# LF Logistics - ETA Prediction

# Final Year Project Report

**Supervisor Prof. Wei YOU**

TSANG, Yu
CHAN, Kwan Kiu
LIN, Xintong

**Department of Industrial Engineering and Decision Analytics**
**The Hong Kong University of Science and Technology**

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Industrial Engineering & Decision Analytics 工業工程及決策分析

LI & FUNG

Project ID Number: 3

| Project Title | LF Logistics - ETA Prediction | | |
|---|---|---|---|
| Supervisor | Prof. Wei YOU | | |
| Department | Industrial Engineering and Decision Analytics | | |
| University | The Hong Kong University of Science and Technology | | |
| Team Members | TSANG, Yu (Leader) | CHAN, Kwan Kiu | LIN, Xintong |
| Contribution Nature | Data processing, Model building, Visualization | Data processing, Visualization | Data processing, Visualization |
| Project Contribution | 34% | 33% | 33% |
| Write-up Contribution | 34% | 33% | 33% |

Project ID Number: 3

**Acknowledgment**

We want to sincerely thank our FYP supervisor, Prof. Wei YOU from the IEDA department, for his valuable support throughout this project. His expertise, guidance, and encouragement have been instrumental in achieving the project's goals.

We would also like to thank Mr Jesmer WONG (Manager of Supply Chain Analytics) and Victor Blancada (Senior Manager of Supply Chain Analytics) from Li & Fung for their assistance in providing datasets and elucidation about Li & Fung. Their dedication and hard work have been greatly appreciated.

Finally, we would like to acknowledge the support and assistance provided by our significant mates and IEDA Department for their assistance in providing support and encouragement. Without their help, this project would not have been possible.

Project ID Number: 3

**Abstract**

Accurate Estimated Time of Arrival (ETA) predictions are critical to the logistics industry. This project aims to improve the accuracy of Estimated Time of Arrival (ETA) shipment predictions by analyzing data provided by the logistics firm Li & Fung. The project utilizes data such as planned start and end times, van types, and initial and target locations to identify patterns and relationships. The data is then cleaned, and models are run to estimate a more accurate parcel ETA. Any deviation from the expected timeline is highlighted, and modifications are made accordingly. The project includes visualizations of the route and a plotly dashboard with filters to make it easy for stakeholders with different needs to interpret the data. The findings of this project have significant implications for logistics providers seeking to optimize their operations and improve their customer satisfaction.

# TABLE OF CONTENTES

Project ID Number: 3

# 1. Background

## 1.1 Importance of ETA

The estimated arrival time (ETA) is a crucial way for logistics companies to monitor and manage shipment details. The importance of ETA as a communication tool cannot be neglected. It allows all parties involved to plan effectively because the sender may provide a realistic time frame for the arrival of a cargo or person, allowing the receiver to plan their calendar accordingly. ETA can help detect any procedural or system concerns that may impact the delivery process and provide clarity. With this information, logistics companies like Li & Fung can enhance their services and minimize future delays or inconveniences. Delivering shipments on time and in good condition is crucial to maintaining a positive relationship with clients. Any delays or inconveniences can adversely impact this relationship, and insights gained from the project are essential to maintaining goodwill. The project findings can aid logistic companies in identifying any system or procedural issues that may affect the delivery process and reflect on areas for improvement in future shipments.

## 1.2 Project Overview

The logistics firm Li & Fung has provided data for this project to improve ETA predictions for shipments. This project identifies patterns and relationships by analyzing data such as planned start and end times, van types, and initial and target locations. The data is then cleaned, and models are run to estimate a more accurate parcel ETA. Any deviation from the expected timeline is highlighted, and modifications are made accordingly. This project also includes visualizations of the route and a plotly dashboard with filters for easy interpretation by stakeholders with different needs.

## 2. Literature review

### 2.1 More consideration on factors to improve accuracy

The paper titled "Vessel estimated time of arrival prediction system based on a path-finding Algorithm" presents a method to estimate the arrival time of vessels in ports (Park, Sim, & Bae, 2021). The authors emphasize the need for accurate ETA prediction for effective port operations and timely cargo delivery. Incorrect ETA forecasts can cause port congestion, cargo delivery delays, and increased shipping company costs.

The research provides a path-finding method that considers numerous ETA parameters such as vessel speed, wind speed, current and projected weather conditions. The technique uses a dynamic programming approach to consider numerous route alternatives and select the best path depending on various constraints. Historical data is used to train the model and improve its accuracy of ETA.

However, there are some limitations to this proposed algorithm. One limitation is the reliance on accurate and up-to-date data on various factors, such as weather conditions and vessel speed. Any inaccuracies in the data can lead to inaccurate ETA predictions. Moreover, there is a lack of consideration of additional factors such as port congestion vessel maintenance.

To summarize, faulty data input should be addressed while creating the model, and more elements that can affect ETA should be included to increase ETA accuracy.

### 2.2 Selections of ETA prediction algorithms

The paper "Review and Comparison of prediction algorithms for the estimated time of arrival using geospatial transportation data" compared the performance of various prediction algorithms for estimating the ETA of vehicles (Kwon, Park, & Bae, 2021). The performance of ETA prediction algorithms is said to be heavily dependent on the input data's precision, the prediction models' quality, and the parameters employed in the models.

The authors then compare and contrast several ETA prediction techniques, such as linear regression, decision trees, and neural networks. They compare how these algorithms perform on various forms of transportation data, such as road networks and air trips. The findings reveal that the accuracy of ETA prediction systems varies greatly depending on the mode of transportation and the algorithm used. The author suggests that integrating real-time data such as weather and traffic conditions can increase the accuracy of ETA prediction models.

To summarize, no one ETA prediction system can be used for all types of transportation. Instead, the algorithm of choice is determined by the transportation system's specific requirements and the quality and quantity of available data.

## 3. Problem Identification

### 3.1 Missing data set and unmodified data collection process

The first challenge faced in this project is the limited accuracy of the data sources given by the company. The initial data set provided has some missing inputs in certain columns, which impact any later analysis or modeling efforts and lower the accuracy of the ending prediction outcomes. The missing values may be due to the unmodified data collection process. For instance, situations such as drivers forgetting to update the exact start time of the trip could be one of the reasons causing missing sessions inside the data source. The missing data in the provided dataset could also be attributed to several other non-human error. One example could be data corruption during storage, transfer, or processing. This problem can significantly impact the accuracy of ETA predictions, leading to late deliveries, increased costs, and customer dissatisfaction.

### 3.2 Unconsidered variables

The second challenge is the need for consideration of some critical variables that can impact the actual arrival time of shipments. Although the assumed speeds used in the prediction have already taken the average of the traffic time within the city, this ETA prediction is likely to stay the same for different parcel types, local weather and traffic conditions, or any other potential variables that may affect the actual arrival time, causing room for improvement. Besides, some of the expected data fields need to be included, such as shipping orders, special arrangements, etc. This information could help system designers verify whether the routes are optimized for time efficiency and serve as a valuable reference for future deliveries. These variables are necessary to ensure accurate ETA predictions, which may adversely affect the logistics firm's efficiency and profitability.

### 3.3 Inconsistent model for different regions

The third challenge lies in the model's inconsistency across diverse regions. Accurately estimating travel time becomes difficult when considering various road conditions in different regions. For example, developing countries often face traffic issues that impede the shipment consolidation process. In contrast, regions with well-developed infrastructure may experience fewer delays. Consequently, there is a pressing need to enhance ETA prediction methods to accommodate such varying circumstances. The absence of a unified ETA prediction system for different regions can result in inconsistencies in ETA forecasts, which can negatively affect a logistics firm's performance and profitability. This, in turn, may lead to dissatisfied customers and increased operational costs.

## 4. Proposed Solution

### 4.1 Objectives

Based on the aforementioned problems identified. We plan to leverage the advantages of OSM for ETA forecasting to design and develop machine learning models using historical delivery data. The performance of the model can be evaluated on a test dataset and compared with existing methods. Once the model is developed and tested, it can be integrated into the firm's existing systems to track and monitor deliveries in real-time. We would also suggest some way the company can take in action for improvements to address some of the initial data collection issues.

### 4.2 Improving data quality and accuracy

In order to address the issue found, it is essential to identify the root causes of the missing data and implement appropriate data cleaning, imputation, or collection strategies to improve the dataset's quality and accuracy. If possible, we would suggest the company to improve the current data collection process by implementing automated data collection methods that reduce the likelihood of human error. This could include using RFID sensors or GPS devices that automatically record trip start and end times, reducing the likelihood of missing data. Secondly, we will use data validation and cleaning processes in this project to identify and correct missing or inaccurate data. This process includes using statistical methods to impute missing values and removing incomplete data points. By implementing these solutions, the company can improve the accuracy of its data sources, resulting in more accurate ETA predictions, reduced costs, and increased customer satisfaction.

### 4.3 Enhancing ETA prediction accuracy with additional variables

To cope with the lack of consideration of critical variables, we believe the company can implement several solutions. Firstly, data on traffic and weather conditions could be implemented into the ETA prediction algorithms in our project. This will enable the system to adjust the predicted arrival time based on the current conditions, resulting in more accurate predictions. Secondly, the company can collect and analyze data on different parcel types and their delivery times to identify patterns and optimize their routes accordingly. This will help to ensure that the routes are optimized for time efficiency, resulting in faster and more reliable deliveries. Thirdly, more information can be gathered from customers in advance, such as shipping orders and special arrangements, to improve the accuracy of their ETA predictions. This information can also serve as a valuable reference for future deliveries as well. By implementing these solutions, logistics firms can improve their efficiency and profitability by reducing the number of delayed or missed deliveries.

Project ID Number: 3

4.4 Integrated ETA prediction system for different regions

We believe the company can implement an integrated ETA prediction system for different regions. This system can consider the unique road conditions, traffic patterns, and infrastructure of each area to provide more accurate ETA predictions. Data on road conditions, traffic patterns, and infrastructure for each region could be collected. This data can be used to train machine learning models that can accurately predict ETA based on the specific conditions of each region. Additionally, the company can use real-time data from GPS tracking devices to update ETA predictions in real-time. This can help account for unexpected delays or changes in road conditions that may impact delivery times. Implementing a region-specific ETA prediction system can help the firm improve its performance and minimizing operational costs. By providing more accurate ETA predictions, logistics firms can better plan and optimize their operations.

## 5. Methodology

### 5.1 Libraries import and data input

Several libraries and packages are imported, including pandas, numpy, os, matplotlib, folium, osmnx, networkx, difflib, ortools, Keras, and XGBoost. These libraries and packages are used for various purposes such as data manipulation, visualization, and optimization.

Also, we imports various libraries and modules for building and evaluating machine learning models, including scikit-learn, scipy, joblib. It demonstrates how to perform regression tasks, hyperparameter tuning, and dimensionality reduction techniques, such as RandomForestRegressor, MLPRegressor, StandardScaler, OneHotEncoder, LabelEncoder, RandomizedSearchCV, TruncatedSVD, and TSNE.

Then, we read a CSV file located at a specified path and create a pandas dataframe containing the data in the file.

### 5.2 Data processing

Basic data cleaning and manipulation tasks are done, including data type conversion and column renaming, that are necessary for subsequent data analysis and modeling tasks. Firstly, the columns in the data frame are renamed with a list of new column names and 5 columns with names starting with “Time-” was converted to datetime format. Secondly, a dictionary is created to map each column name to the desired data type and then make a new data frame with accurate data types. Thirdly, for missing values in the column of “Time4” (actual ending time), they are filled with the values of “Time 5” (the miscellaneous time). Fourthly, the 5 columns starting with “Time-” are renamed with corresponding time name.

New columns are created using arithmetic operations between existing columns, such as plan duration time, actual duration and actual duration for non-null data. Then, the column “shipNo” is divided into several new columns to identify the shipment with more details, such as “companyCode”, “sysCode”, “countryCode”, “warehouse” and “shipNo”. Furthermore, two columns are created called 'origLoc' and 'destLoc' by concatenating the 'Start-Lng' and 'Start-Lat' columns, and the 'Dest-Lng' and 'Dest-Lat' columns, respectively. After that, the column “shipNo” and the original 4 columns for concatenation are dropped from the data frame.

For deduplication and reindex, two functions called “primary_sort” and “drop_duplicate” are defined.

The first function takes a dataframe, a list of columns to sort by, a list of columns to check for duplicates, and a target column to merge duplicates on as input. This function sorts the input dataframe by the specified columns in ascending order and identifies duplicates based on the specified columns. For each set of duplicates, a unique index is assigned and stored in a new column called “is_duplicated_index”, and a dictionary mapping each index to the corresponding set of duplicate values is created. The function then creates a new column called

"shipNo_dummy" that contains the index values for each row, and drops unnecessary columns from the dataframe. Finally, the function renames the "shipNo_dummy" column to the original target column name and returns the resulting dataframe.

The second function called "drop_duplicate" that takes a pandas data frame, a list of primary columns to sort by, a column to sort on separately, and a column to aggregate on as input. The function sorts the input data frame by the primary columns in ascending order, identifies duplicate rows based on the primary columns, assigns a unique index value to each non-duplicate row, and creates a new column in the data frame containing the unique index values for each non-duplicate row. The resulting data frame is returned with the original "agg_col" column replaced with a new column containing the unique index values.

For future use, we collect the points of each route based on shipment information and generate one new column called "Route".

Firstly, two empty lists called "dest" and 'route' are created. Then, we iterate over the rows of the data. For each row, if the current index is less than the length of the data frame as well as the current index is 0, the "origLoc" and "destLoc" values are appended to the 'route' list. If the "sameRoute" value for the current row is False and the index is not 0, the current 'route' list is appended to the "dest" list, the 'route' list is reset, and the "origLoc" and "destLoc" values are appended for the current row to the 'route' list. If the 'sameRoute' value for the current row is True, and the "destLoc" value for the current row is not equal to the last element in the "route" list, the "destLoc' value is appended to the "route" list. If the current index is equal to the length of the dataframe minus one, the final 'route' list is appended to "dest".

Secondly, we create a new pandas data frame called "df_dest" with a single column and resetting its index, as well create a new pandas data frame called "raw_clean" by filtering the original data frame and reset its index, and create a new column in 'raw_clean' by assigning the "df_dest" data frame to this column. Finally, a sorted and unduplicated dataset with route is done.

5.3 New features

To consider more environmental factors that may impact the accuracy of ETA, historical weather data is obtained from the National Centers for Environmental Information (NCEI) website. The weather data for Hong Kong is obtained by choosing the weather station code 45007, and the same method is used to find weather data for other regions. And here is the weather station code summary for this project:

| Location | Weather Station Code | Location |
|---|---|---|
| Hong Kong (HKG) | 45007 | HONG KONG INTERNATIONAL, HK |
| Taiwan (TWN) | 58968 | TAIBEI, CH |
| Thailand (THA) | 48455 | BANGKOK METROPOLIS, TH |
| Singapore (SGP) | 48698 | SINGAPORE CHANGI INTERNATIONAL, SN |
| Malaysia (MYS) | 48665 | MALACCA, MY |
| Indonesia (IDN) | 96749 | SOEKARNO HATTA INTERNATIONAL, ID |
| China (CHN) | 54511 | BEIJING CAPITAL INTERNATIONAL AIRPORT, CH |
| Philippines (PHL) | 98429 | NINOY AQUINO INTERNATIONAL, RP |

The weather data file is read into a pandas DataFrame and processed to prepare it for further analysis. The desired columns are selected, and the latitude and longitude columns are merged into a point. The code replaces specific values with 0, converts a column to a string, splits it into six separate columns, and drops unwanted columns. The NAME column is modified, and rows not equal to the chosen weather station are dropped. The STATION column is converted to a string, and only the first five digits are kept. The POINT column is moved to be after the STATION column, and the columns are reordered.

The code then converts the DATE column in the 'df_weather_hk' DataFrame and the 'planStartTime' column in 'raw_clean' to datetime format. The two DataFrames are merged based on the 'planStartTime' and 'DATE' columns using the 'merge_asof()' function. Finally, the unused columns are dropped from the resulting merged DataFrame.

The resulting dataset will be used for the model learning part and to predict the ETA, taking into account the environmental factors such as temperature, precipitation, and wind speed.

It should be noted that due to some restrictions, the weather data used is based on the main and common weather station for each region. Therefore, the weather data may not cover the exact location of each point when mapping the weather data into the DataFrame. However, this

approach still provides valuable insights into the environmental factors that may affect the ETA, making it a useful tool for optimizing transportation and logistics.

Apart from weather, traffic conditions are also key factors that impact the accuracy of ETA. We have found TomTom Traffic Stats, which is a website that provides access to traffic data and insights for cities and countries around the world. It offers real-time and historical traffic information, such as congestion levels, travel times, and incident reports, based on anonymous data collected from millions of GPS-enabled devices.

The objective of this report is to describe a methodology for improving the accuracy of ETA calculations on an OSM map by utilizing TomTom traffic data. To achieve this, the following tools and techniques were employed:

**Data acquisition**: TomTom traffic data was obtained by purchasing it or signing up for a developer account on the TomTom Developer Portal and the free-tier is enough for this project usage. The traffic data was retrieved for the road network in the area of interest using the TomTom traffic APIs.

**Data processing**: Various data processing tools were utilized to transform the TomTom traffic data to match the format of the OSM data and merge the two datasets. This involved converting the data to a compatible file format, matching the road segments in the TomTom data to the corresponding edges in the OSM data, and merging the traffic data with the OSM data. The tools used included Python, Pandas, and GeoPandas.

**Edge speed updating**: The speed attributes of the edges in the OSM data were updated based on the traffic data using the average or current speed of each road segment from the TomTom data. This allowed for more accurate ETA calculations on an OSM map.

**ETA calculation**: Routing engines and libraries that support custom edge speeds, such as OSRM and GraphHopper, were used to calculate the ETA for a given route more accurately. These engines and libraries take into account the updated edge speeds in the OSM data to provide a more accurate ETA.

**Regular updates**: Automation tools such as pipelines and scripts were utilized to ensure that the edge speeds in the OSM data were updated regularly based on the latest traffic data from TomTom. This was critical because traffic conditions can change frequently, and updated data was necessary to maintain the accuracy of ETA calculations.

Overall, by implementing this methodology and using these tools and techniques, the accuracy of ETA calculations on an OSM map was significantly improved. This can be particularly useful for applications that require real-time traffic information, such as navigation systems and logistics planning.

5.4 Route calculation and map display

*'loadGraph(graphml_file="hongkong_speed.graphml")'*:

This program function loads a street network for Hong Kong using the OpenStreetMap API and assigns speed data to the edges in the street network based on the time of day. The speed data is loaded from two JSON files ('rush_hour_path' and 'non_rush_hour_path'), which contain TomTom speed data. The speed data is classified as rush hour or non-rush hour based on the time of day, and is assigned to the edges in the street network using the 'ox.speed' module. The rush hour intervals are defined as 7:00 AM to 10:00 AM and 4:00 PM to 7:00 PM. The function returns the loaded street network. The TomTom speed data is loaded into a dictionary that also stores the rush hour classification. The function then assigns the speed data to the edges in the street network using the 'ox.speed' module.

*'shortestRoute(graph, orig:(float, float), dest:(float, float), optimizer="travel_time")'*:

This program function uses the 'ox.nearest_nodes' function to find the nearest nodes in the network to the origin and destination coordinates. It then calculates the shortest route between these nodes using the 'nx.shortest_path' function with the specified optimizer. If an exception is raised, meaning a route cannot be found, the function tries again with the origin and destination positions exchanged. The function returns the shortest route as a list of node IDs.'routeLength': This function calculates the total length of the shortest route in kilometers, and returns the result as an integer.

'*routeTime(graph, shortest_route)*':

This program function calculates the total time required to travel the shortest route in minutes using the 'ox.utils_graph.get_route_edge_attributes' function to get the travel time for each edge in the route. The function then sums the travel times and converts the result to minutes by dividing by 60. Finally, the function returns the result as an integer.

*'routeLength(graph, shortest_route)'*:

This program function calculates the total length of the shortest route in kilometers using the 'ox.utils_graph.get_route_edge_attributes' function to get the length of each edge in the route. The function then sums the lengths and converts the result to kilometers by dividing by 1000. Finally, the function returns the result as an integer.

*'tunnelTest(nodes: int)*':

This program function checks whether the input nodes include the IDs for a specific tunnel, and returns either the name of the tunnel or "None" depending on the result. If the tunnel is included in the input nodes, the function returns the name of the tunnel. Otherwise, it returns "None".

*'getDepotFromDf(df)'*: This program function determines the most common value in the 'origLoc' column of a given dataframe. This value represents the depot location. The function then returns this value as output.

*'removeDupRoutes(routes, simRatio)':*

This program function removes duplicate routes from a given route list. It then checks for pairs of remaining routes that have a similarity ratio greater than the specified ratio. If such a pair is found, it removes one of the routes from the list. The function returns the cleaned list of routes. ‘lngLatStrToFloat’: This function takes a string point in the format of "longitude,latitude" and returns a tuple of two floats representing the longitude and latitude respectively.

*'routeLength_for_distMatrix(graph, orig, dest, weight)':*

This program function calculates the shortest path length between two points in a graph using the networkx library, and returns the result in kilometers. The 'orig' and 'dest' parameters are tuples of two floats representing the longitude and latitude of the origin and destination points respectively. The 'graph' parameter is a graph object. The 'weight' parameter is a string parameter indicating the type of weight to be used in the calculation.

*'VRP(graph, solution_list)':*

‘VRP’: The 'VRP' function takes a list of strings representing the coordinates of delivery locations and the depot as input. It returns a folium map object, the total route length, and the total route time. The function calculates the shortest route between each pair of consecutive nodes in the solution list using the 'shortestRoute' function. It then plots the resulting routes on the folium map object using the 'ox.plot_route_folium' function. The function also calculates the total route length and time by calling the 'routeLength' and 'routeTime' functions. Finally, the function returns the folium map object, the total route length, and the total route time as output.

*'decode_route(list_lnglat: list, list_order: list)':*

The 'decode_route' function takes a list of strings representing the coordinates of nodes and a list of integers representing the order of the nodes as input. It returns a list of strings representing the coordinates of the nodes in the correct order. The function removes duplicate nodes that are adjacent to each other and nodes that are less than 100 meters apart. It does this by calculating the distance between the current node and the previous node using the Haversine formula. If the distance is greater than or equal to 0.1 kilometers, the current node is added to the route order list. Otherwise, it is skipped. Finally, the function returns the route order list as output.

*'getDistanceMatrix(routelist, optimizer = "length")':*

The 'getDistanceMatrix' function takes a list of strings representing the coordinates of nodes as input and an optimizer parameter to specify whether to optimize for distance or travel time. It returns a list of lists representing the distance matrix between each pair of nodes. The function calculates the distance between each pair of nodes using the 'routeLength' function and stores the distances in the distance matrix. It does this by iterating through the nodes in the input list and calculating the distance between each pair of nodes using the 'shortestRoute' and 'routeLength' functions. Finally, the function returns the distance matrix as a list of lists. The optimizer parameter is used to specify whether to optimize for distance or travel time.

*'create_data_model(list)':*

This function returns a dictionary containing the data required for the OR-Tools library to optimize the order of stops along the route. It creates a dictionary named data and stores the distance matrix of the locations using the getDistanceMatrix function with a distance type of "length". It also sets the number of vehicles to 1 and the depot (starting point) to index 0.

*'print_solution(data, manager, routing, solution)':*

The 'print_solution' function provides a visual representation of the optimized route for the user. It loops through the vehicles in the optimization problem, gets the start index of each vehicle, and iterates through the indices of the optimized route for the vehicle. It calculates the cost of the arc between the current index and the next index and adds it to the total distance of the route. The function stores the optimized route for each vehicle in the 'routingSol' list and returns the optimized route for the first vehicle in 'routingSol'. This function is useful for providing a visual representation of the optimized route to the user.

*'get_routes(solution, routing, manager)':*

The 'get_routes' function retrieves the vehicle routes from a solution and stores them in a two-dimensional array. It loops through the vehicles in the routing and gets the start index of each vehicle. It then iterates through the indices of the optimized route for the vehicle until it reaches the end index. For each iteration, it appends the current index to the route array. Once it reaches the end index, it appends the end index to the route array and appends the route array to the routes array. Finally, the function returns the routes array. This function is useful for retrieving the optimized routes for each vehicle in the routing from a solution.

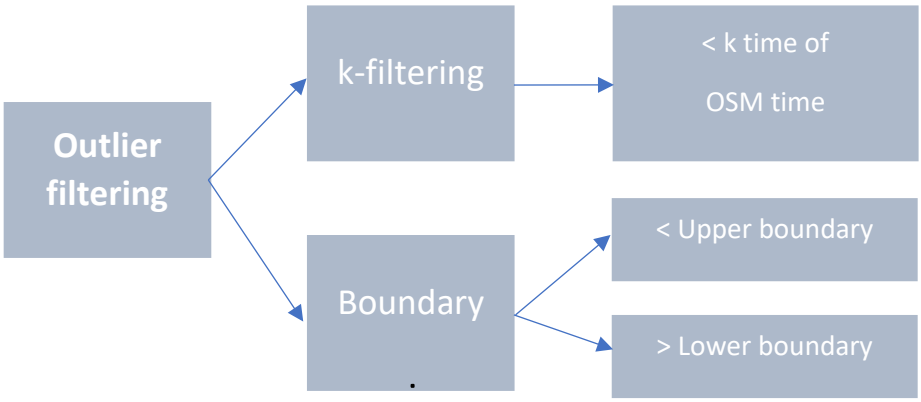
*'ortool(list)':*

The 'ortool' function is the entry point of the program and sets up the optimization problem for the OR-Tools library. It creates a data model using the 'create_data_model' function. It creates a routing index manager and a routing model using classes from the OR-Tools library. It then defines a transit callback to calculate the distance between two nodes. This modified version of the function is using the Google OR-Tools library for our use case. The function adds a distance constraint to the optimization problem and sets the first solution strategy to 'PATH_CHEAPEST_ARC'. It solves the optimization problem using the OR-Tools library and returns the optimized route as an array of indices. This function is useful for setting up the optimization problem and solving it using the OR-Tools library.

Project ID Number: 3

5.5 Dataset filtering: Dealing with Outliers

There are some problems in the current dataset, such as unreasonable time interval between starting and ending times in one order record. To prevent outliers which have negative impact on the model accuracy, we establish new rules to filter the dataset.




(Figure 1: Outlier Detection Algorithm)

***K-Filtering****:* The first rule is identifying reasonable total transportation time. In the early time, we use Open Street Map (OSM) to calculate transportation time for each route, which is to provide a trustworthy reference time. Due to loading and unloading time and other factors, the exact time will be mostly longer or sometimes shorter within a range by observation. So we first set a factor k, which is the identifier of the ratio of dividing the actual time by the calculated time from OpenStreetMap, to eliminate rows with transportation time which are k-time larger than the reference time from OSM. Then, we set upper and lower boundaries of total transportation time to make sure the time is within a reasonable interval. During the data cleaning, due to the data quality, we believe that there involve some human error or procedure issue. Therefore, we believe that the ratio is in [0.5, 10], which means the actual finish time would fall between half to 10 times of the calculated time from the OpenStreetMap, which is by common sense a very large tolerance level. At the same time, by using the K-Filtering, most of the data are kept and do not incur many data loss problem and we believe the some extreme outliers are excluded in this stage.

**Outlier Detector**: We have tried three different methods are used for outlier detection, including the IQR method, the modified Z-score method, and the Local Outlier Factor (LOF) algorithm. Each method is implemented as a separate method with its own set of parameters. And select the one with the best result. However, the result from these three methods are more or less the same.

The IQR method is based on the interquartile range, which is the difference between the first quartile (Q1) and the third quartile (Q3). The method defines a threshold for outlier detection and

removes any data points that fall outside the range of Q1 - thresholdIQR to Q3 + thresholdIQR. The threshold value is set to 1 by default, but it can be adjusted as needed.

The modified Z-score method is based on the median absolute deviation (MAD) and is designed to be more robust than the standard Z-score method. It calculates the modified Z-score for each data point and removes any points that have a modified Z-score greater than a specified threshold. The threshold value is set to 3.5 by default, but it can be adjusted as needed.

The LOF algorithm is a machine learning algorithm that is designed to detect outliers in high-dimensional datasets. It works by calculating the local density of each data point and comparing it to the densities of its neighbors. Any data points that have a significantly lower density than their neighbors are considered outliers. The algorithm requires two parameters: the number of neighbors to use for density estimation (n_neighbors) and the contamination parameter, which controls the proportion of outliers in the dataset. By default, the n_neighbors parameter is set to 20, and the contamination parameter is set to 0.01.

All three methods share a common parameter: the columns_to_clean parameter, which is a list of column names that should be evaluated for outliers. This parameter can be adjusted as needed to include or exclude specific columns from outlier detection.

The resulting DataFrame, intermediate, contains only the data points that are not identified as outliers by the selected outlier detection method. The shape of the DataFrame before and after outlier removal is printed, along with the first few rows of the cleaned DataFrame.

## 5.6 Model selection

The code contains several functions that are used for model training and evaluation. The first function, called 'train_rf_svd', trains a random forest regressor model with TruncatedSVD dimensionality reduction. The function first drops unnecessary columns from the DataFrame, splits the data into training and testing sets, preprocesses the data using StandardScaler and OneHotEncoder transformers, and performs TruncatedSVD to reduce the dimensionality of the data. It then defines a hyperparameter distribution, creates a randomized search object with cross-validation, fits the search object to the training data, selects the best hyperparameters, and trains a random forest model with the best hyperparameters and the reduced data using cross-validation. The function evaluates the model's performance on the testing set and returns the Mean Absolute Percentage Error (MAPE) and R-squared values. It also appends the predictions to the original DataFrame if specified and saves the trained model to a file.

The second function, 'train_rf', trains a random forest regressor model and evaluates its performance on a test set. It drops certain columns, prepares the data for training, and defines a hyperparameter distribution for RandomizedSearchCV. The function creates a random forest model with the best hyperparameters and saves the preprocessor and model using joblib. If 'have_model' is True, it loads the preprocessor and model and evaluates the predictions using MAE, MSE, R^2, and MAPE. If 'print_result' is True, the function prints the evaluation metrics. If 'append_results' is True and 'col_name' is not None, it appends the predictions to the input dataframe and returns the modified dataframe, otherwise, it returns a dictionary containing the evaluation metrics.

The third function, 'train_tsne', trains a random forest regressor model with TruncatedSVD on a given training set, evaluates the model's performance on a test set, and saves the model to a file. The function performs t-SNE to reduce the dimensionality of the data, defines a hyperparameter distribution for RandomizedSearchCV, and trains a random forest model with the best hyperparameters and the reduced data. The function evaluates the model's performance and calculates MAE, MSE, R^2, and MAPE. If 'print_result' is True, it prints the evaluation metrics and saves the model to a file using joblib. If 'append_results' is True, it appends the predictions to the input data frame and returns the modified data frame.

The fourth function, 'train_nn', trains a neural network model on a given data frame using the MLPRegressor from scikit-learn. The function also preprocesses the data by encoding categorical columns using binary encoding, standardizing the features, and converting datetime columns to string data type. The function evaluates the model's performance by calculating MAE, MSE, R^2, and MAPE on a test set and saves the model using joblib. If 'append_results' is True, the function appends the predictions to the input data frame and returns the modified data frame. If 'print_result' is True, the function prints the evaluation metrics. The function is flexible and can be customized by providing options such as 'have_model' to load a pre-trained model.

To split the training and test data sets, the code calculates a 'cutoff_date' using the minimum and maximum dates in the "planStartTime" column of a given ratio. It then creates two new data frames, 'df_before', and 'df_after', by filtering the original data frame based on whether the

"planStartTime" values are before or after the cutoff_date. The 'df_before' will be used as the training set, and the 'df_after' will be used as the testing set. Different ratios will be tested, and they will drive different results in the model training part.

A series of model training and evaluation tasks are performed using the 'train_rf', 'train_tsne', 'train_rf_svd', and 'train_nn' functions on a given training set. The code appends the evaluation metrics of each model to a 'result_df' data frame using the 'ignore_index=True' option. The data frame is then sorted in ascending order of MAPE. Finally, the resulting 'result_df' data frame is printed to display the evaluation metrics of each model, and we can make a clear comparison for different machine learning models.

The best machine learning model is selected from a previously trained set of models stored in 'result_df'. A dictionary is defined to map each model to a lambda function that trains and evaluates the model on the training set. Then the selected model is used to make predictions on the testing set, with the option to append the results to the input data frame. Finally, we make predictions on the entire dataset and append the results to the original data frame.

After performing the model training and evaluation tasks, we found that the random forest models had the best results, with the largest R2 and lowest MAE and MAPE among all models. Therefore, we selected the random forest model as our best model and saved it for future use.

To predict the remaining dataset, we used the 'cutoff_date' to split the data into a training set and a testing set. We trained the random forest model on the training set and evaluated its performance on the testing set. Then, we saved the model and loaded it to make predictions on the remaining dataset. Our approach is designed to be automated, as the best model for each region may not be a random forest model, and the program could automatically switch to another model based on the performance metrics.

## 6. Result

### 6.1 ETA

The performance of the ETA (estimated time of arrival) model was evaluated using various metrics, including MAE, MSE, R2, and MAPE. The results showed that the accuracy of the ETA model was not very high. This may be due to several factors, such as the complexity of the traffic patterns, unforeseen events such as accidents or road closures, or errors in data collection.

Despite the less accurate results, the ETA model still provides valuable information for route planning and travel time estimation. It can help drivers make informed decisions about the best route to take and when to leave, potentially reducing the time and cost of travel. Additionally, the ETA model can be used to monitor traffic patterns and identify areas where improvements can be made, such as adding additional lanes or implementing traffic calming measures.

To improve the accuracy of the ETA model, several strategies can be used. One approach is to incorporate real-time data, such as traffic flow and incident reports, to update the model in real-time. Another strategy is to use machine learning techniques to better account for complex traffic patterns and identify patterns that may not be immediately apparent. Additionally, improving the quality and accuracy of the data used in the model can help to reduce errors and improve accuracy.

Overall, while the ETA model may not be highly accurate, it still provides valuable information for travel planning and can be used to identify areas for improvement in traffic management. By incorporating real-time data and using advanced machine learning techniques, the accuracy of the ETA model can be improved, providing even greater value in the future.

Project ID Number: 3

## 6.2 Road map

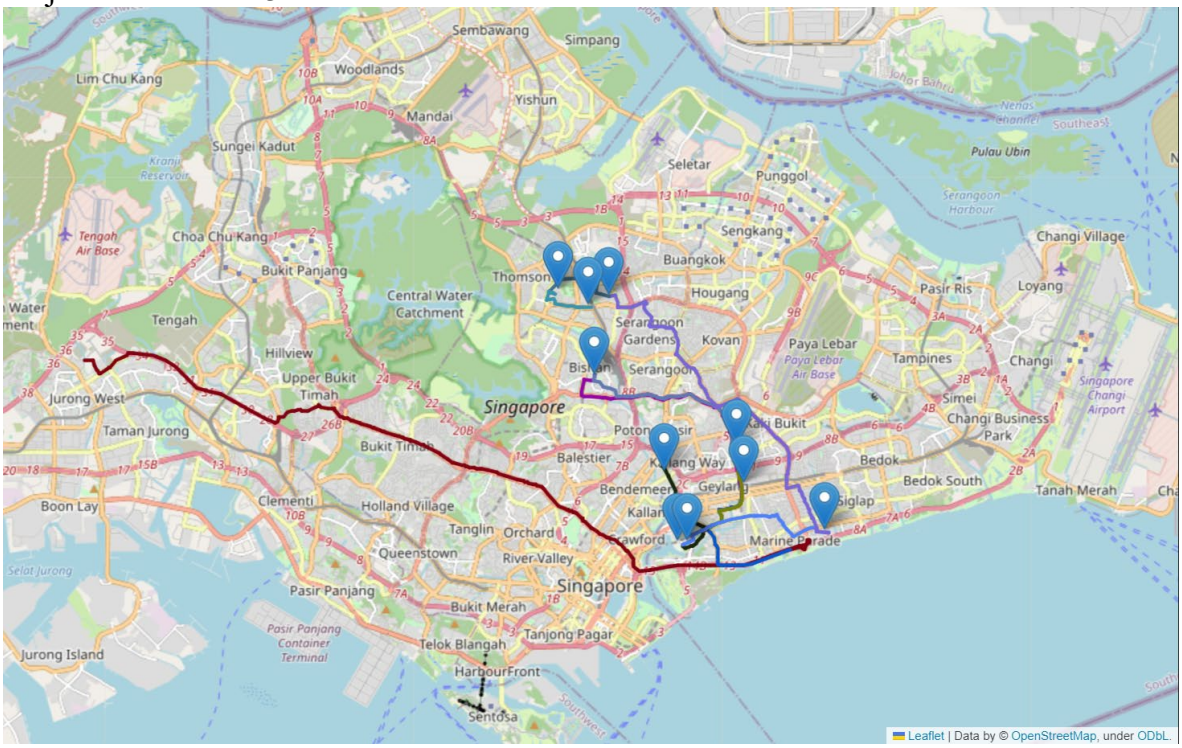
After calculating the estimated time of arrival for each shipment, we meticulously generated the optimal routes and seamlessly integrated them into the map for enhanced visualization. We compute the optimal route as well as their orders and the maximum distances. The following graph illustrate one example of our calculated results for the road map session. Utilizing OpenStreetMap as one of the additional data sources for this project, we will obtain and preprocess map data to extract pertinent features, including road types, speed limits, and traffic signals. This comprehensive approach ensures that our map provides users with an intuitive and user-friendly experience, allowing them to easily navigate and understand the optimal routes for their shipments.

```
Route for vehicle 0:
 0 ->  2 ->  4 ->  8 ->  9 ->  11 ->  1 ->  12 ->  10 ->  3 ->  7 ->  5 ->  6 -> 0
Distance of the route: 169m

Maximum of the route distances: 169m
1 0 (22.45055, 114.01209) (22.39502, 113.97302) 8.78 11.666666666666666
2 1 (22.39502, 113.97302) (22.20682, 114.02839) 31.69 43.68333333333333
3 2 (22.20682, 114.02839) (22.37382, 114.11763) 37.57 44.61666666666667
4 3 (22.37382, 114.11763) (22.35673, 114.12773) 3.121 4.95
5 4 (22.35673, 114.12773) (22.3136, 114.17052) 7.88 9.366666666666667
6 5 (22.3136, 114.17052) (22.31245, 114.22519) 7.64 10.133333333333333
7 6 (22.31245, 114.22519) (22.3077, 114.25998) 5.88 6.25
8 7 (22.3077, 114.25998) (22.38362, 114.27111) 11.022 18.55
9 8 (22.38362, 114.27111) (22.38215, 114.18678) 17.73 19.4
10 9 (22.38215, 114.18678) (22.44486, 114.17033) 15.561 13.8
11 10 (22.44486, 114.17033) (22.50131, 114.12773) 10.758 13.3
12 11 (22.50131, 114.12773) (22.44591, 114.03469) 14.89 24.35
13 12 (22.44591, 114.03469) (22.45055, 114.01209) 4.293 8.583333333333334
176.815 km
```

(Figure 2: Code for generating routes)

Project ID Number: 3



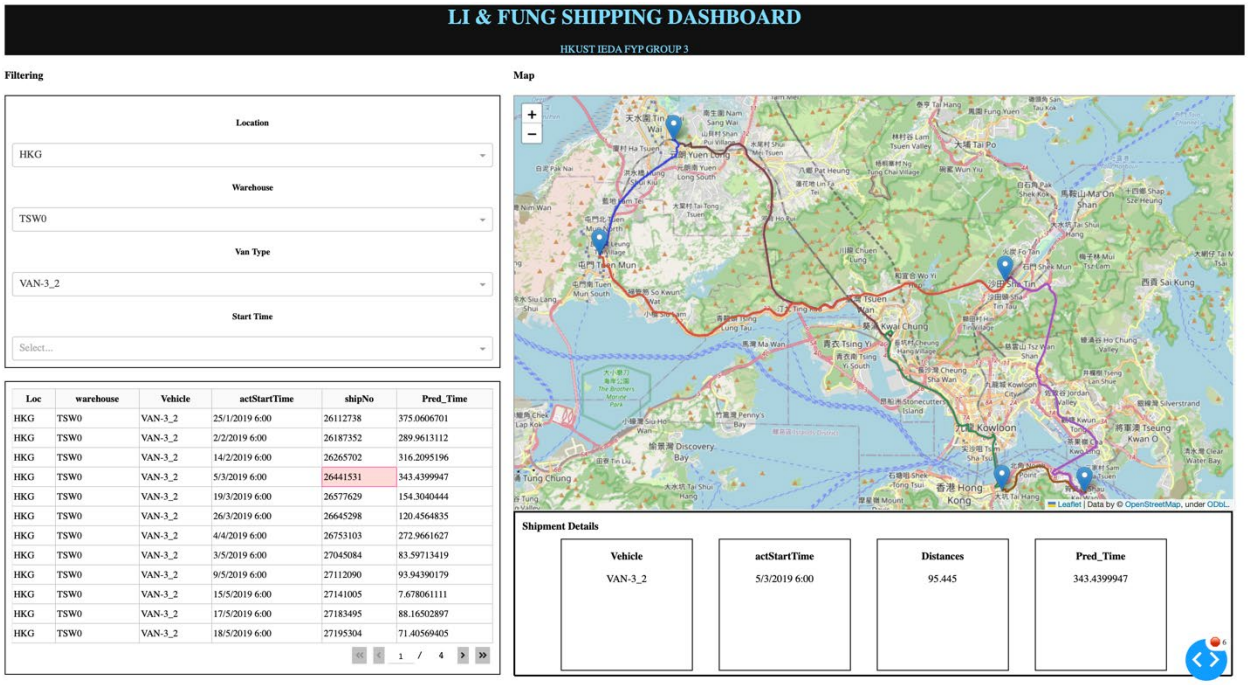

(Figure 3: Demonstration of printing rote on map)

We have opted to use OpenStreetMap over Google Maps for our project due to its open-source nature and the flexibility it offers in customizing map data. This choice allows us to tailor the map to our specific needs and requirements, ensuring a more personalized and efficient solution. Additionally, the open-source aspect of OpenStreetMap fosters a collaborative environment, enabling continuous improvements and updates to the map data, ultimately benefiting our project and its users in the long run.

Project ID Number: 3

## 6.3 Dashboard

We choose to use Dash to display our results and provide a comprehensive overview of each shipment's details and its ETA. Dash is a web application framework for Python that allows developers to create interactive visualizations. It is built on top of Flask, Plotly.js, and React.js, providing us with the ability to create complex dashboards with various interactive components such as tables, graphs, and maps. Since we needed to display our results in tables and also on a map with embedded routes, we found Dash to be a suitable tool for better visualize the ending results of our project.




| Loc | warehouse | Vehicle | actStartTime | shipNo | Pred_Time |
|---|---|---|---|---|---|
| HKG | TSW0 | VAN-3_2 | 25/1/2019 6:00 | 26112738 | 375.0606701 |
| HKG | TSW0 | VAN-3_2 | 2/2/2019 6:00 | 26187352 | 289.9613112 |
| HKG | TSW0 | VAN-3_2 | 14/2/2019 6:00 | 26265702 | 316.2095196 |
| HKG | TSW0 | VAN-3_2 | 5/3/2019 6:00 | 26441531 | 343.4399947 |
| HKG | TSW0 | VAN-3_2 | 19/3/2019 6:00 | 26577629 | 154.3040444 |
| HKG | TSW0 | VAN-3_2 | 26/3/2019 6:00 | 26645298 | 120.4564835 |
| HKG | TSW0 | VAN-3_2 | 4/4/2019 6:00 | 26753103 | 272.9661627 |
| HKG | TSW0 | VAN-3_2 | 3/5/2019 6:00 | 27045084 | 83.59713419 |
| HKG | TSW0 | VAN-3_2 | 9/5/2019 6:00 | 27112090 | 93.94390179 |
| HKG | TSW0 | VAN-3_2 | 15/5/2019 6:00 | 27141005 | 7.678061111 |
| HKG | TSW0 | VAN-3_2 | 17/5/2019 6:00 | 27183495 | 88.16502897 |
| HKG | TSW0 | VAN-3_2 | 18/5/2019 6:00 | 27195304 | 71.40569405 |

(Figure 4: Dashboard)

The graph above gives a demonstration of our sample dashboard, which includes four main sessions, the filtering, the table, the shipment detail boxes and finally the map with routes embedded. When the user first entering this dashboard, they may select the filters from the top left hand conner, they may choose to filter only one element, or they can filter them all with the limit up to four. After that a detailed table with a list of information form the filters will be shown. The users may use the page turner to see all the outcomes, and if they are particularly interested in one row, they may click on that specific row and the corresponding shipment details will pop up on the right bottom of the page inside the four boxes as shown. Additionally, the map with detailed routes of that selected shipment will also be shown.

One of the benefits of using Dash is that it provides a simple and intuitive user interface that can be easily customized to suit the needs of the end-user. With the filtering functions we added, the user can select different parameters such as shipment location, warehouse, van type, and start time to view the estimated time of arrival of the shipments that they are particularly concerned

about. The ability to customize the interface allows the end-user to tailor the dashboard to their specific needs and make informed decisions in a quicker manner based on the visualizations presented by the dashboard we provided.

Furthermore, since Dash is a web application framework, it enables the results to be accessed remotely, making it possible for different stakeholders to easily view the data and make their decisions. This feature is particularly useful for companies like Li & Fung, who operate globally and need to keep track of their shipments' ETA in different parts of the world. Dash's ability to present data in a visual and interactive format can help stakeholders make informed decisions, identify areas for improvement, and ultimately improve the delivery process.

Moving on to the dash creation process, we first brainstormed the initial layout of the board by standing at the user's perspective. We try to present what is essential in a shipment and include the details which the users may expected to see when making their decisions. Then, after attaining the estimated time of arrival results for our project by detailed calculation and training processes, we use the file generated and created the dashboard with the layout preferred that allowed users to filter shipment location, warehouse, van type, and start time to get the estimated time of arrival. Our dashboard also displayed the results on a map with embedded different routes to give a clearer view to the users how each shipment transport from the initial location to its end destination.

The use of Plotly Dash for our project was a good idea because it enabled us to create a user-friendly interface that allowed our stakeholders to easily access and interpret our data. The filtering functions we implemented in the dashboard made it easy for users to find specific information related to their shipments. The map visualization was also helpful because it allowed users to see the planned route for the shipment and any potential delays or diversions during the transition process because of traffic jam that may affect the estimated time of arrival. We believe the end-users of our dashboard will benefit greatly from its use. Our dashboard provides real-time data on estimated time of arrival, making it easier for users to plan their schedules accordingly.

## 7. Conclusion

### 7.1 Project review

To summarize, our project aims to address the challenges faced in predicting estimated time of arrival accurately, which can have a significant impact on logistics firms' performance and profitability, the missing dataset, and unmodified data collection process as well as the lack of consideration of critical variables that can impact the actual arrival time of shipments. Our proposed solution includes leveraging the advantages of OpenStreetMap for ETA forecasting to design and develop machine learning models using historical delivery data. We also suggest implementing appropriate data cleaning, imputation, or collection strategies to improve data quality and accuracy, integrating data on traffic and weather conditions, collecting, and analyzing data on different parcel types and their delivery times, and implementing a region-specific ETA prediction system.

### 7.2 Insights

The accuracy of ETA prediction can be significantly improved by considering several variables such as traffic, weather conditions, different parcel types, and infrastructure of each region. Furthermore, automated data collection methods such as RFID sensors or GPS devices that reduce human error can improve the dataset's quality and accuracy. Moreover, using statistical methods to impute missing values and removing incomplete data points can help improve the accuracy of the dataset. Finally, a region-specific ETA prediction system can be used to provide more accurate ETA predictions that consider unique road conditions, traffic patterns, and infrastructure of each area.

### 7.3 Improvement

To integrate the system for data cleansing across different regions, we can develop a standardized data cleaning pipeline that can be customized based on the unique data characteristics of each region. This pipeline can include steps such as missing value imputation, outlier detection and removal, and feature scaling. The pipeline can be designed to automatically apply appropriate data cleaning techniques based on the data types and distribution of each region's data.

To prevent over-fitting when establishing machine learning models, we can use techniques such as cross-validation and regularization. Cross-validation involves dividing the data into training and validation sets and using multiple iterations to evaluate the model's performance on different subsets of the data. Regularization involves adding a penalty term to the loss function to discourage the model from fitting the training data too closely. We can also use techniques such as early stopping and ensemble learning to prevent over-fitting. Early stopping involves stopping the training process once the model's performance on the validation set starts to deteriorate, while ensemble learning involves combining multiple models to reduce the risk of over-fitting.

By implementing these techniques, we can develop more robust and accurate machine learning models that can generalize well to new data.

For the dash visualization part, if possible, we could also add different views for user with different using purpose. For instance, we could generate a manager view option or worker view option to make the system more user-friendly and efficient. The manager view could include more detailed information and metrics, such as the number of shipments completed, the average time taken to complete one shipment, and the overall productivity of certain van type. On the other hand, the worker view could focus more on the individual tasks assigned to them, with clear instructions and deadlines. To ensure that the dash visualization is accessible to all users, we could also make it responsive and mobile-friendly, so that users can access it on-the-go. Finally, we could also consider integrating real-time updates and notifications, so that users can stay up to date with the latest information and changes in the system. Overall, by adding these features and options, we can make the dash visualization part more versatile, user-friendly, and efficient.

**8. Reference**

Bektas, T., Laporte, G., & Vigo, D. (2019). A review of machine learning in vehicle routing and scheduling. European Journal of Operational Research, 276(3), 795-811.

Goh, M., Lim, J., & Meng, Q. (2018). A decision support system for real-time logistics planning and scheduling in emergency response. Transportation Research Part E: Logistics and Transportation Review, 110, 1-18.

Han, Z., Li, Y., & Cao, Z. (2019). A machine learning approach for real-time ETA prediction in urban logistics. IEEE Access, 7, 107786-107793.

Kwon, K., Park, K., & Bae, H. (2021). Review and comparison of prediction algorithms for the estimated time of arrival using geospatial transportation data. Journal of Marine Science and Engineering, 9(3), 322. https://doi.org/10.3390/jmse9030322

Luo, T., Wang, S., & Tian, Z. (2017). Predicting truck arrival times at ports using machine learning techniques. IEEE Transactions on Intelligent Transportation Systems, 18(7), 1908-1917.

Mohan, R., Vinodh, S., & Karthik, K. (2016). Logistics service quality and logistics performance: A conceptual framework for measuring logistics performance. International Journal of Logistics Systems and Management, 23(1), 1-20.

Park, K., Sim, S., & Bae, H. (2021). Vessel estimated time of arrival prediction system based on a path-finding algorithm. Journal of Marine Science and Engineering, 9(5), 468. https://doi.org/10.3390/jmse9050468

Zhang, W., Liu, X., & Li, Z. (2019). A deep learning approach to predicting delivery times in urban logistics. IEEE Transactions on Intelligent Transportation Systems, 21(9), 3938-3947.

Project ID Number: 3

## 9. Appendix

Project ID Number: 3

**10. Tools and technologies used**

| |
|---|
| OpenStreetMap |
| Google Ortool |
| Machine Learning Models by Sklearn |
| Tomtom Traffic Data |
| Weather Data by NECI |
| Dash |