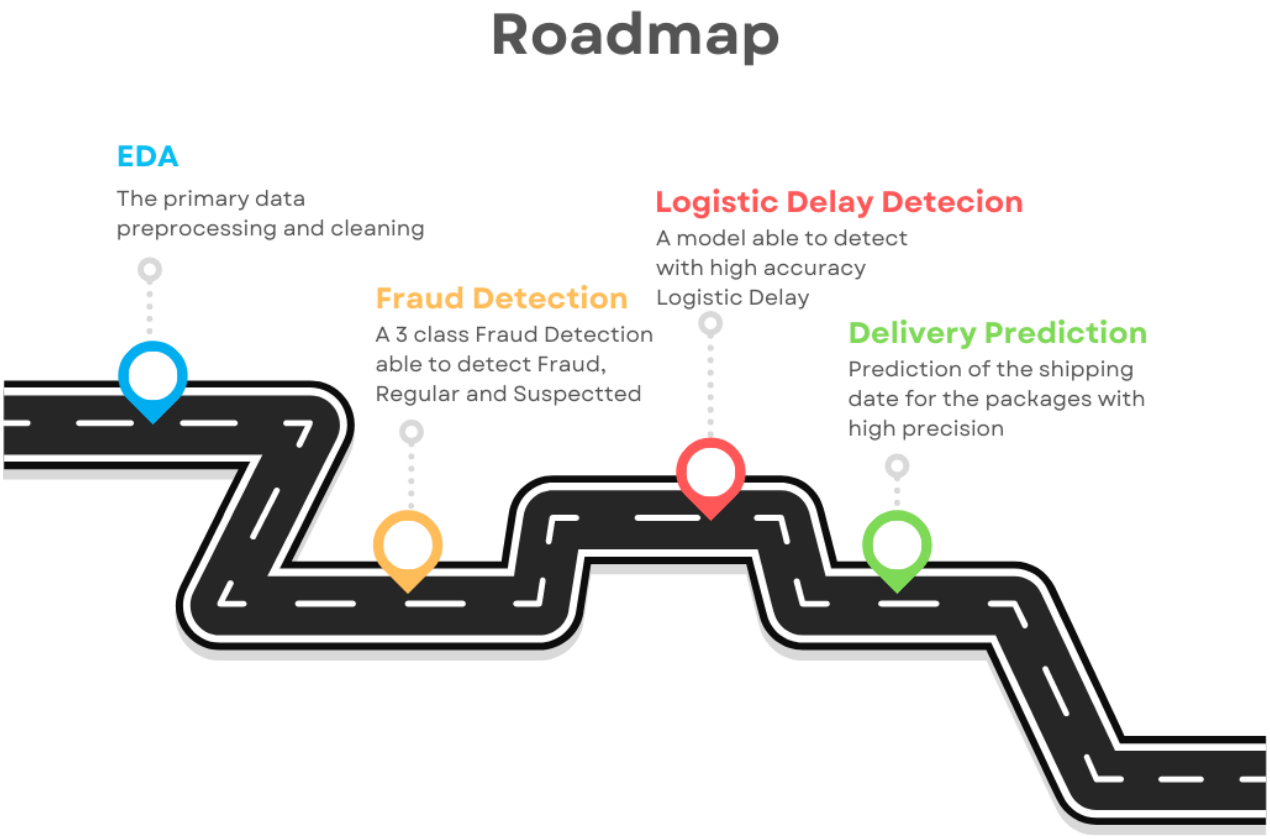
Enhancing Supply Chain Resilience: Fraud Detection and Delivery Prediction

Daniele Fiorucci – daniele.fiorucci@studenti.luiss.it – matr. 761461
Giacomo Rossi – g.rossi1803@studenti.luiss.it – matr. 763391
Costanza Placanica – costanza.placanica@studenti.luiss.it – matr. 759641

1. Introduction

1.1 Roadmap

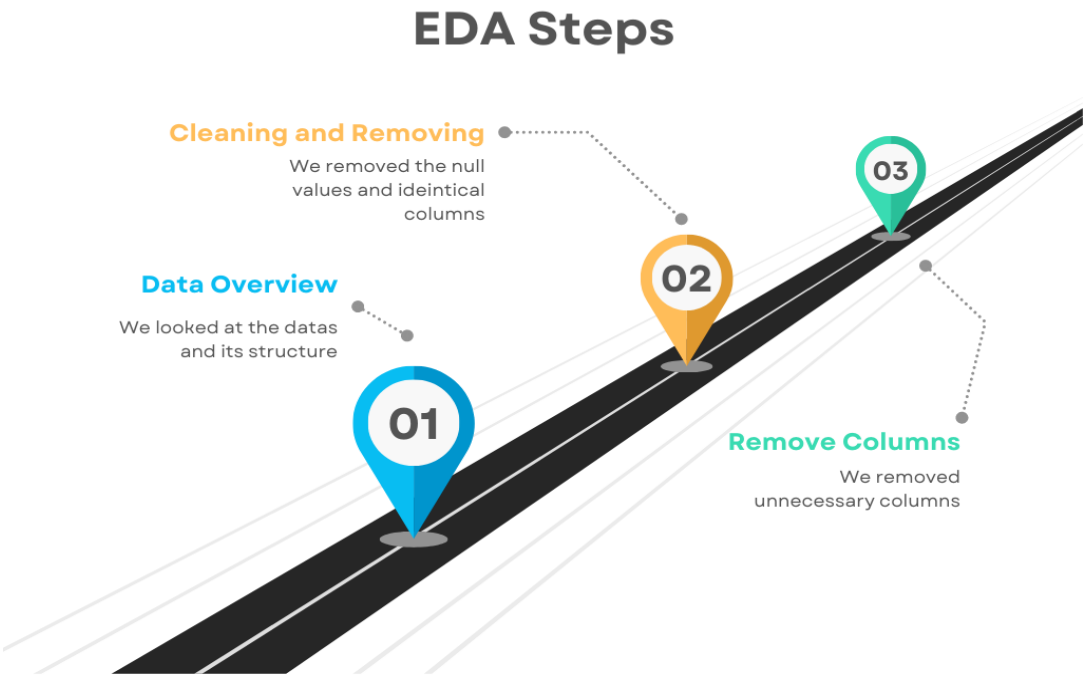


Artificial intelligence is well-suited for optimizing business problems, particularly in the field of supply chain management, where machine learning algorithms can detect fraudulent operations and predict delivery times. Our advanced models have been trained on a vast amount of relevant data and can analyze patterns and trends to provide valuable insights to supply chain professionals. By leveraging our machine learning capabilities, we want to optimize this process to improve performances and minimizing the losses.

We started by cleaning the data we were given so that it would be in the right format for our models. The tasks we had to solve are the following:

1. Fraud Detection: we had to solve a multiclass fraud detection problem.
2. Logistic Delay Detection: we had to predict a possible delay in the logistic.
3. Delivery Prediction: we had to accurately predict the shipping days of the packages for our customers.

1.2 Exploratory Data Analysis

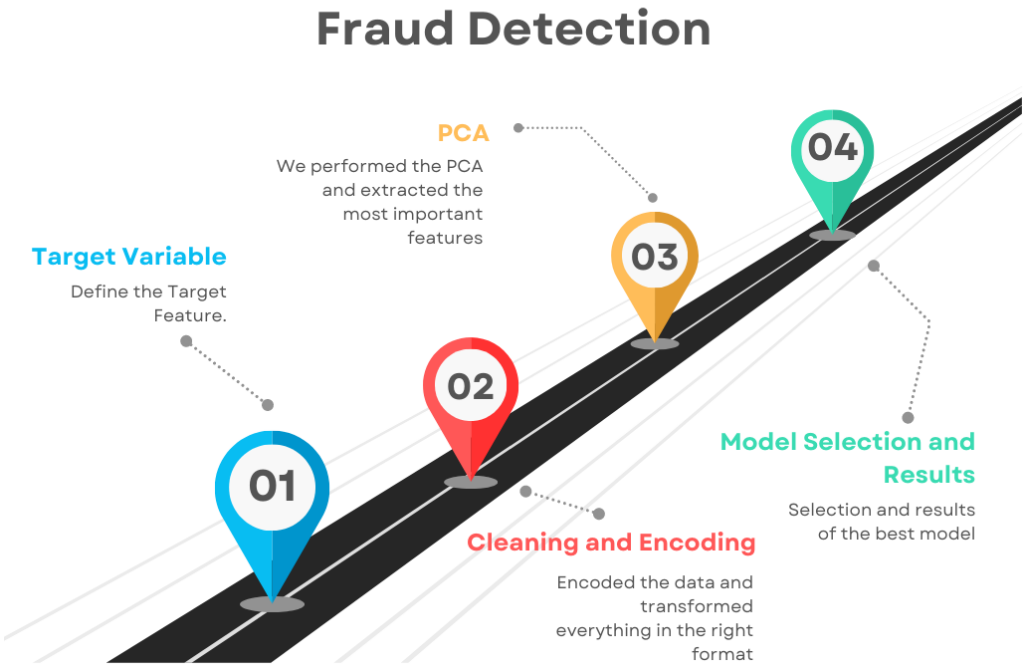


The data we were given has 180519 rows and 53 features. These are divided into 29 numeric variables and 24 categorical variables.
We started cleaning the data by removing null values, since we had very few of them, and by removing Product Description and Order Zipcode, as they contained a high number of null values, together with Customer Password and Customer Email that, for privacy reasons, are filled with XXXXX.
We proceeded by looking for any duplicated columns. The columns Benefit per order and Order Profit Per Order, Sales per customer and Order Item Total, Category Id and Product Category Id, Customer Id and Order Customer Id, Order Customer Id and Customer Id, Order Item Cardprod Id and Product Card Id, Order Item Product Price and Product Price were respectively duplicated. Therefore, we decided to remove one for each duplicate.
We removed all the columns containing ID.
We are left with the following variables in our dataframe:

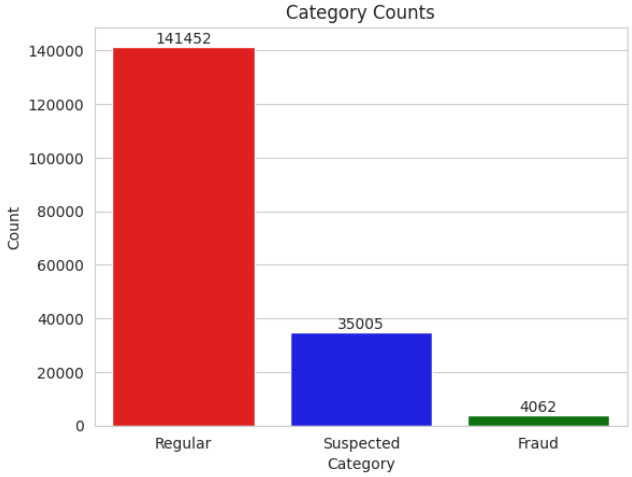
Type, Days for shipping (real), Days for shipment (scheduled), Benefit per order, Sales per customer, Delivery Status, Late_delivery_risk, Customer City Customer Segment', 'Department Name', 'Latitude', 'Longitude', 'Market', 'Order City', 'order date (DateOrders)', 'Order Item Discount', 'Order Item Discount Rate', 'Order Item Product Price', 'Order Item Profit Ratio', 'Order Item Quantity', 'Sales', 'Order Status', 'Product Price', 'Product Status', 'shipping date (DateOrders)', 'Shipping Mode'.

2. Methods

2.1 Fraud Detection



Moving to the fraud detection task, the first thing we had to do was to create our target variable. To do so we used the Order Status variable we had in our data frame.
We decided on three classes: Regular, Suspected and Fraud. All those packets that were classified as 'COMPLETE', 'PENDING', 'PENDING_PAYMENT' and 'PROCESSING' were grouped into ‘Regular’. Those classified as 'CLOSED', 'CANCELED', 'ON_HOLD' and ‘PAYMENT_REVIEW' were grouped into ‘Suspected’; those which were classified as 'SUSPECTED_FRAUD' we considered them as ‘Fraud’. Altogether, the resulting categories were Regular, Suspected and Fraud, with a distribution of 141452 Regular (78.4%), 35005 Suspected (19.4%) and 4062 Fraud (2.3%).



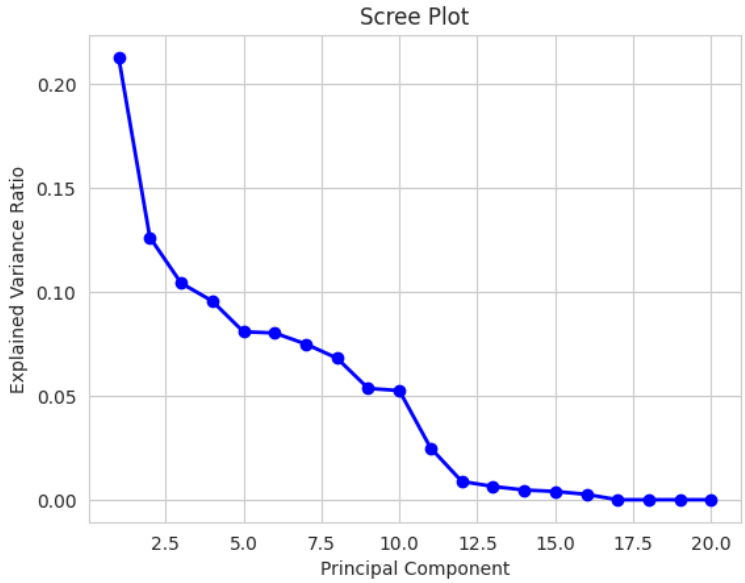
We then dropped the Order Status variable and moved to the encoding, as we had to transform the categorical variables to numerical.
Before doing this, we also needed to transform the Days for shipping (real), Days for shipment (scheduled) from DateTime objects to numbers. To achieve this, we used the timestamp technique, which returns the time elapsed from the epoch time which is set to 00:00:00 UTC for 1 January 1970.

After converting the dates, we can continue with the encoding.
The variables we had to encode were Shipping Mode, Delivery Status, Order City, Type, Customer Segment, department Name and Market.
For the first two we used a label encoder, which assigns a number to each category. Since this will apply a hierarchy, we used a custom order.

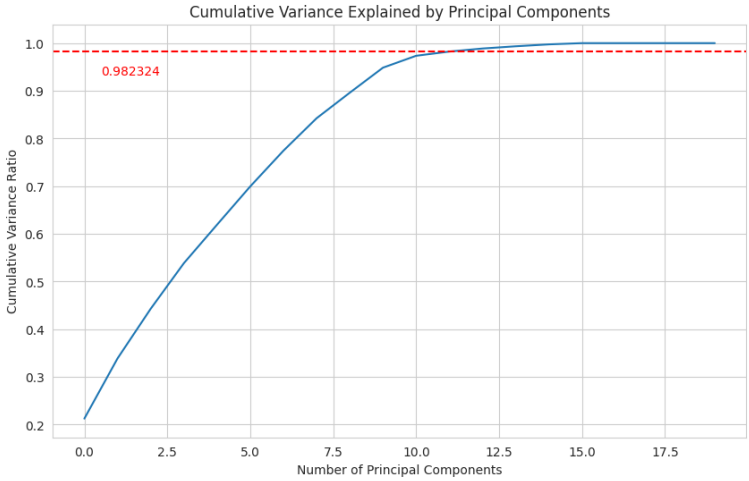
- Shipping Mode:
  - Same Day = 0,
  - First Class = 1,
  - Second Class = 2,
  - Standard Class = 3.
- Delivery Status:
  - Shipping on Time = 0,
  - Advanced Shipping = 1,
  - Late Delivery = 2,
  - Shipping Canceled = 3.

For Order City we used instead a Leave One Out Encoding technique since the number of categories was too high to perform a One Hot Encoding or to establish a hierarchy. In this Encoding, the value for the corresponding categorical feature is replaced by the average of the target variable for all other data points that have the same categorical value. This is done for each categorical value in the dataset.
For the other features listed above we used a One Hot Encoding. To encode a specific data point, the value for the corresponding categorical feature is set to 1 for the new feature that matches the categorical value, and 0 for all other new features.

After we cleaned and encoded the data, we are now ready for the next step, the PCA.
Principal Component Analysis (PCA) is a technique used in data analysis to reduce the dimensionality of a dataset while preserving the most important information. PCA analyzes the dataset to identify the directions (or axes) in which the data varies the most. These directions are then used to create the principal components, which are linear combinations of the original features.
Our goal was to select the most important features of the data for more business interpretability. We performed this analysis just on the variables that were not encoded using One Hot Encoding.
We were able to reduce by 45% our variables, keeping only 11 variables.



These 11 features explain 98% of the variance of our data.

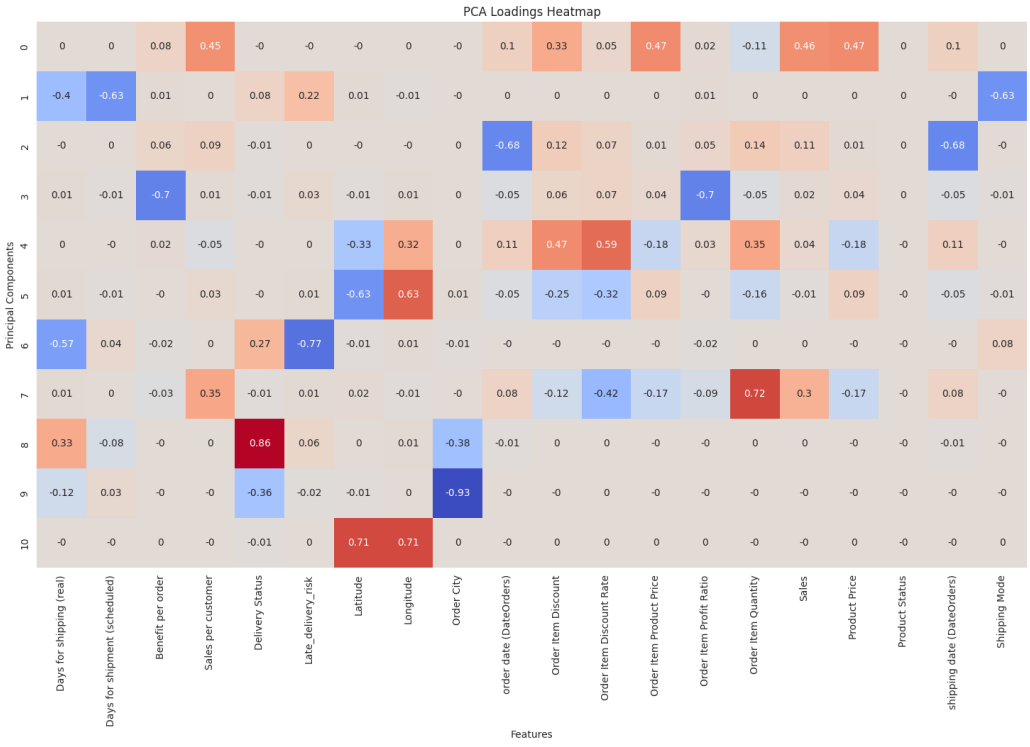


We then concatenated the encoded variables to these new variables.
We can also look at the loadings of the PCs. Loadings represent the correlation between the original features and the principal component. The magnitude of the loading indicates the strength of the relationship, while the sign indicates the direction of the relationship. A positive loading indicates a positive relationship between the original feature and the principal component, while a negative loading indicates a negative relationship. The results are the following:

- PC2:
  - Days for shipment (scheduled): -0.6272
  - Shipping Mode: -0.6274
- PC3:
  - order date (DateOrders): -0.6834
  - shipping date (DateOrders): -0.6834
- PC4:
  - Benefit per order: -0.6954

    - Order Item Profit Ratio: -0.7036
- PC6:
    - Latitude: -0.6271
    - Longitude: 0.6285
- PC7:
    - Late_delivery_risk: -0.7697
- PC8:
    - Order Item Quantity: 0.7162
- PC9:
    - Delivery Status: 0.8570
- PC10:
    - Order City: -0.9253
- PC11:
    - Latitude: 0.7070
    - Longitude: 0.7070

Here we selected just the loadings with absolute value greater the 0.6 but we can see all of them in the picture below.

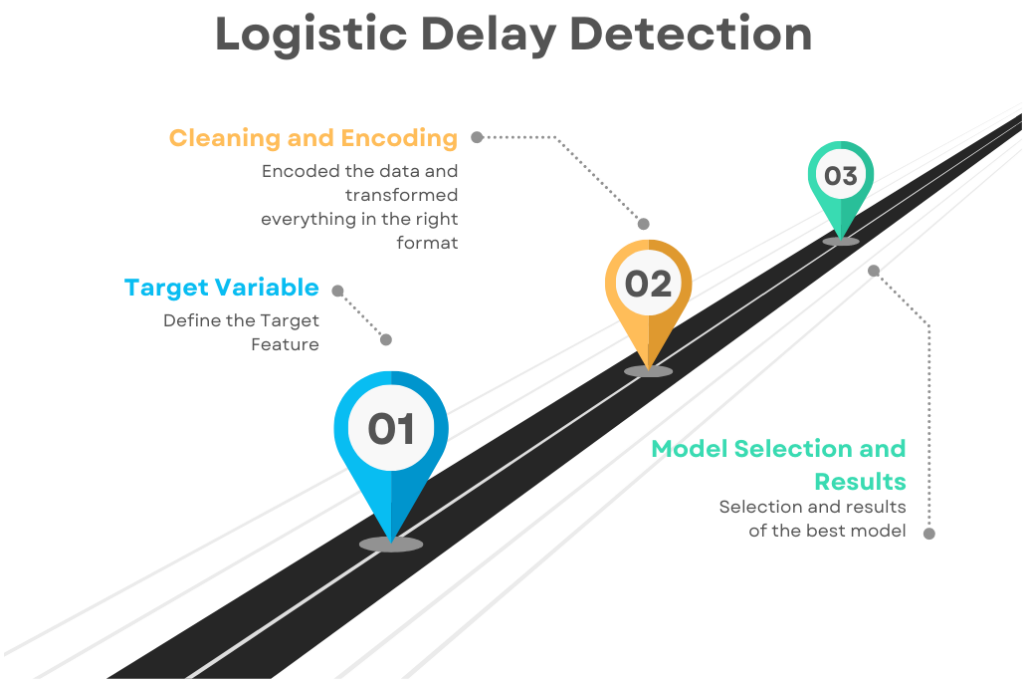


The data is almost ready for the model. The last step is just to scale the data.
We used StandardScaler which is a technique used to transform numerical data so that it has a mean of zero and a standard deviation of one.

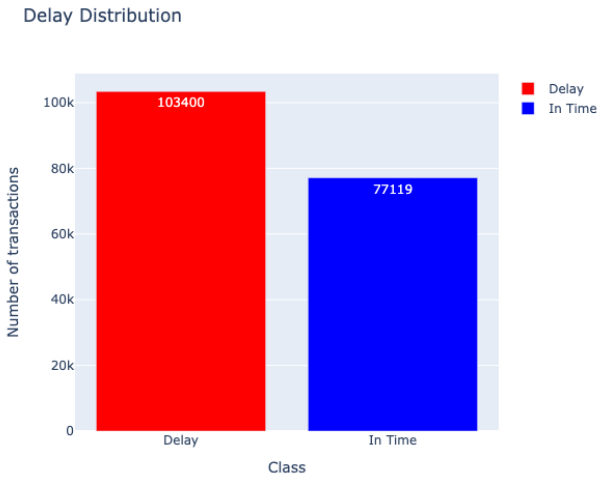
Now the data for the Fraud Detection is ready for the model fit.
To ensure the right proportion of each class in both the train and test set we specified the ‘stratify = y’ in our train_test split function.

We developed a Voting Classifier which puts together an XGBoost and a Logistic Regression model with a soft voting strategy. This particular classifier is able to provide accurate results for our fraud detection problem.

2.2 Logistic Delay Detection



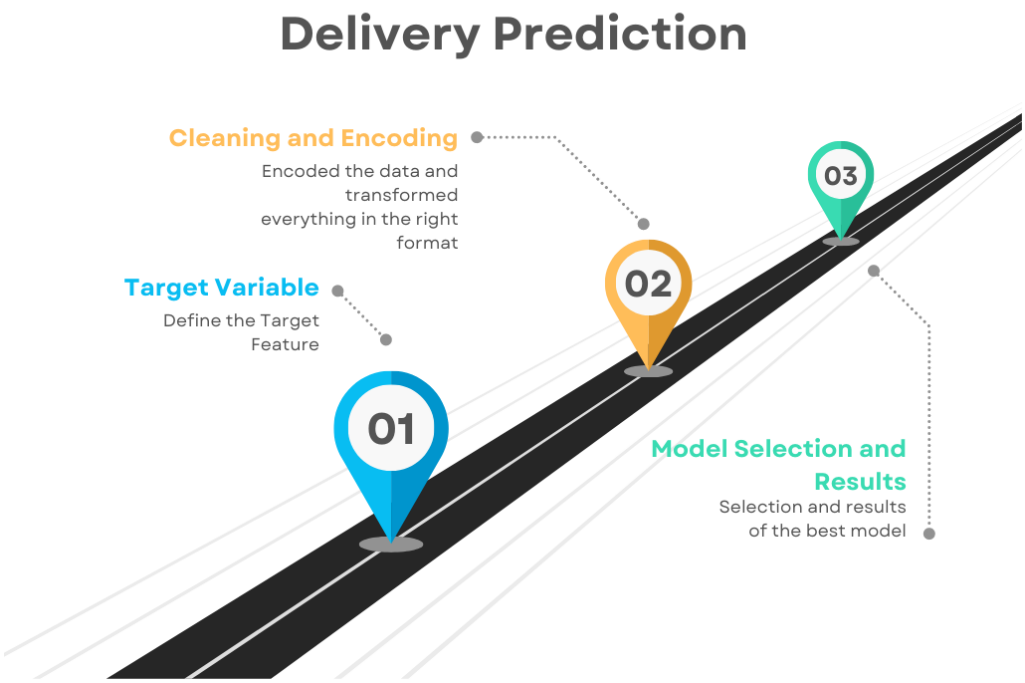
For what regards the Logistic delay detection task, we first created our target variable by subtracting the variable Days for shipment (scheduled) to the variable Days for shipping (real). The negative values representing the delay in logistic and the positive values the one in time. We assigned the delay to the number 1 and those in time to number 0.



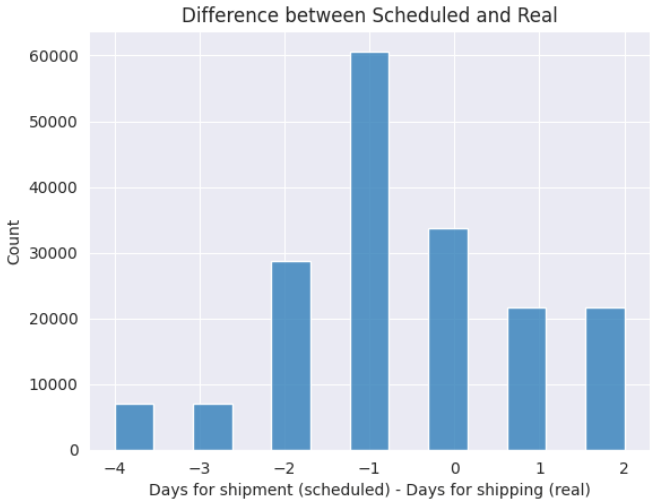
One important step we want to highlight is that we dropped the columns Delivery Status and Late_delivery_risk. We dropped them to avoid any possible data implication with the target variable, that could result in data leakage. Moreover, because we are not sure on how the variable Late_delivery_risk is measured, we wanted to avoid any possible bias that might come from using it in the model, especially because we realized that the models considered it to be highly significant for the final prediction.

For the data preprocessing and encoding we proceeded the same way as we did for the Fraud Detection.
We used the timestamp for the date columns, we encoded Customer City and Order City with the Leave One Out Encoding, we used the One Hot Encoding for the features Type, Category Name, Customer Segment, Department Name and Order Status.
We then encoded Shipping mode with the same method as before and we scaled the data with the StandardScaler.
The model we selected for this task is a Logistic Regression model.

2.3 Delivery Prediction



We started our task by looking at the previous system used to estimate the delivery days by looking at the following plot:



This plot shows that the old system was wrong most of the times.
Starting with the analysis we removed the Delivery Status and Late_delivery_risk for the same reason explain in the Logistic Delay Detection section.
The procedure for the data preprocessing and encoding is also the same as before.
We used the timestamp for the date columns, we encoded Customer City, Order City Order State and Order Region with the Leave One Out Encoding, we used the One Hot Encoding for the features Type, Category Name, Customer Segment, Department Name and Order Status.
We then encoded Shipping mode with the same method as before and we scaled the data with the StandardScaler.

We wanted to apply a more probabilistic approach, and one model that does so is the Bayesian Ridge Regressor.
Bayesian Ridge Regression is a type of regression analysis that combines the principles of Bayesian inference and ridge regression. In Bayesian Ridge Regression, prior information is used to specify a prior distribution for the regression parameters.
The ridge regression part of Bayesian Ridge Regression involves adding a penalty term to the likelihood function. This penalty term helps to reduce the variance of the estimates by shrinking the regression coefficients towards zero. This can help to prevent overfitting and improve the predictive accuracy of the model.
Overall, Bayesian Ridge Regression provides a flexible and powerful framework for modeling complex relationships between variables, while also incorporating prior information and addressing issues like overfitting.

3. Experimental Design

3.1 Fraud Detection

3.1.1 Model selection

Purpose

The purpose of this experiment is to compare the performance of different machine learning algorithms for fraud detection. We compare the results of using a logistic regression model with a Random Forest, an XGBoost, a LGBM and a Decision Tree. All this to come up with the final model.

Baseline

We choose a Logistic Regression model with the only parameter being class_weights = 'balanced'. We used this model as a baseline for its simplicity, interpretability and due to its expedient training capability.

Evaluation Metrics

In a fraud detection task, identifying true positives (i.e., detecting actual fraud cases) is crucial, even if it means potentially sacrificing some true negatives (i.e., classifying non-fraud cases correctly). Therefore, the recall metric, also known as sensitivity or true positive rate, is often prioritized over other metrics in such tasks.

Model Comparison

Starting by fitting the baseline model, the results were actually very good.
We obtained a **Fraud Recall** of 0.99, a **Regular Recall** of 0.86 and a **Suspected Recall** of 0.64.
We then tried the other models, the Random Forest, the XGBoost, the LGBM and the Decision Tree.
The results were the following:

- Fraud Recall: **rf** = 1, **xgb** = 0.68, **lgbm** = 0.67, **dt** = 0.59
- Regular Recall: **rf** = 0.94, **xgb** = 0.99, **lgbm** = 0.99, **dt** = 0.94
- Suspected Recall: **rf** = 0.55, **xgb** = 0.64, **lgbm** = 0.65, **dt** = 0.71

There was not a model that had good results for all the 3 classes, the Random Forest was perfect for the Fraud and Regular but not for the Suspected, the XGB, LGBM and Decision Tree were perfect for the Regular and better for the Suspected but awful for the Frauds and the Logistic Regression is still the best model.

We then decided to combine the previous models.
The resulting best model we selected is a Voting Classifier which combines an XGBoost with no parameters and the Logistic Regression used above.
Compared to our baseline model, this one has the same Fraud Recall while the Regular Recall has increased by 0.13 but the Suspected Recall has decreased by 0.04.
We were able to get a more balanced performance across all three classes by combining these two models into a Voting Classifier and leveraging the characteristics of each model. This strategy helped

us to retain a strong Fraud recall while simultaneously boosting Regular recall. Although the Suspected recall declined marginally, the voting classifier's overall performance improved significantly over the baseline model.

This lower recall occurred for two main reasons. The first is that we built the three classes from what was in Order Status based on our assumptions about the definition of each class. The second is that, unlike the other two classes, Fraud and Regular, the Suspected has a less clear-cut boundary, implying that the model also made a mistake in applying a more defined distinction.

3.2 Delivery Prediction

3.2.1 Model selection

Purpose

The purpose of this experiment is to compare the performance of different machine learning algorithms for delivery prediction finding the right tradeoff between the statistics and business interpretation. We compare the results of using a random forest model as the baseline model, followed by a lasso model, a ridge model, and a Bayesian ridge model. The aim is to determine the best performing algorithm for this task and to use it as the final model for delivery prediction.

Baseline

We choose a Random Forest Regressor with no parameter specified.
We chose it as the baseline model because it is a popular and versatile algorithm that can handle complex and non-linear relationships between features and the target variable. It is also relatively easy to implement.

Evaluation Metrics

In this experiment, we used two main evaluation metrics to assess the performances: rMSE and R2.

rMSE measures the difference between the predicted and actual delivery times in units of time.
R2 measures the proportion of the variance in the target variable that is explained by the model.
Additionally, we also computed the average prediction by day, which gives an idea of the overall accuracy of the model in predicting delivery times. This metric can be useful for understanding the practical implications of the model's performance, as it reflects the average prediction error for each day.

Model Comparison

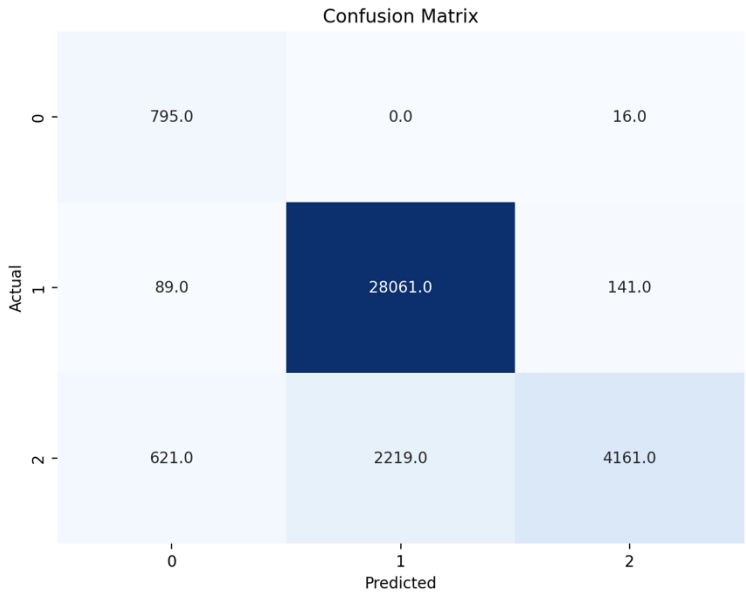
We started our model selection with a Random Forest Regressor. The results were not very good. The rMSE was around 1.36 and the R2 of 0.30.
To improve our results, we thought about introducing some kind of regularization.
We started by introducing some L1 regularization by fitting a Lasso Regression.
The results were worse, since the rMSE turned out to be 1.62 and the R2 very close to 0.
We understood that the L1 was not the right path, so we implemented a Ridge Regression to use the L2 regularization.
We got a 0.58 for the rMSE and a 0.87 for the R2.

These results started to be interesting.
We wanted to apply a more probabilistic approach and keep the L2 regularization term, so we decided to try a Bayesian Regression Model

4. Results

4.1 Fraud Detection

We can see here the Confusion Matrix of the final model:



Here we reported the table with all the results for every model we tried:

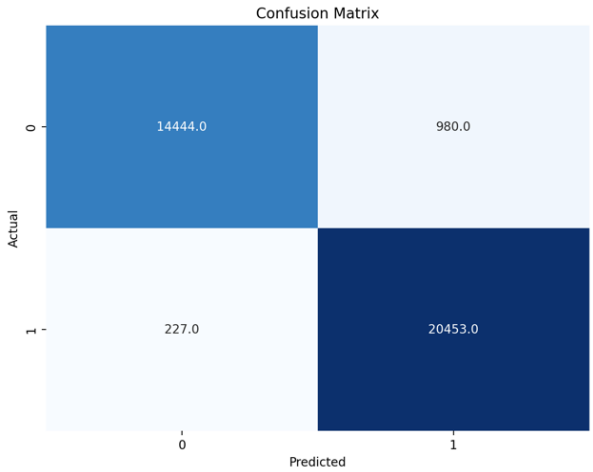
Model	***Parameters***	***Fraud, Regular, Suspected***
Logistic Regression	class_weights = 'balanced', max_iter = 1000	0.9963, 0.8702, 0.6387
Random Forest	class_weights = 'balanced', max_depth = 4	0.9987, 0.9470, 0.5562
XGBoost		0.6798, 0.9984, 0.6399
LGBM	max_depth=4, n_estimators=1000	0.6637, 0.9982, 0.6481
Decision Tree	class_weights = 'balanced'	0.5886, 0.9393, 0.7130
Voting Classifier	Logistic Regression, XGBoost	0.9729, 0.9942, 0.6003

4.2 Logistic Delay Detection

We did some hyperparameter tuning and the best parameters we found are the following:
C = 10, solver = 'lbfgs', max_iter = 1000.
With these parameters we achieved the following results.

- Accuracy = 0.9678
- Recall = 0.9889
- Precision = 0.9564
- F1 = 0.9723

With the following Confusion Matrix:

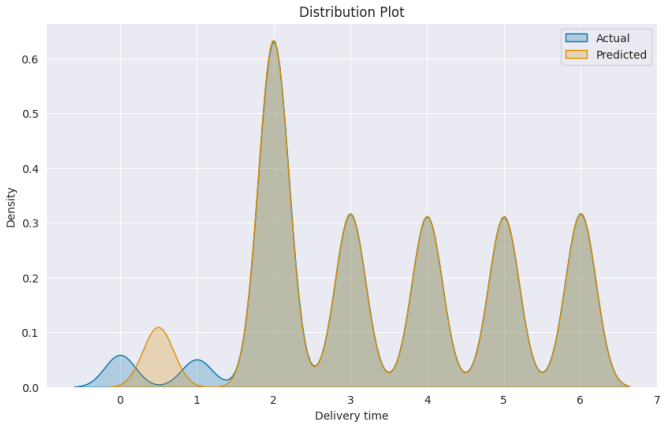


4.3 Delivery Prediction

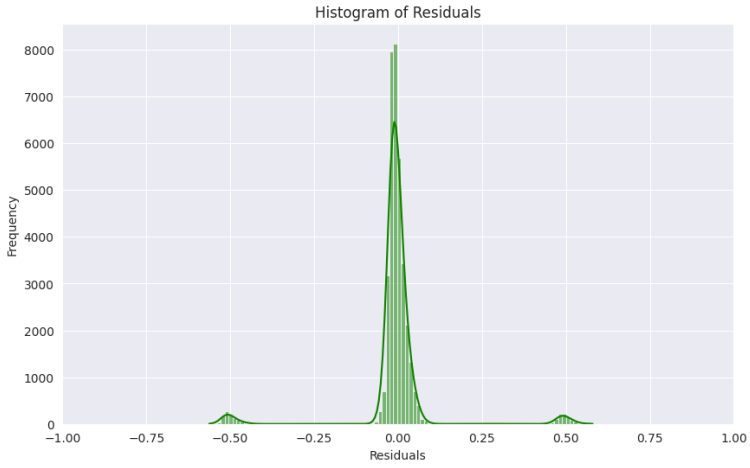
If we look at the results of this model, we see a huge improvement.
The rMSE drops down to 0,117 and the R2 reaches 0.995.
In addition to these metrics, we can look deeper to the average predicted values for each day.

Day	*Instances*	*rMSE*	*Avg. Predicted Value*
0	1027	0.5765	0.4945
1	910	0.4345	0.4958
2	11300	0.0899	2.0174
3	5715	0.0764	3.0300
4	5664	0.0722	3.9842
5	5680	0.0923	4.9417
6	5808	0.1272	5.8955

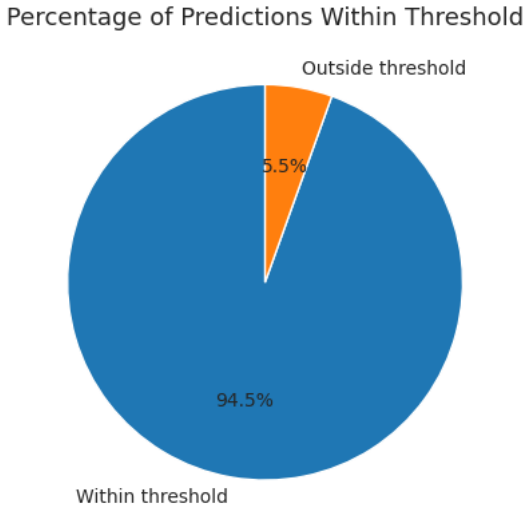
We can see this also in the following distribution plot.



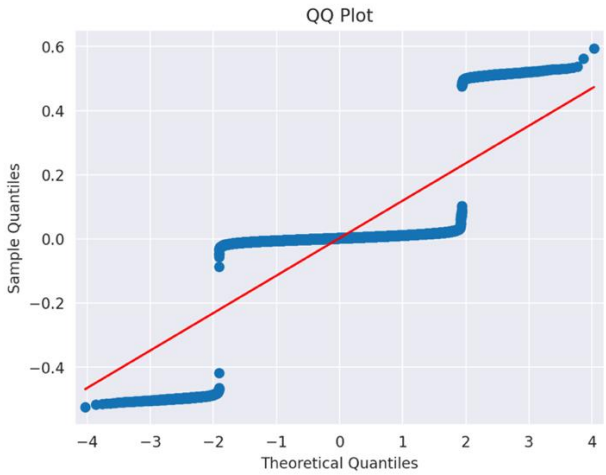
The first thing we notice is that the model is not able to predict day 0 and 1. This is because the instances are less that the other days and because they are closer in time.
Looking at the residuals below we see they are almost all very close to zero except for some bins around +- 0.5.



We can also add the percentage of prediction that are within a specific threshold.
Choosing a threshold of 0.1 days (corresponding to 2 hours and 24 minutes) this is the result.
Almost 95% of our predictions are within 0.1 days.



Finally, we have the QQ plot.



We see it is pretty good apart from the tail where it moves from the line. This is due to the residuals around +- 0.5.

Here we list all the models used with the corresponding results.

Model	*rMSE*	*R2*
Random Forest	1.3583	0.3040
Lasso(alpha=0.5)	1.3516	0.3102
Ridge	0.5775	0.8740
Bayesian Ridge(compute_score=True)	0.1157	0.9949

5. Conclusions

5.1 Fraud Detection

Based on the evaluation metrics and model comparison, the voting classifier model is the best performing model for this fraud detection task. It achieves a perfect Fraud Recall of 0.99, an improved Regular Recall of 0.99 and a comparable Suspected Recall of 0.60 relative to the baseline Logistic Regression model.

With a Fraud Recall of 0.99, the model is highly effective in detecting actual fraud cases. This will allow the company to minimize fraud losses by flagging and investigating nearly all fraudulent transactions.

The high Regular Recall of 0.99 indicates that the model is accurately classifying most legitimate transactions. This will reduce the number of false positives and ensure a good customer experience with minimal disruption.

Although the Suspected Recall is slightly lower at 0.60, it is still reasonably high given the assumptions and the interpretation of this class as mentioned above. The model can recover more than half of the suspected fraud cases, which can then be further reviewed. Some false negatives in this category may be acceptable given the high performance on the Fraud and Regular classes.

In summary, the voting classifier model proposed is well-suited for this fraud detection problem and will provide substantial business value.

5.2 Logistic Delay Detection

Based on the results of our logistic delay detection task, we can conclude that the Logistic Regression model performed very well in detecting delays in the logistic process of our deliveries. With an accuracy of 0.9678, the model can correctly classify 96.78% of the samples. Moreover, with a recall of 0.9889 and precision of 0.9564, the model can identify almost all the actual logistic delays while also avoiding a high rate of false positives.

By being able to accurately detect logistic delays, we can take proactive measures to mitigate the negative impact on our customers and our reputation. For example, we can prioritize the handling and delivery of delayed orders, provide updates to affected customers, and work with our logistics partners to identify and address any issues in the delivery process. This can lead to increased customer satisfaction, retention, and loyalty. Additionally, by improving the efficiency of our delivery process, we can reduce costs and increase profitability.

5.3 Delivery Prediction

Based on the results of our experiment, we conclude that the Bayesian Ridge Regression model is the best algorithm for delivery prediction. It outperformed the other models in terms of rMSE and R2 and had the lowest prediction error across all days.

Using this model can improve the accuracy and efficiency of delivery prediction, which is crucial for logistics and e-commerce companies. Accurate delivery time predictions can help companies to better manage their resources, reduce costs, and improve customer satisfaction by providing more reliable delivery estimates. Furthermore, with a more precise prediction, companies can avoid overcommitting to delivery times that they cannot fulfill, which can help to avoid customer dissatisfaction and increase their reputation in the market. Therefore, the implementation of the Bayesian Ridge Regression model in a delivery prediction system can lead to significant benefits for the company.

6. Web Site

This website is done to call a set of functions. Each function takes in a trained model and a DataFrame as inputs and returns a Streamlit app that displays the model's performance metrics and prediction results.

The GitHub repository is cloned to import the functions.

The function uses a loop to train and evaluate the model multiple times on shuffled subsets of the input DataFrame. For each iteration, the function first shuffles the input DataFrame, and then splits it into training and test sets. The function then performs several preprocessing steps on the training and test sets, including encoding categorical variables using one-hot encoding and label encoding, scaling the numerical features, and transforming the data using PCA.

After the data has been preprocessed, the function uses the trained model to make predictions on the test set and calculates several performance metrics, including mean squared error, mean absolute error, and r squared. The function also calculates the percentage of predictions within a given threshold value and stores this value in a list.

After all iterations have completed, the function calculates the mean of the performance metrics and the percentage of predictions within the threshold value across all iterations and displays these values in a Streamlit table. The function also displays a plot of the model's residuals and a QQ plot of the residuals to check for normality assumptions.

This is the case of the Fraud Detection but the same process with the appropriate preprocessing steps and appropriate output is done for the Logistic Delay Detection and Delivery Prediction.

Appendix A: Code Description

Here we are providing a short guide on how to run the code in a more fluent way.

01_EDA.ipynb:

To load the file (SupplyChainDataset.csv), it must be in the same directory of the notebook.
If running it on Google Colab, you can either load the file locally and insert the local path in the read_csv or load it into the Google Drive connected to Google Colab and inserting the path instead.
Running the notebook, a dataset called SupplyChainDataset_eda containing the preprocessing steps will be generated either in the folder. If using Colab, remember to specify the path in the to_csv function.

02_Prediction.ipynb

Since the requirement for this task in terms of encodings were slightly different with respect of other tasks, we imported the original dataset (SupplyChainDataset.csv) and performed the encoding. To run the notebook, you must follow the steps above.
A dataset called SupplyChainDataset_prediction.csv will be generated.

03_FraudDetection.ipynb

To run this notebook, you have to import the dataset generated from the 01_EDA.ipynb (SupplyChainDataset_eda).
Follow the steps above for a correct importing.
A dataset called df_fraud.csv will be generated.

04_LogisticDelay.ipynb

To run this notebook, you must import the dataset generated from the 01_EDA.ipynb (SupplyChainDataset_eda).
Follow the steps above for a correct importing.
A dataset called SupplyChainDataset_delay.csv will be generated.

05_WebSite.ipynb

To run this Streamlit application on the Colab notebook follow these steps.
You have to load in the results.py cell the datasets and the models.
In the delivery function you will have to load the SupplyChainDataset_prediction.csv and the br.pkl model generated above.
In the fraud function you will have to load the df_fraud.csv and the vc.pkl model generated above.
In the delivery function you will have to load the SupplyChainDataset_delay.csv and the lr.pkl model generated above.
Then in the following cell, my Authentication Key of pyngrok is present. You can either put yours if you have any or just use the one present. It should work anyway.

The following cell:

```
get_ipython().system_raw('./ngrok http 8501 &')
```

```
! curl -s http://localhost:4040/api/tunnels | python3 -c \
    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"
```

Needs to be run twice, then you can run this cell:

```
!streamlit run /content/results.py
```

And finally, by clicking on the link generated from the previous cell (this one below)

```
get_ipython().system_raw('./ngrok http 8501 &')
```

```
! curl -s http://localhost:4040/api/tunnels | python3 -c \
    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"
```

You can access the application.

If instead you are running in your local pc follow these steps.
Run this on the terminal:

```
git clone https://github.com/giakomorssi/Deloitte_Project
```

Run the results.py, then you may need to run this on the terminal

```
pip install --upgrade jinja2
```

Finally run the following on the terminal for the app to open:

```
streamlit run results.py
```

All the plots are generated from the code. The plots for the results are generated from the 05_WebSite.ipynb.

Appendix B: Author Contribution

Each member of the group worked together to achieve the results of the project.