang, D. (2019). Route prediction for instant delivery. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *3*(3), 1-25.
16. Wahyudi, T., & Arroufu, D. S. (2022). Implementation of Data Mining Prediction Delivery Time Using Linear Regression Algorithm. *Journal of Applied Engineering and Technological Science (JAETS)*, *4*(1), 84-92.
17. Chen, X., Yan, H. F., Zheng, Y. J., & Karatas, M. (2023). Integration of machine learning prediction and heuristic optimization for mask delivery in COVID-19. *Swarm and Evolutionary Computation*, *76*, 101208.
18. Gmira, M., Gendreau, M., Lodi, A., & Potvin, J. Y. (2020). Travel speed prediction based on learning methods for home delivery. *EURO Journal on Transportation and Logistics*, *9*(4), 100006.
19. Bezirgiannidis, N., Burleigh, S., & Tsaoussidis, V. (2013). Delivery time estimation for space bundles. *IEEE Transactions on Aerospace and Electronic Systems*, *49*(3), 1897-1910.
20. Moghe, R. P., Rathee, S., Nayak, B., & Adusumilli, K. M. (2021, January). Machine learning based batching prediction system for food delivery. In *Proceedings of the 3rd ACM India Joint International Conference on Data Science & Management of Data (8th ACM IKDD CODS & 26th COMAD)* (pp. 316-322).