TAYLOR'S UNIVERSITY
Wisdom • Integrity • Excellence

Bachelor of Computer Science (Hons)
Bachelor of Software Engineering (Hons)
Bachelor of Information Technology (Hons)

ASSIGNMENT COVER SHEET

Module Code: **ITS65704**

Module Name: **Data Science Principles**

Students Name	**Students ID:**
1) Ishihara Satoaki	0354208
2) Saranya Kandikatti	0354361
3) Nathaniel Roshan	0342485
4) Rishyal Richard Gomez	0344321
5) Muhammad Irsyal Bin Noraimi	0354790

Assignment No. / Title	Group Assignment Report
Course Tutor/Lecturer	Liyana 'Adilla Burhanuddin

Declaration *(need to be signed by students. Otherwise, the assignment will not be marked)*
We certify that this assignment is entirely our work, except where we have given fully documented references to the work of others, and that the material contained in this assignment has not previously been submitted for assessment in any other formal course of study.

Signature of Students: Ishihara Satoaki [illegible] [illegible] [illegible] [illegible]

Marks:	**Evaluated by:**
Evaluator's Comments:	

Table of Contents

Introduction

Background of Study

With the digital revolution, e-commerce has experienced unprecedented growth, especially in countries like Malaysia, with an anticipated market worth of USD 8.1 billion by 2025, up from USD 1.1 billion in 2021. As the e-commerce sector grows, so does the importance of ensuring timely package deliveries to maintain customer trust and satisfaction. This timely delivery relies heavily on vast amounts of data generated from user behaviours and various other e-commerce variables. Predicting whether a package will arrive on time has become essential, and this study focuses on leveraging this data to make accurate predictions.

- General Purpose and Motivation: To leverage data science principles in addressing challenges faced by the e-commerce industry and to extract actionable insights that could significantly enhance business operations.
- Problem Addressed: The often-overwhelming amount of e-commerce data that, if not analysed, could lead to missed opportunities for business enhancement.
- General Methods and Materials: The study will employ systematic data collection and transformation techniques, data cleaning using Pandas and NumPy libraries, and machine learning algorithms to build prediction models.

Problem Statement

E-commerce platforms possess an abundance of data, but the challenge lies in effectively leveraging this data to predict on-time package deliveries. The research question revolves around identifying the best machine learning model to predict whether a package will arrive on time, considering various e-commerce variables.

- Problem: The pivotal challenge is to analyse the expansive e-commerce dataset to derive insights specifically for predicting on-time deliveries by determining the most influential feature on the target.
- Negative Impact: Without precise predictions, e-commerce platforms risk tarnishing their reputation, losing customer trust, and incurring financial losses due to potential compensations or returns.

- Parties Affected: This problem affects e-commerce businesses, stakeholders, investors, delivery personnel, and most importantly, customers awaiting their packages.
- Benefit of Solving the Problem: Addressing this challenge ensures e-commerce platforms function more efficiently, leading to increased customer trust, reduced operational costs due to fewer delivery issues, and overall industry growth.

Objectives of Study

The primary objective of this study is to utilise data science methodologies to analyse the extensive e-commerce datasets, aiming to:

- To analyse the e-commerce dataset and extract variables that most influence package delivery times.
- To test and evaluate various machine learning models for their accuracy in predicting on-time package deliveries.
- To recommend the best machine learning model for e-commerce platforms to integrate into their systems for predicting delivery times.
- To determine the most influential feature with the certain feature importance methods to determine the most influential feature on outcomes of whether the delivery reached on time or not.

By achieving these objectives, we aim to bridge the gap between raw e-commerce data and actionable business strategies.

Literature Reviews

Huang and Benyoucef's study (2013) explores the shift from e-commerce to social commerce. It highlights the importance of design features in enhancing user experience and business strategies. These features include the integration of social media, ratings and reviews, recommendations, referrals, and community forums. The paper emphasises their significance in the Malaysian big data analytics (BDA) market, where such data is essential for tailored marketing, product placement optimization, and user engagement. The paper classifies design features that support a more community-oriented approach. It suggests that leveraging user-generated content and machine learning algorithms for recommendations can benefit not only e-commerce but also broader areas like financial decision-making and investment strategies. This underlines the ongoing importance of data and AI in the evolving digital economy.

With the rapidly growing significance of data and AI in the evolving digital economy, many researchers are exploring the use of Machine Learning (ML) algorithms for predictive purposes.

Jonquais and Krempl's 2019 research delved into the use of ML algorithms to predict shipping times between Southeast Asia and North America. They underscored the importance of predictive analytics in enhancing supply chain management and the reliability of maritime shipping schedules. Given the crucial role of shipping in international trade, they highlighted the need for accurate route scheduling to avoid delays and extra costs due to various obstacles, like adverse weather conditions. They developed a model that uses historical shipment times and external data on potential delay-causing factors to offer realistic predictions. Specifically, they used the Random Forest algorithm to better estimate shipment arrival times, integrating it with Maersk's Harmony tool, which provided past transit data. This successful implementation showcased the substantial benefits and future potential of predictive analytics in reducing operational costs and improving the efficiency of the transportation industry through meticulous data collection and advanced ML techniques.

Al-Saghir explored how machine learning can predict delays within supply chains in his 2022 article. He addressed the complexities of managing various variables and the unpredictability inherent in supply chain management. His study emphasised the importance of data collection, preprocessing, and ensuring data quality, which are critical steps for enhancing business strategies in e-commerce. Al-Saghir evaluated different machine learning algorithms—Linear Regression, Support Vector Machine, Logistic Regression, and Naive Bayes—to assess their predictive accuracy and performance in forecasting delays. Logistic Regression proved to be the most effective, with the highest accuracy of 75% and a recall rate of 68%. These insights into predictive analytics showcase the transformative potential of machine learning in optimising supply chain operations, which can be directly applied to e-commerce scenarios for predicting customer behaviour, improving inventory management, and refining decision-making processes to ultimately reduce costs, minimise delays, and boost customer satisfaction.

Salari, Liu, and Shen's 2020 study addressed the critical issue of real-time delivery forecasting in e-commerce, with a focus on accurately predicting package arrival times—a factor that directly influences customer satisfaction and retailer success. They proposed a novel data-driven framework using Quantile Regression Forests (QRF) and an asymmetric loss function to enhance the prediction of order delivery time distribution and to set reliable promised delivery times. This method outperforms traditional methods used by companies like JD.com, potentially improving sales volume by 3.7-6.1% while controlling delivery delays. The study underlines the need for integrated collaboration among online retailers, logistics, and last-mile delivery services to effectively implement such forecasting systems. Their work is significant as it provides a comprehensive solution to the delivery time prediction challenge, demonstrating the transformative impact of accurate, dynamic forecasting on operational efficiency and customer satisfaction in online retailing.

The use of predictive modelling techniques has gained popularity in e-commerce to enhance delivery time estimates. Kusumawati, K. N. (2021) paper discussed the application of various regression models, including Linear Regression, Ridge Regression, Lasso Regression, Elastic Net Regression, and Random Forest Regression. These models are used to predict delivery times and improve accuracy. The content highlights the success of the Random Forest Regression model with hyperparameter tuning in improving delivery time prediction. Compared to original data, MAE on original data has a bigger error than modelling using random forest with hyperparameter tuning which is 13.3% error while random forest with hyperparameter tuning has 5.6% error. Based on these results, the author makes a comparison for count number of shipment status differences between original data and after modelling. The author stated that prior to modelling, many shipments arrived too soon than estimated. After modelling, 49,5% of deliveries arrive on time while the original data has 14.2% of deliveries arriving on time. This means that on the new model which is a random forest with hyperparameter tuning , they can estimate arrival times better than the previous model. This shows that the use of predictive modelling techniques, as demonstrated in the paper, is a promising approach to improve delivery time estimates.

Methods

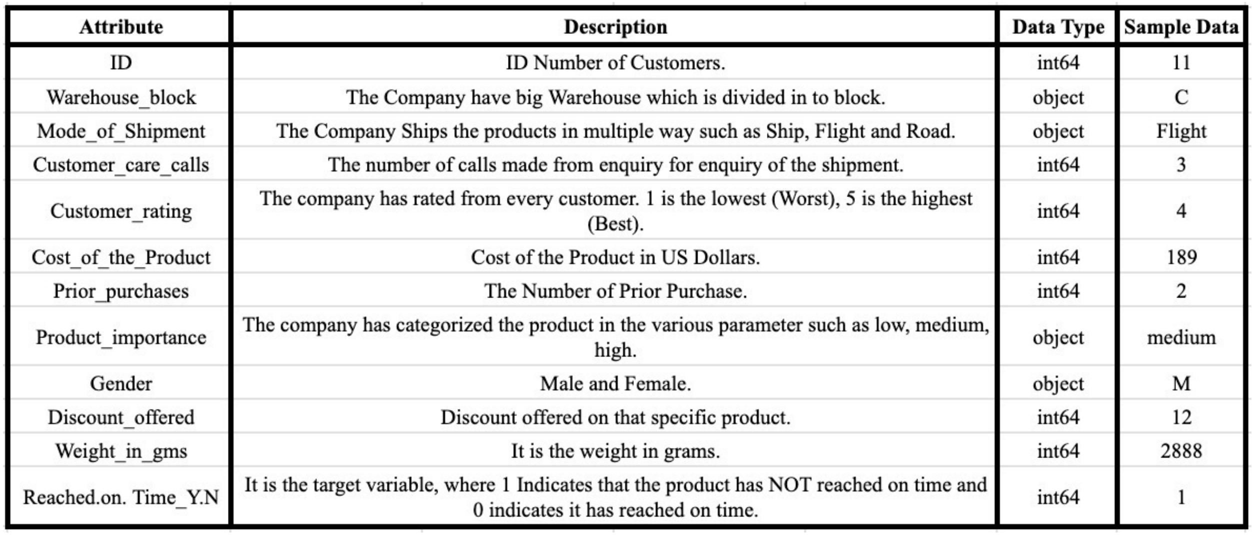
Jupyter Notebook (mainly for Windows), Google Colab (mainly for MacOS), Python with version 3.10.12. were modified as softwares for developing, fitting, and evaluating machine learning algorithms.

We have utilised Pandas and NumPy libraries from Python to perform the following data cleaning steps:

- Removing major errors such as duplicates or missing values, and outliers.
- Removing unwanted data points that are irrelevant to this study.
- Handling an imbalanced dataset by sampling.

The libraries were also used to investigate the structure and shape of the dataset. Also, the Matplotlib and Seaborn library has been used to generate visual contexts (e.g., scatter plots, bar charts, histograms etc.) based on the datasets in this study.

Dataset



Attribute	Description	Data Type	Sample Data
ID	ID Number of Customers.	int64	11
Warehouse_block	The Company have big Warehouse which is divided in to block.	object	C
Mode_of_Shipment	The Company Ships the products in multiple way such as Ship, Flight and Road.	object	Flight
Customer_care_calls	The number of calls made from enquiry for enquiry of the shipment.	int64	3
Customer_rating	The company has rated from every customer. 1 is the lowest (Worst), 5 is the highest (Best).	int64	4
Cost_of_the_Product	Cost of the Product in US Dollars.	int64	189
Prior_purchases	The Number of Prior Purchase.	int64	2
Product_importance	The company has categorized the product in the various parameter such as low, medium, high.	object	medium
Gender	Male and Female.	object	M
Discount_offered	Discount offered on that specific product.	int64	12
Weight_in_gms	It is the weight in grams.	int64	2888
Reached.on. Time_Y.N	It is the target variable, where 1 Indicates that the product has NOT reached on time and 0 indicates it has reached on time.	int64	1

Figure 1: Metadata of our dataset.

The "E-Commerce Shipping Data" dataset, curated by Prachi Gopalani, offers a window into the operations of an international e-commerce company. The company, specialising in the sale of electronic products, is keen on extracting insights from their extensive customer database using advanced machine learning techniques.

The dataset is both vast and detailed, encompassing 10,999 observations across 12 distinct variables. Each customer is uniquely identified through an ID. The data further sheds light on the warehouse block from which products are dispatched, ranging from blocks A to E. It provides insights into the preferred mode of shipment—whether by Ship, Flight, or Road. An interesting metric is the number of calls made to customer care, reflecting shipment inquiries. Additionally, customers rate their experience on a scale of 1 to 5, providing a qualitative assessment of the service. The dataset also enumerates the cost of the product in US Dollars, previous purchases made by the customer, and the importance level of the product, categorised as low, medium, or high. Gender details are captured as Male or Female. Discounts offered on specific products are also documented. Each product's weight is meticulously recorded in grams. The dataset culminates in a binary target variable indicating the timeliness of the product's delivery: a 1 signifies a delay, while a 0 confirms an on-time delivery.

This comprehensive dataset serves as an invaluable resource for those keen on understanding the nuances of e-commerce operations and customer behaviours.

Algorithms

The Machine Learning Models were evaluated as the best algorithms in related works, and a number of other orthodox models that are used for classification prediction tasks are employed in this experiment.

- Logistic Regression
- Decision Tree
- Random Forest
- XGBoost
- K-Nearest Neighbors

The base model of these five models, and the resampled version of each (total 10 cases) would be tested, and evaluations would be compared to determine the best model for predicting shipping arrival on time.

Evaluation Metrics

Four evaluation metrics are applied in this project:

- **Mean Squared Error (MSE):** It is the value calculated by squaring the 'difference (or error) between the correct and predicted values' for each data and dividing the sum of the squared values by the number of data (or mean value).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$n = the\ number\ of\ data$$

$$Y_i = observed\ data$$

$$\hat{Y}_i = actual\ data$$

- **Root Mean Squared Error (RMSE):** The value of the square root of the mean squared error (MSE).

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

$$n = the\ number\ of\ data$$

$$Y_i = observed\ data$$

$$\hat{Y}_i = actual\ data$$

- **Mean Absolute Error (MAE):** It is the absolute value of the difference (or error) between the correct and predicted values calculated for each data and the sum divided by the number of data (or mean value).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|^2$$

$$n = the\ number\ of\ data$$

$$Y_i = observed\ data$$

$$\hat{Y}_i = actual\ data$$

- **R-Squared (R^2)**: It is the measure that shows the proportion of the variance of a dependent variable explained by an independent variable in the model. It assesses the fit of the regression model.

$$R^2 = 1 - \sum_{i=1}^{n}\frac{(Y_i - \hat{Y}_i)^2}{(Y_i - \bar{Y})^2}$$

$$n = the\ number\ of\ data$$

$$Y_i = observed\ data$$

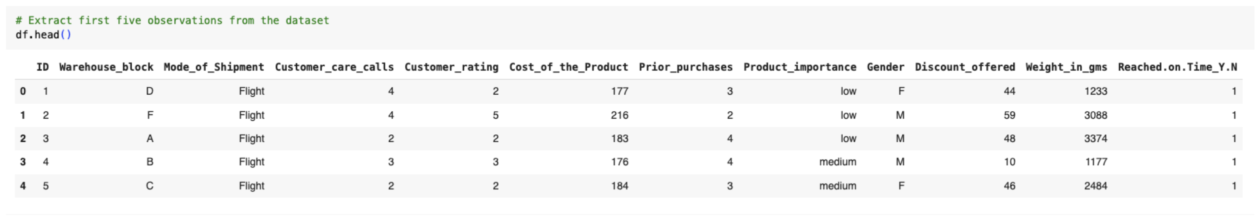
$$\hat{Y}_i = actual\ data$$

$$\bar{Y} = mean\ of\ the\ data$$

Analysis Implementation with coding

1. Exploratory Data Analysis

1.1 Initial Exploratory Data Analysis

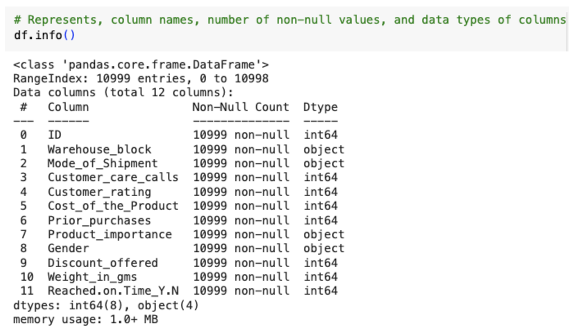


```
# Extract first five observations from the dataset
df.head()
```

	ID	Warehouse_block	Mode_of_Shipment	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Product_importance	Gender	Discount_offered	Weight_in_gms	Reached.on.Time_Y.N
0	1	D	Flight	4	2	177	3	low	F	44	1233	1
1	2	F	Flight	4	5	216	2	low	M	59	3088	1
2	3	A	Flight	2	2	183	4	low	M	48	3374	1
3	4	B	Flight	3	3	176	4	medium	M	10	1177	1
4	5	C	Flight	2	2	184	3	medium	F	46	2484	1

Figure 2: The top 5 records in the csv file dataset “e-commerce_shipping_data.csv”.

Figure 2 is the top 5 instances in the dataframe with the data from “e-commerce_shipping_data.csv” extracted with head() function.

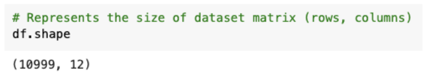


```
# Represents, column names, number of non-null values, and data types of columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10999 entries, 0 to 10998
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10999 non-null  int64
 1   Warehouse_block      10999 non-null  object
 2   Mode_of_Shipment     10999 non-null  object
 3   Customer_care_calls  10999 non-null  int64
 4   Customer_rating      10999 non-null  int64
 5   Cost_of_the_Product  10999 non-null  int64
 6   Prior_purchases      10999 non-null  int64
 7   Product_importance   10999 non-null  object
 8   Gender               10999 non-null  object
 9   Discount_offered     10999 non-null  int64
 10  Weight_in_gms        10999 non-null  int64
 11  Reached.on.Time_Y.N  10999 non-null  int64
dtypes: int64(8), object(4)
memory usage: 1.0+ MB
```

Figure 3: Column names, number of non-null values and data types, extracted with df.info() function.

Information in Figure 3 represents the output extracted with df.info(). The number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values) are contained in the information. Referring to the production of information, all columns are holding 10999 non-null values, without exception.

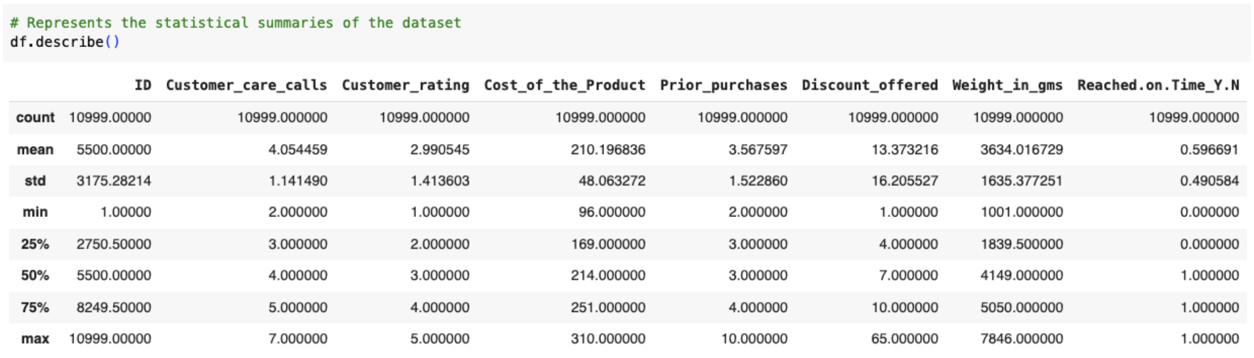


```
# Represents the size of dataset matrix (rows, columns)
df.shape
```

```
(10999, 12)
```

Figure 4: A tuple of dataset size returned by df.shape.

The tuple in Figure 4 represents the dimensionality of the DataFrame. The results show that the data we are interacting with for this analysis comprises 10999 rows and 12 columns of value groupings.



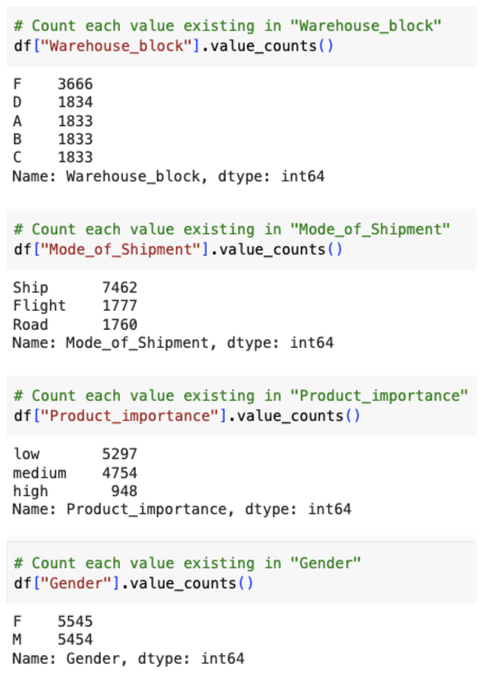
```
# Represents the statistical summaries of the dataset
df.describe()
```

	ID	Customer_care_calls	Customer_rating	Cost_of_the_Product	Prior_purchases	Discount_offered	Weight_in_gms	Reached.on.Time_Y.N
count	10999.00000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000	10999.000000
mean	5500.00000	4.054459	2.990545	210.196836	3.567597	13.373216	3634.016729	0.596691
std	3175.28214	1.141490	1.413603	48.063272	1.522860	16.205527	1635.377251	0.490584
min	1.00000	2.000000	1.000000	96.000000	2.000000	1.000000	1001.000000	0.000000
25%	2750.50000	3.000000	2.000000	169.000000	3.000000	4.000000	1839.500000	0.000000
50%	5500.00000	4.000000	3.000000	214.000000	3.000000	7.000000	4149.000000	1.000000
75%	8249.50000	5.000000	4.000000	251.000000	4.000000	10.000000	5050.000000	1.000000
max	10999.00000	7.000000	5.000000	310.000000	10.000000	65.000000	7846.000000	1.000000

Figure 5: Description of statistical summaries of values in columns. Extracted with df.describe() function.

Figure 5 provides a comprehensive statistical summary of various columns in the dataset using the df.describe() function. The dataset encompasses 10,999 records, as evident from the count row. The 'ID' column, representing unique identifiers, spans from 1 to 10,999, with a mean value of 5,500. Analysing customer interactions, the average number of customer care calls made is approximately 4.05, with the calls ranging from a minimum of 2 to a maximum of 7.

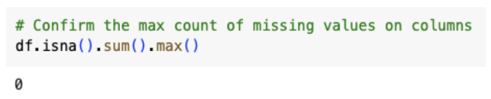
The customer rating column reveals that the average rating given is around 2.99, with ratings varying from 1 to 5. Delving into the product's financial aspects, the average cost is about 210.20 units, with the products priced as low as 96 units and going up to 310 units. On the purchasing behaviour front, customers, on average, have made about 3.57 prior purchases, with the number fluctuating between 1 and 10. Additionally, the discount offered on products averages around 13.37 units, reaching up to 65 units in some cases. The products' weight has a mean value of approximately 3634.02 grams, with the lightest product weighing 1001 grams and the heaviest tipping the scales at 7846 grams. Lastly, the 'Reached.on.Time_Y.N' column, likely indicating whether a shipment was timely (with 1 being "Yes" and 0 being "No"), shows that approximately 59.67% of the products were delivered on time.



```
# Count each value existing in "Warehouse_block"
df["Warehouse_block"].value_counts()

F    3666
D    1834
A    1833
B    1833
C    1833
Name: Warehouse_block, dtype: int64

# Count each value existing in "Mode_of_Shipment"
df["Mode_of_Shipment"].value_counts()

Ship      7462
Flight    1777
Road      1760
Name: Mode_of_Shipment, dtype: int64

# Count each value existing in "Product_importance"
df["Product_importance"].value_counts()

low       5297
medium    4754
high       948
Name: Product_importance, dtype: int64

# Count each value existing in "Gender"
df["Gender"].value_counts()

F    5545
M    5454
Name: Gender, dtype: int64
```

Figure 6: Value count

Figure 6 depicts the use of various Python programmes to analyse distinct aspects of a dataframe, df. Initially, the algorithm investigates the distribution of various attributes inside the dataframe. When examining the "Warehouse_block" column, for example, it is discovered that there are five separate warehouse blocks, with block 'F' being the most prevalent, housing 3666 entries. Similarly, the mode of shipment reveals that the 'Ship' method is the most commonly utilised, accounting for 7462 of total shipments, whilst the distribution in "Product_importance" reveals a preference for 'low' importance products, accounting for 5297 entries. A gender analysis shows a virtually even distribution of males and females, with females slightly outnumbering males by a count of 5545 to 5454.



```
# Confirm the max count of missing values on columns
df.isna().sum().max()

0
```

Figure 7: Codes and an Output to analyse missing values.

The DataFrame function “df.isna().sum().max()” in Figure 7 compares the total number of missing values in each column, discovers the largest count and prints it out as an output. On the other hand, Figure 7 output indicates the output was 0, which means none of the columns were holding missing values, therefore we could work on developing machine learning algorithms for prediction without coping with missing value problems.

1.2 Descriptive Exploratory Data Analysis with Data Visualization

First, we would explore the numerical features. To distinguish numerical features from categorical features, we would execute the below code.

```python
# Extract numerical features from the DataFrame
numeric_columns = df.select_dtypes(include = ["int64", "float64"])
```

Figure 8 : the source code to include features which are integer data type or float data type.

Now we explore with Histogram plots.

```python
# Histogram for each numerical feature
for nu_column in numeric_columns:
    plt.figure(figsize=(10,6))
    sns.histplot(df[nu_column], kde=False, bins=30)
    plt.title(f'Distribution of {nu_column}')
    plt.show()
```

Figure 9 : Source codes to extract histograms for each of the numerical columns.

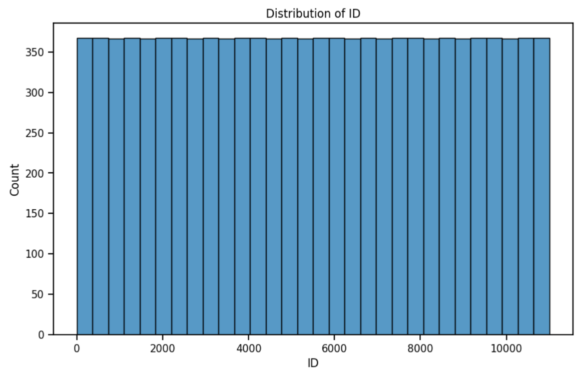


Figure 9.1

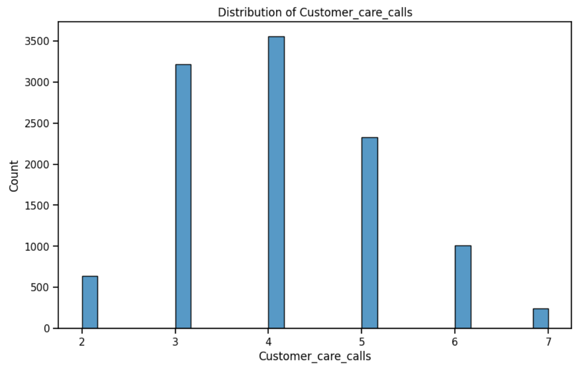


Figure 9.2

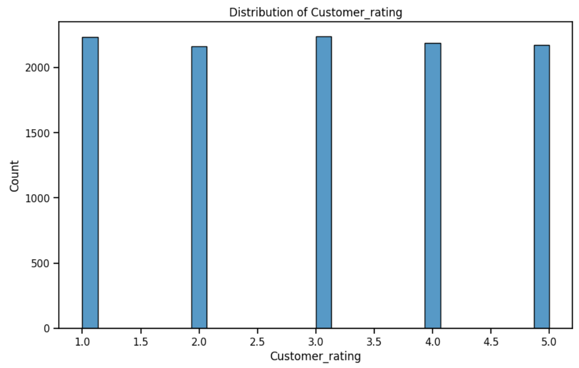


Figure 9.3

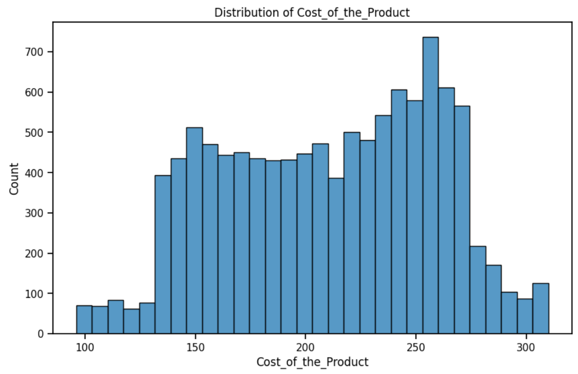


Figure 9.4

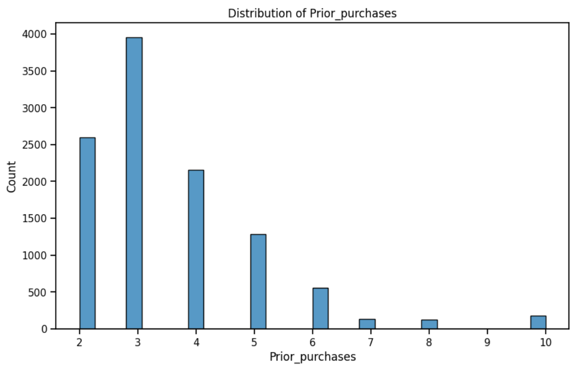


Figure 9.5

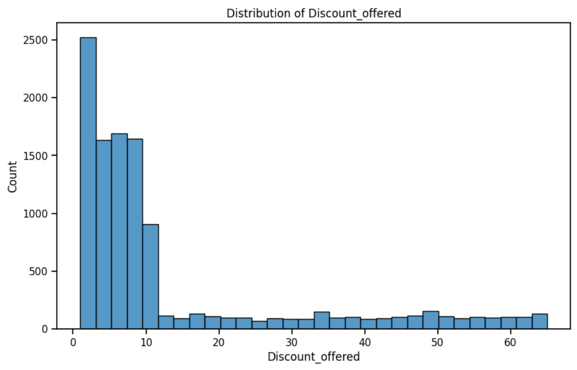


Figure 9.6

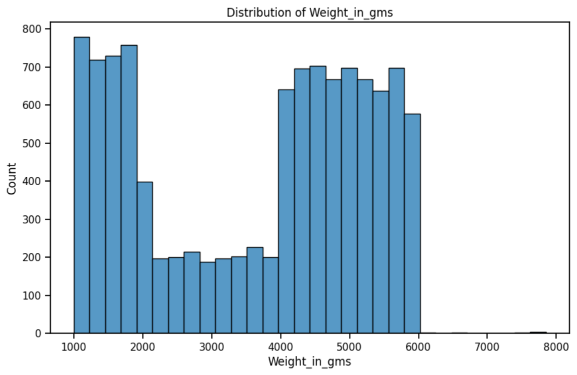


Figure 9.7

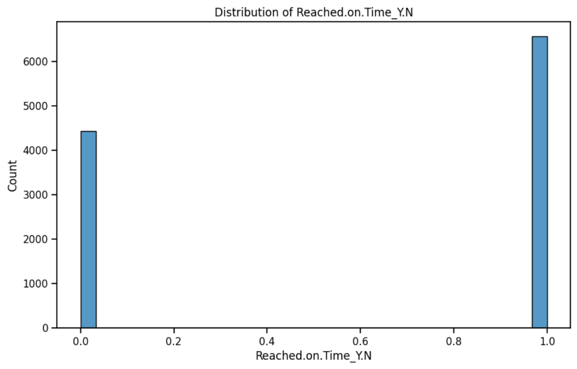
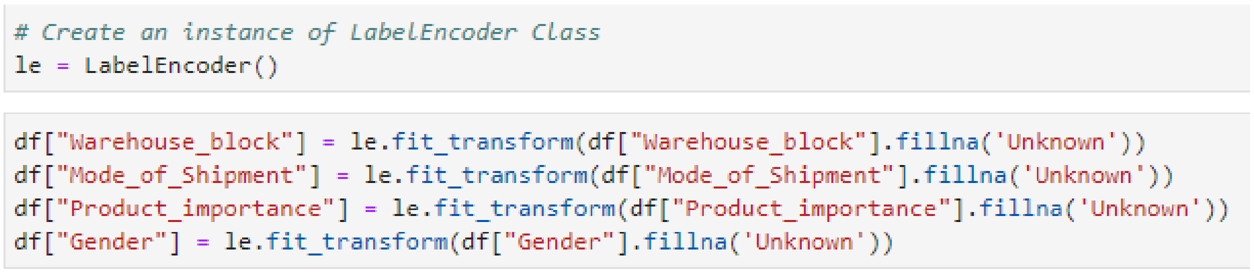


Figure 9.8

Figure 9.1 Shows the Distribution of ID (ID over count). Figure 9.2 Shows the distribution of customer care calls (Calls over count) and we can note that the highest frequency is 4. Figure 9.3 depicts the distribution of customer rating (Rating over count) where we can see that the number of customer ratings for 1,2,3,4, and 5 are all high. In figure 9.4 we can see the distribution of cost of the product (cost over count) and we can note that the highest frequency is near 250.

Figure 9.5 shows the distribution of prior purchases (Prior purchases over count) and we can see the highest frequency is 3, and the count is close to 4000. Figure 9.6 depicts the distribution of discount offered (Discount offer over count) and we can see that the majority of the high frequency is below 10. Figure 9.7 shows the weight distribution in grams (Weight over count) we can see that the highest frequency is 1000 grams. Figure 9.8 shows the distribution that the delivery reached on time in Yes/No form, (Reached on time over count) and the highest frequency is Yes at around 6500 and the No at around 4500.

2. Label Encoding

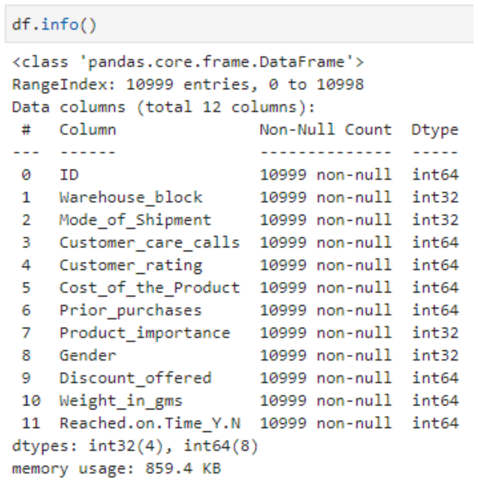


```
# Create an instance of LabelEncoder Class
le = LabelEncoder()
```

```
df["Warehouse_block"] = le.fit_transform(df["Warehouse_block"].fillna('Unknown'))
df["Mode_of_Shipment"] = le.fit_transform(df["Mode_of_Shipment"].fillna('Unknown'))
df["Product_importance"] = le.fit_transform(df["Product_importance"].fillna('Unknown'))
df["Gender"] = le.fit_transform(df["Gender"].fillna('Unknown'))
```

Figure 10: convert four categorical features with initialised LabelEncoder class.

Since we have a couple of categorical features, we would perform label encoding to convert them into numerical features.



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10999 entries, 0 to 10998
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   10999 non-null  int64
 1   Warehouse_block      10999 non-null  int32
 2   Mode_of_Shipment     10999 non-null  int32
 3   Customer_care_calls  10999 non-null  int64
 4   Customer_rating      10999 non-null  int64
 5   Cost_of_the_Product  10999 non-null  int64
 6   Prior_purchases      10999 non-null  int64
 7   Product_importance   10999 non-null  int32
 8   Gender               10999 non-null  int32
 9   Discount_offered     10999 non-null  int64
 10  Weight_in_gms        10999 non-null  int64
 11  Reached.on.Time_Y.N  10999 non-null  int64
dtypes: int32(4), int64(8)
memory usage: 859.4 KB
```

Figure 11: now all columns in the dataset are numerical columns.

After the label encoding is successfully performed, we would analyse the correlation matrix of all 12 columns' correlations with the correlation matrix heatmap, created with the entire dataset.

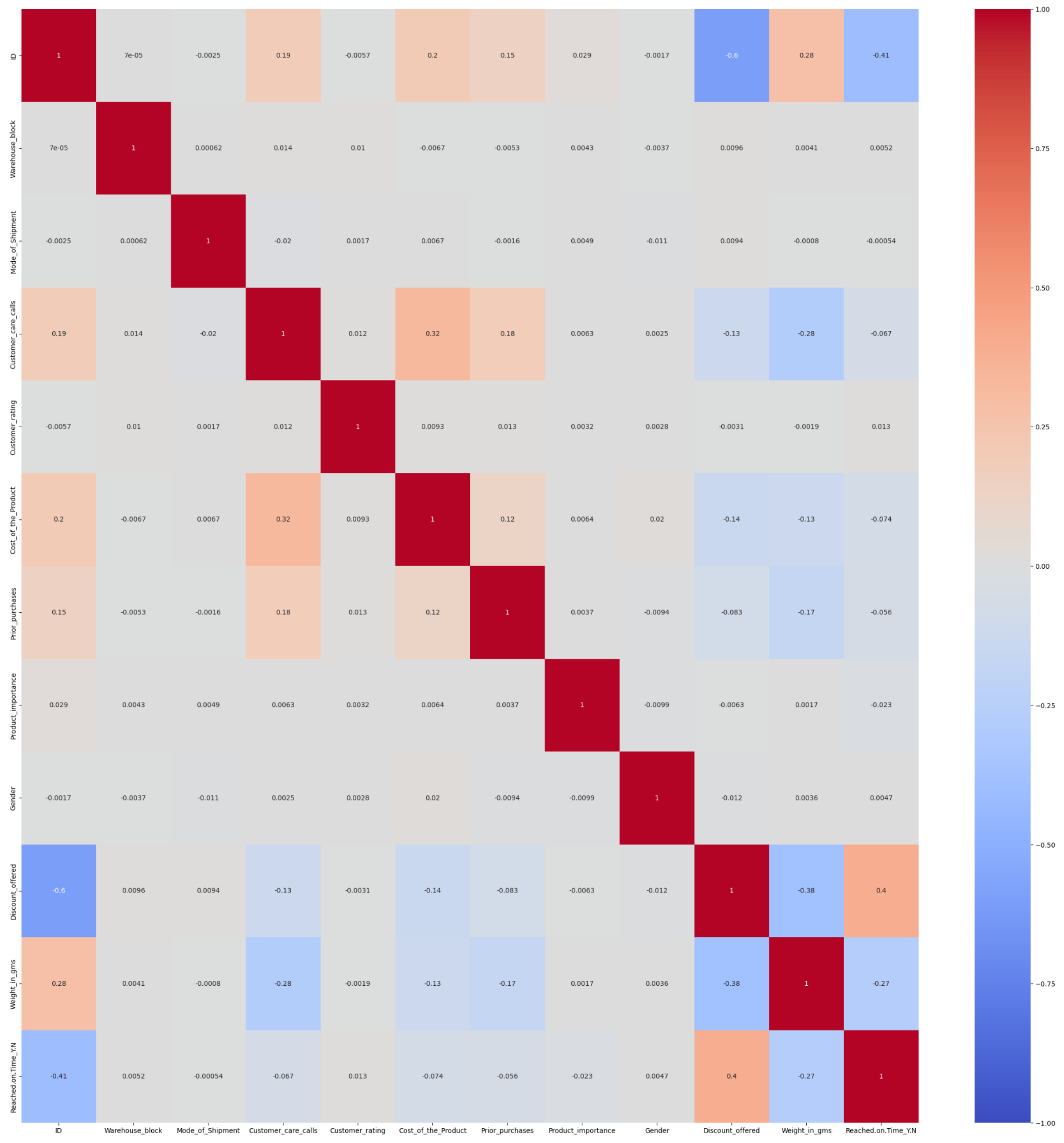
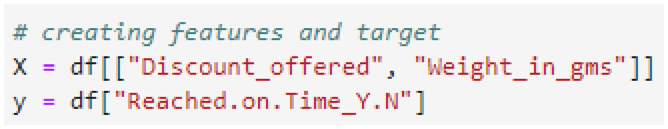


Figure 12: An updated Correlation Matrix Heatmap (constructed by 12 features).

Strongest correlations with the target variable are the one with “Discount_offered” and “Weight_in_gms”, “id”. Other features’ correlations with the target variable are generally between -0.1 to 0.1, guessed as “extremely low correlation”.

3. Modelling

2.0 Splitting data and resampling



```
# creating features and target
X = df[["Discount_offered", "Weight_in_gms"]]
y = df["Reached.on.Time_Y.N"]
```

Figure 13 : Codes to split data into features and the target.

Below are the features we have considered including or excluding for modelling:

[Include]

- **Discount_offered**: It has a notable negative correlation (-0.38) with Reached.on.Time_Y.N, which is significant.
- **Weight_in_gms**: A negative correlation of -0.27 with Reached.on.Time_Y.N is also significant.

[Possibly Exclude]

- **ID**: It is a feature for identifying each observation, so would be excluded.
- **Gender**: Its correlation with Reached.on.Time_Y.N is relatively low (0.0047).
- **Warehouse_Block**: Again, the correlation with the target variable is low (0.0052).
- **Prior_purchases**: The correlation is not strong enough (0.0087) to indicate a strong linear relationship.
- **Customer_care_calls**: The correlation is not very strong (0.0094).
- **Mode_of_Shipment**: Shows relatively low correlation (-0.00054) with Reached.on.Time_Y.N.
- **Product_importance**: Has a correlation of -0.023 with Reached.on.Time_Y.N, it's not as strong as some others.
- **Cost_of_the_Product**: This has a correlation of -0.074 with the target, which is not worth considering.
- **Customer_rating**: Since the correlation is low (0.013), it is not worth including.

The selections were based on the correlation matrix and its analyses on Figure 12.

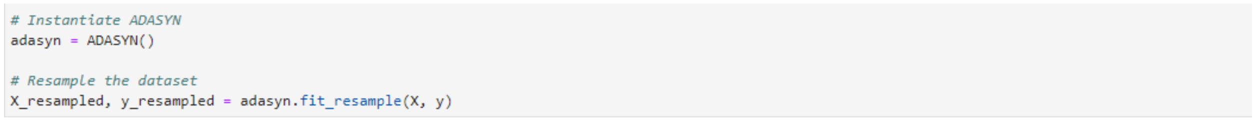

```
# Instantiate ADASYN
adasyn = ADASYN()

# Resample the dataset
X_resampled, y_resampled = adasyn.fit_resample(X, y)
```

Figure 14: ADASYN resampling.

Before actually creating the Train set and Test set, create another X variable and y variable with resampling for further analyses. Figure 14 indicates the processes to create an instance of ADASYN class, fit X and y to create x_resampled and y_resampled (resampled features and resampled target variable).

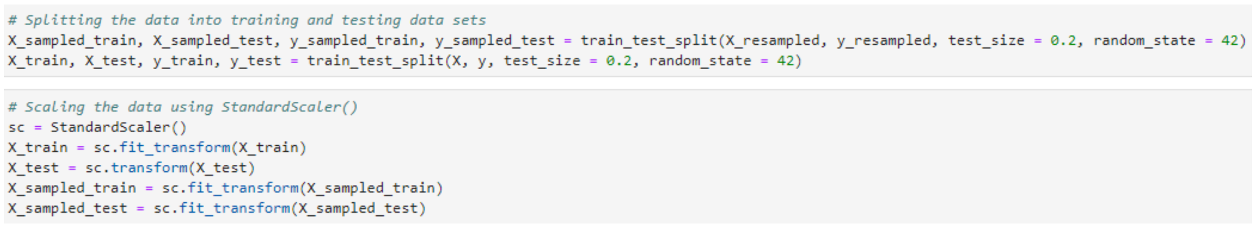

```
# Splitting the data into training and testing data sets
X_sampled_train, X_sampled_test, y_sampled_train, y_sampled_test = train_test_split(X_resampled, y_resampled, test_size = 0.2, random_state = 42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```
```
# Scaling the data using StandardScaler()
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_sampled_train = sc.fit_transform(X_sampled_train)
X_sampled_test = sc.fit_transform(X_sampled_test)
```

Figure 15: Splitting the data without resampling and resampled data.

Then, create Training and Testing dataset that can be used to evaluate the model's performance. Figure 15 portrays the command line for splitting the original dataset into two, then fit the X and Y variables into the X_train, X_test datasets and Y_train, Y_test datasets.

2.1 Logistic Regression

Logistic regression is a statistical method that predicts the probability that an input belongs to one of the two classes. It uses the sigmoid function to transform the linear combination of input features into a value between 0 and 1. We must also set a threshold to make the binary prediction.

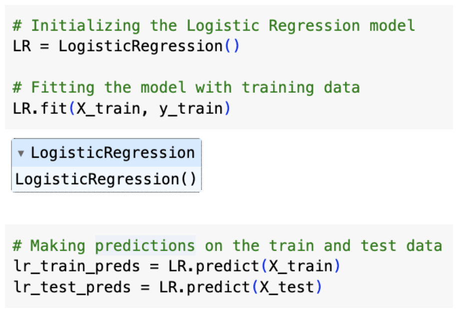
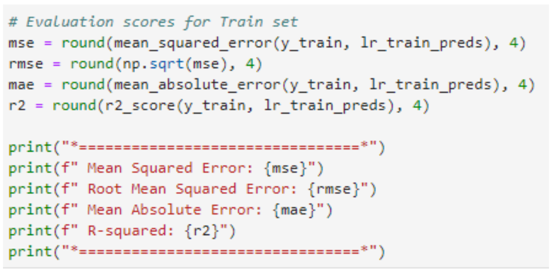
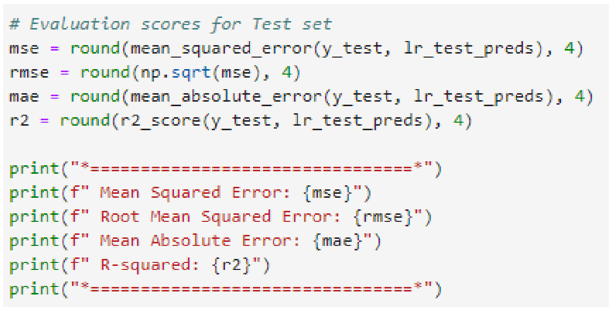

```
# Initializing the Logistic Regression model
LR = LogisticRegression()

# Fitting the model with training data
LR.fit(X_train, y_train)
```
```
▾ LogisticRegression
LogisticRegression()
```
```
# Making predictions on the train and test data
lr_train_preds = LR.predict(X_train)
lr_test_preds = LR.predict(X_test)
```

Figure 16: Initialising and fitting, making predictions with Logistic Regression.



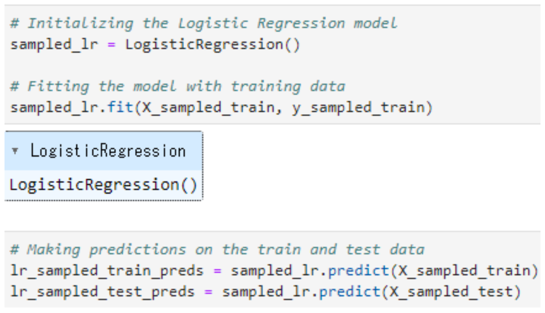
```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_train, lr_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_train, lr_train_preds), 4)
r2 = round(r2_score(y_train, lr_train_preds), 4)

print("*===============================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*===============================*")
```

Figure 17: Calculating evaluations with Logistic Regression model on the train set, and returning them.



```
# Evaluation scores for Test set
mse = round(mean_squared_error(y_test, lr_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_test, lr_test_preds), 4)
r2 = round(r2_score(y_test, lr_test_preds), 4)

print("*===============================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*===============================*")
```

Figure 18: Calculating evaluations with Logistic Regression model on the test set, and returning them.

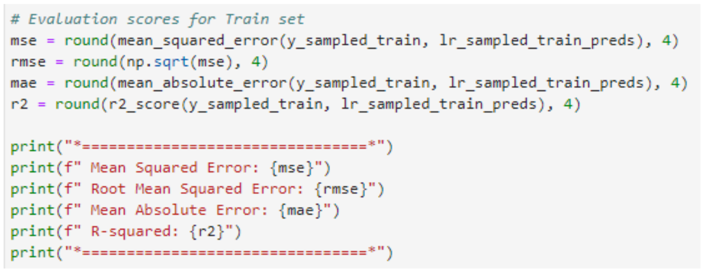


```
# Initializing the Logistic Regression model
sampled_lr = LogisticRegression()

# Fitting the model with training data
sampled_lr.fit(X_sampled_train, y_sampled_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

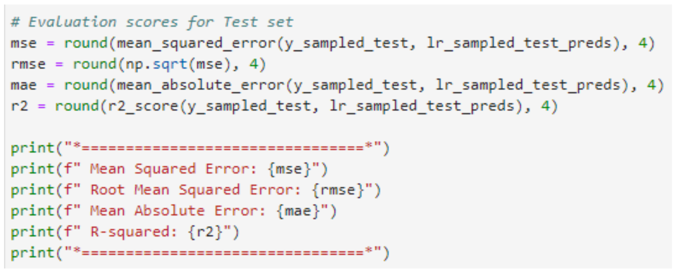
```
# Making predictions on the train and test data
lr_sampled_train_preds = sampled_lr.predict(X_sampled_train)
lr_sampled_test_preds = sampled_lr.predict(X_sampled_test)
```

Figure 19: Initialising and fitting, making predictions with resampled data on Logistic Regression.



```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_sampled_train, lr_sampled_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_train, lr_sampled_train_preds), 4)
r2 = round(r2_score(y_sampled_train, lr_sampled_train_preds), 4)

print("*===============================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*===============================*")
```

Figure 20: Calculating evaluations of ADASYN - Logistic Regression on the train set, and returning them.

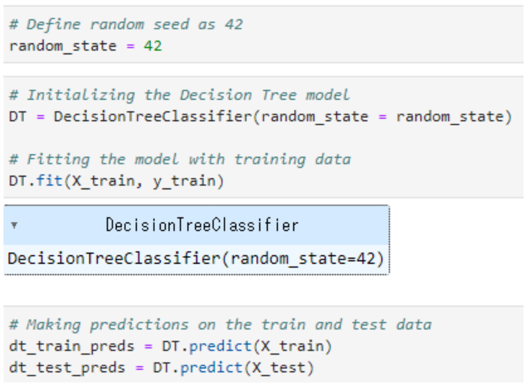


```
# Evaluation scores for Test set
mse = round(mean_squared_error(y_sampled_test, lr_sampled_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_test, lr_sampled_test_preds), 4)
r2 = round(r2_score(y_sampled_test, lr_sampled_test_preds), 4)

print("*==============================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*==============================*")
```

Figure 21: Calculating evaluations of ADASYN - Logistic Regression on the test set, and returning them.

2.2 Decision Tree

The decision tree is a ML algorithm that works by splitting the data based on features and leads to a final prediction at the leaf nodes. It is used for classification and regression. However decision trees can overfit. This issue can be mitigated by methods like pruning. In our code below we define the seed, initialise and fit the model and make predictions.



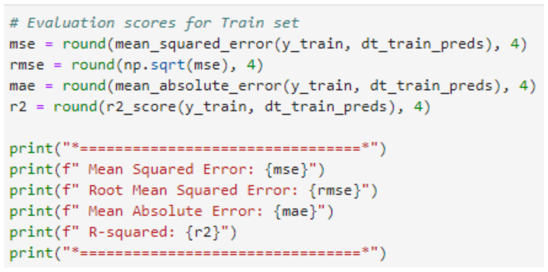
```
# Define random seed as 42
random_state = 42
```

```
# Initializing the Decision Tree model
DT = DecisionTreeClassifier(random_state = random_state)

# Fitting the model with training data
DT.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

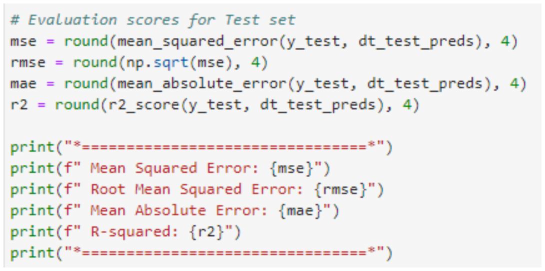
```
# Making predictions on the train and test data
dt_train_preds = DT.predict(X_train)
dt_test_preds = DT.predict(X_test)
```

Figure 22: Initialising and fitting, making predictions with Decision Tree.



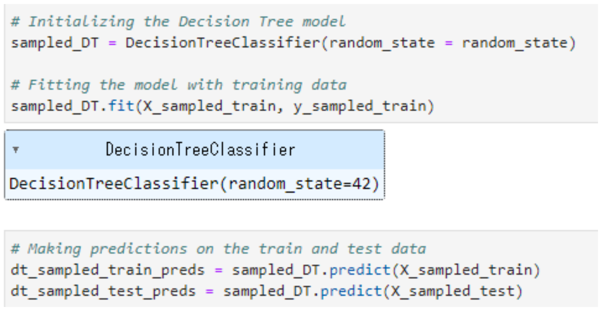
```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_train, dt_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_train, dt_train_preds), 4)
r2 = round(r2_score(y_train, dt_train_preds), 4)

print("*==============================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*==============================*")
```

Figure 23: Calculating evaluations with Decision Tree classifier on the train set, and returning them.



```python
# Evaluation scores for Test set
mse = round(mean_squared_error(y_test, dt_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_test, dt_test_preds), 4)
r2 = round(r2_score(y_test, dt_test_preds), 4)

print("*===============================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*===============================*")
```

Figure 24: Calculating evaluations with Decision Tree classifier on the test set, and returning them.

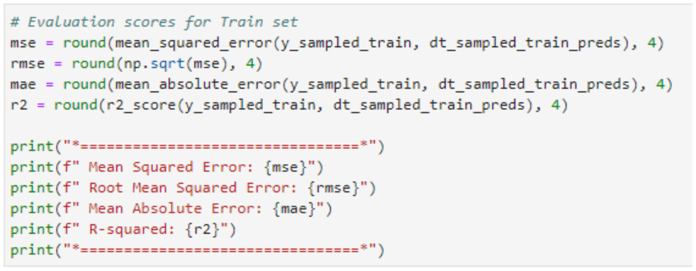


```python
# Initializing the Decision Tree model
sampled_DT = DecisionTreeClassifier(random_state = random_state)

# Fitting the model with training data
sampled_DT.fit(X_sampled_train, y_sampled_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

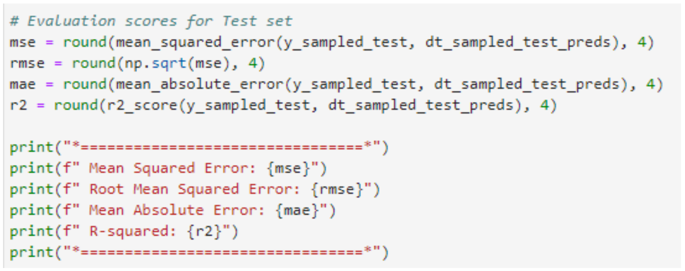
```python
# Making predictions on the train and test data
dt_sampled_train_preds = sampled_DT.predict(X_sampled_train)
dt_sampled_test_preds = sampled_DT.predict(X_sampled_test)
```

Figure 25: Initialising and fitting, making predictions with resampled data on Decision Tree.



```python
# Evaluation scores for Train set
mse = round(mean_squared_error(y_sampled_train, dt_sampled_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_train, dt_sampled_train_preds), 4)
r2 = round(r2_score(y_sampled_train, dt_sampled_train_preds), 4)

print("*===============================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*===============================*")
```

Figure 26: Calculating evaluations of ADASYN - Decision Tree classifier on the train set, and returning them.

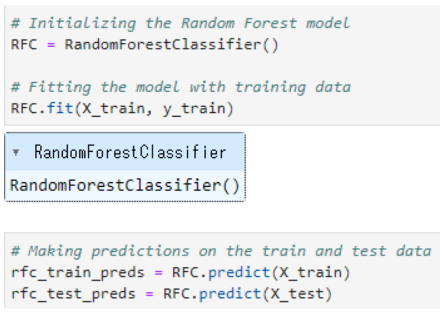


```python
# Evaluation scores for Test set
mse = round(mean_squared_error(y_sampled_test, dt_sampled_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_test, dt_sampled_test_preds), 4)
r2 = round(r2_score(y_sampled_test, dt_sampled_test_preds), 4)

print("*===============================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*===============================*")
```

Figure 27: Calculating evaluations of ADASYN - Decision Tree classifier on the test set, and returning them.

2.3 Random Forest

Random forest is a learning method that is an ensemble of decision trees that combines bootstrapped samples and random selection of features to make accurate predictions.

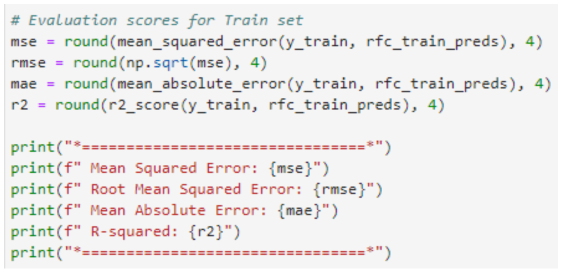


```
# Initializing the Random Forest model
RFC = RandomForestClassifier()

# Fitting the model with training data
RFC.fit(X_train, y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

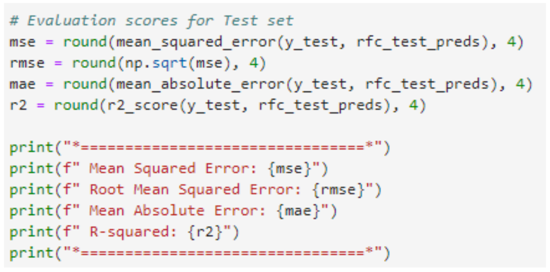
```
# Making predictions on the train and test data
rfc_train_preds = RFC.predict(X_train)
rfc_test_preds = RFC.predict(X_test)
```

Figure 28: Initialising and fitting, making predictions with Random Forest.



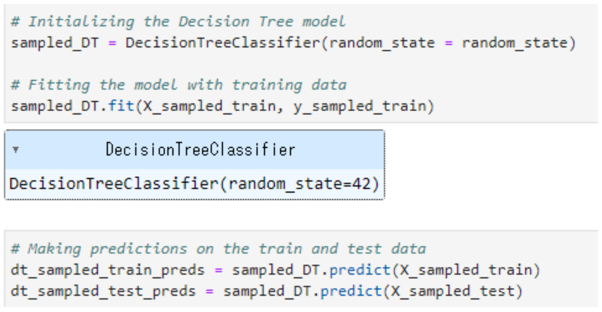
```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_train, rfc_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_train, rfc_train_preds), 4)
r2 = round(r2_score(y_train, rfc_train_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 29: Calculating evaluations with Random Forest classifier on the train set, and returning them.



```
# Evaluation scores for Test set
mse = round(mean_squared_error(y_test, rfc_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_test, rfc_test_preds), 4)
r2 = round(r2_score(y_test, rfc_test_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 30: Calculating evaluations with Random Forest classifier on the test set, and returning them.

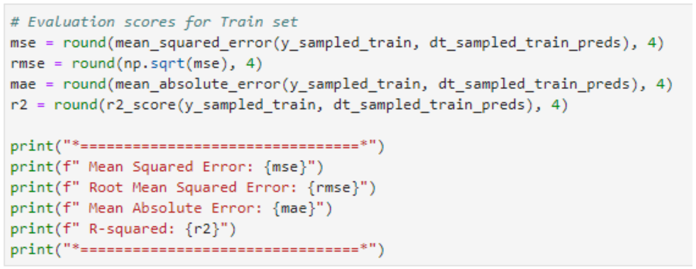


```
# Initializing the Decision Tree model
sampled_DT = DecisionTreeClassifier(random_state = random_state)

# Fitting the model with training data
sampled_DT.fit(X_sampled_train, y_sampled_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

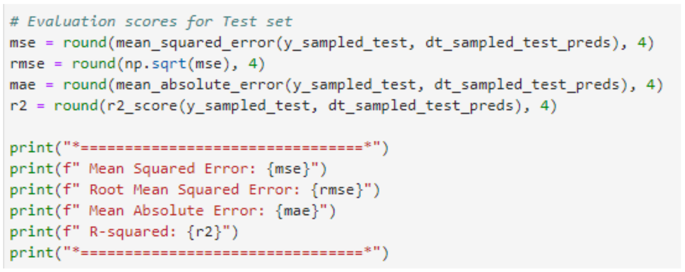
```
# Making predictions on the train and test data
dt_sampled_train_preds = sampled_DT.predict(X_sampled_train)
dt_sampled_test_preds = sampled_DT.predict(X_sampled_test)
```

Figure 31: Initialising and fitting, making predictions with resampled data on Random Forest.



```python
# Evaluation scores for Train set
mse = round(mean_squared_error(y_sampled_train, dt_sampled_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_train, dt_sampled_train_preds), 4)
r2 = round(r2_score(y_sampled_train, dt_sampled_train_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 32: Calculating evaluations of ADASYN - Random Forest classifier on the train set, and returning them.

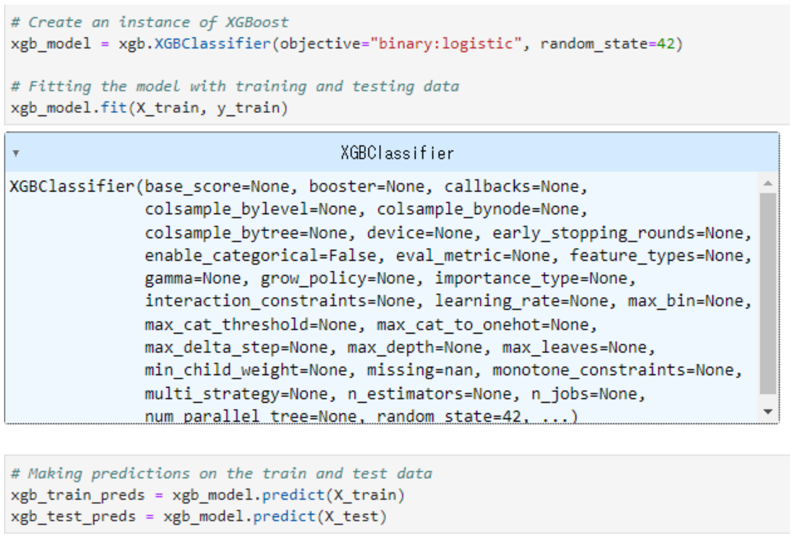


```python
# Evaluation scores for Test set
mse = round(mean_squared_error(y_sampled_test, dt_sampled_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_test, dt_sampled_test_preds), 4)
r2 = round(r2_score(y_sampled_test, dt_sampled_test_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 33: Calculating evaluations of ADASYN - Random Forest classifier on the test set, and returning them.

2.4 XGBoost

Extreme gradient boosting is another machine learning algorithm that specialises in supervising learning tasks that we have used. It consists of a combination of decision trees in a way that optimises its predictive accuracy. It incorporates regularisation techniques like L1 and L2 to avoid overfitting and control the complexity of the trees. It also uses cross validation.

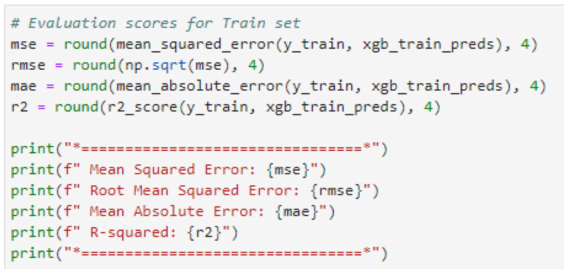


```python
# Create an instance of XGBoost
xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)

# Fitting the model with training and testing data
xgb_model.fit(X_train, y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=42, ...)
```

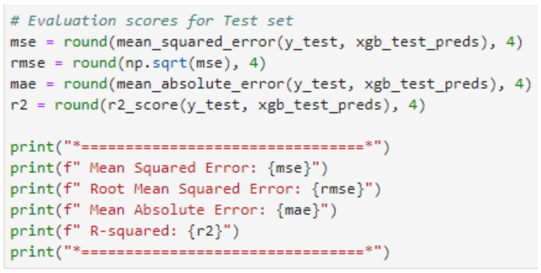
```python
# Making predictions on the train and test data
xgb_train_preds = xgb_model.predict(X_train)
xgb_test_preds = xgb_model.predict(X_test)
```

Figure 34: Initialising and fitting, making predictions with XGBoost.



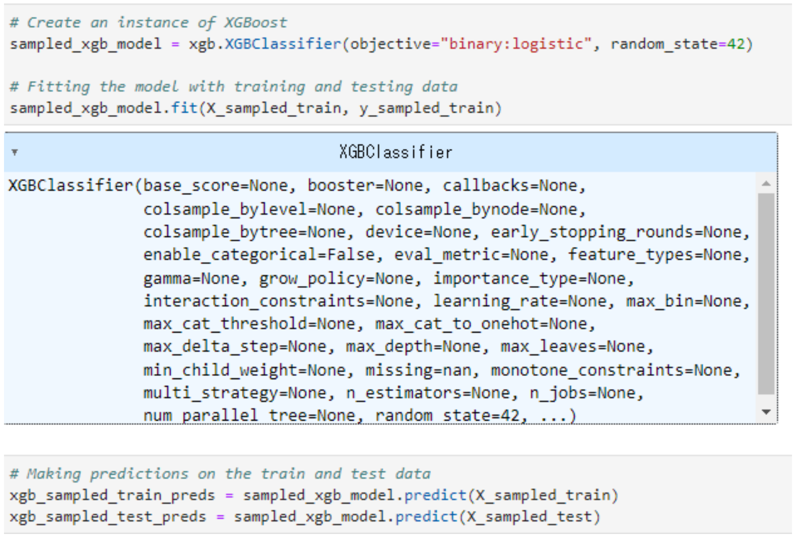
```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_train, xgb_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_train, xgb_train_preds), 4)
r2 = round(r2_score(y_train, xgb_train_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 35: Calculating evaluations with XGBoost on the train set, and returning them.



```
# Evaluation scores for Test set
mse = round(mean_squared_error(y_test, xgb_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_test, xgb_test_preds), 4)
r2 = round(r2_score(y_test, xgb_test_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 36: Calculating evaluations with XGBoost on the test set, and returning them.

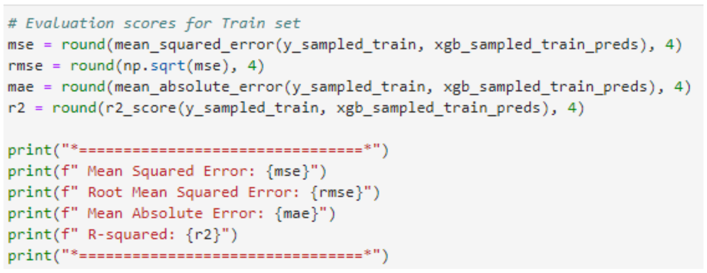


```
# Create an instance of XGBoost
sampled_xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)

# Fitting the model with training and testing data
sampled_xgb_model.fit(X_sampled_train, y_sampled_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=42, ...)
```

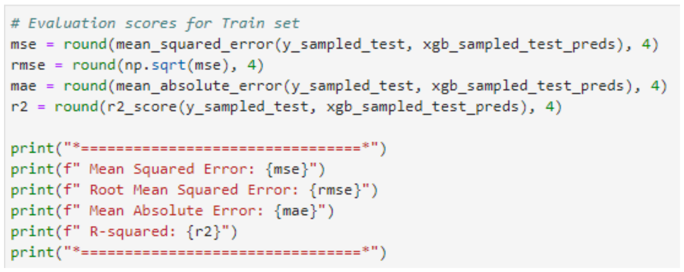
```
# Making predictions on the train and test data
xgb_sampled_train_preds = sampled_xgb_model.predict(X_sampled_train)
xgb_sampled_test_preds = sampled_xgb_model.predict(X_sampled_test)
```

Figure 37: Initialising and fitting, making predictions with resampled data on XGBoost.



```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_sampled_train, xgb_sampled_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_train, xgb_sampled_train_preds), 4)
r2 = round(r2_score(y_sampled_train, xgb_sampled_train_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 38: Calculating evaluations of ADASYN - XGBoost on the train set, and returning them.

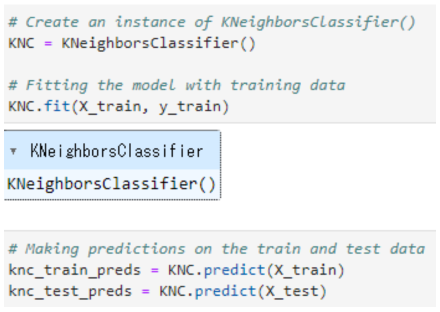


```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_sampled_test, xgb_sampled_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_test, xgb_sampled_test_preds), 4)
r2 = round(r2_score(y_sampled_test, xgb_sampled_test_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 39: Calculating evaluations of ADASYN - XGBoost on the test set, and returning them.

2.5 K-Nearest Neighbors (KNN)

A straightforward technique for classification and regression, K-Nearest Neighbours (KNN) bases predictions on the average of the K-nearest data points in the training set or the majority class. Although simple to use, it is dependent on K and the selected distance metric.

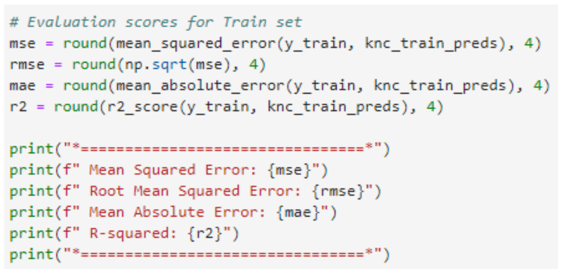


```
# Create an instance of KNeighborsClassifier()
KNC = KNeighborsClassifier()

# Fitting the model with training data
KNC.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

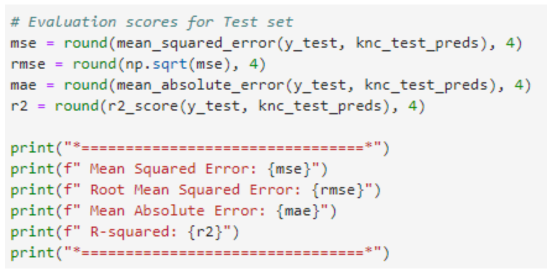
```
# Making predictions on the train and test data
knc_train_preds = KNC.predict(X_train)
knc_test_preds = KNC.predict(X_test)
```

Figure 40: Initialising and fitting, making predictions with K-Nearest Neighbors.



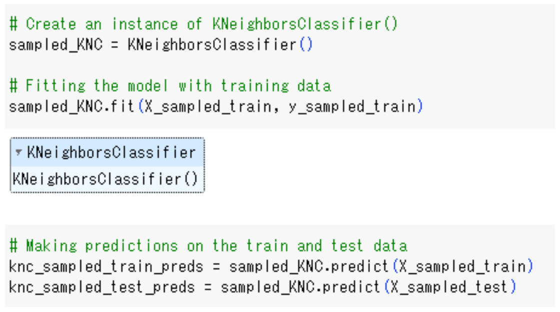
```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_train, knc_train_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_train, knc_train_preds), 4)
r2 = round(r2_score(y_train, knc_train_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 41: Calculating evaluations with K-Nearest Classifier on the train set, and returning them.



```
# Evaluation scores for Test set
mse = round(mean_squared_error(y_test, knc_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_test, knc_test_preds), 4)
r2 = round(r2_score(y_test, knc_test_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 42: Calculating evaluations with K-Nearest Classifier on the test set, and returning them.

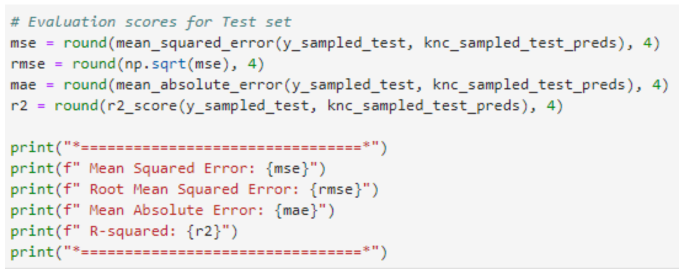


```
# Create an instance of KNeighborsClassifier()
sampled_KNC = KNeighborsClassifier()

# Fitting the model with training data
sampled_KNC.fit(X_sampled_train, y_sampled_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

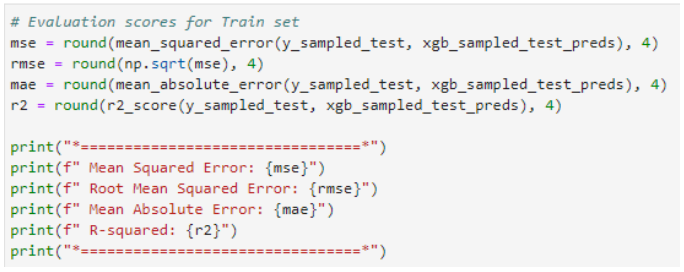
```
# Making predictions on the train and test data
knc_sampled_train_preds = sampled_KNC.predict(X_sampled_train)
knc_sampled_test_preds = sampled_KNC.predict(X_sampled_test)
```

Figure 43: Initialising and fitting, making predictions with resampled data on XGBoost.



```
# Evaluation scores for Test set
mse = round(mean_squared_error(y_sampled_test, knc_sampled_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_test, knc_sampled_test_preds), 4)
r2 = round(r2_score(y_sampled_test, knc_sampled_test_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 44: Calculating evaluations of ADASYN - XGBoost on the train set, and returning them.

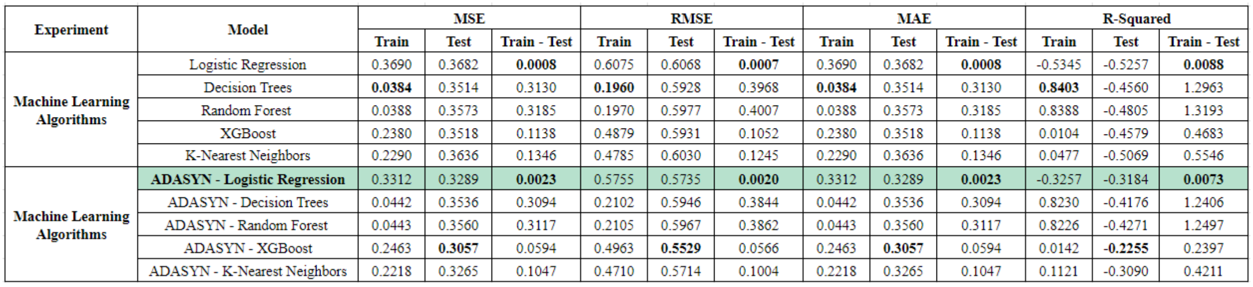


```
# Evaluation scores for Train set
mse = round(mean_squared_error(y_sampled_test, xgb_sampled_test_preds), 4)
rmse = round(np.sqrt(mse), 4)
mae = round(mean_absolute_error(y_sampled_test, xgb_sampled_test_preds), 4)
r2 = round(r2_score(y_sampled_test, xgb_sampled_test_preds), 4)

print("*================================*")
print(f" Mean Squared Error: {mse}")
print(f" Root Mean Squared Error: {rmse}")
print(f" Mean Absolute Error: {mae}")
print(f" R-squared: {r2}")
print("*================================*")
```

Figure 45: Calculating evaluations of ADASYN - XGBoost on the test set, and returning them.

Results

Modelling Results



Experiment	Model	MSE			RMSE			MAE			R-Squared		
		Train	Test	Train - Test	Train	Test	Train - Test	Train	Test	Train - Test	Train	Test	Train - Test
Machine Learning Algorithms	Logistic Regression	0.3690	0.3682	**0.0008**	0.6075	0.6068	**0.0007**	0.3690	0.3682	**0.0008**	-0.5345	-0.5257	**0.0088**
	Decision Trees	**0.0384**	0.3514	0.3130	**0.1960**	0.5928	0.3968	**0.0384**	0.3514	0.3130	**0.8403**	-0.4560	1.2963
	Random Forest	0.0388	0.3573	0.3185	0.1970	0.5977	0.4007	0.0388	0.3573	0.3185	0.8388	-0.4805	1.3193
	XGBoost	0.2380	0.3518	0.1138	0.4879	0.5931	0.1052	0.2380	0.3518	0.1138	0.0104	-0.4579	0.4683
	K-Nearest Neighbors	0.2290	0.3636	0.1346	0.4785	0.6030	0.1245	0.2290	0.3636	0.1346	0.0477	-0.5069	0.5546
Machine Learning Algorithms	**ADASYN - Logistic Regression**	0.3312	0.3289	**0.0023**	0.5755	0.5735	**0.0020**	0.3312	0.3289	**0.0023**	-0.3257	-0.3184	**0.0073**
	ADASYN - Decision Trees	0.0442	0.3536	0.3094	0.2102	0.5946	0.3844	0.0442	0.3536	0.3094	0.8230	-0.4176	1.2406
	ADASYN - Random Forest	0.0443	0.3560	0.3117	0.2105	0.5967	0.3862	0.0443	0.3560	0.3117	0.8226	-0.4271	1.2497
	ADASYN - XGBoost	0.2463	**0.3057**	0.0594	0.4963	**0.5529**	0.0566	0.2463	**0.3057**	0.0594	0.0142	**-0.2255**	0.2397
	ADASYN - K-Nearest Neighbors	0.2218	0.3265	0.1047	0.4710	0.5714	0.1004	0.2218	0.3265	0.1047	0.1121	-0.3090	0.4211

Figure 46: Evaluation Scores on each of machine learning algorithms.

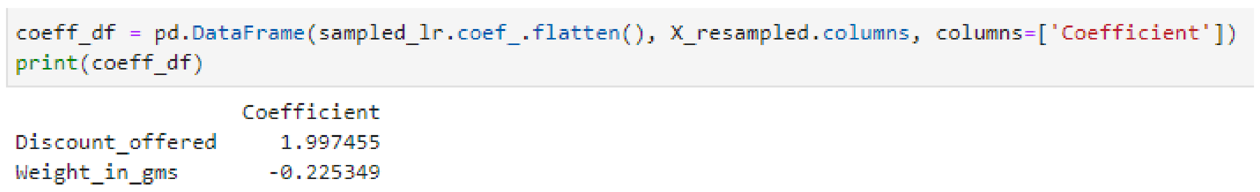
We could gain below insights from the summary of outputs in the table on above figure:

- Although **ADASYN - XGBoost** has the lowest MAE and MSE on the test set, it performs worse than a simple mean predictor, as seen by its negative R^2 value.
- Among the models with ADASYN applied, **ADASYN - XGBoost** has the greatest R^2 and appropriate MSE and MAE scores. After ADASYN was applied, the initial model's negative R^2 improved to -0.2255, which is still negative but shows a better match than any other ADASYN models.
- Although the R^2 of the **ADASYN Decision Tree** is higher than that of the **non-ADASYN Decision Tree**, it is still negative, indicating poor generalisation of the model.
- Both with and without ADASYN, **Random Forest**'s R^2 value is comparatively lower than **Decision Trees'**. This could be a sign of overfitting in the Random Forest model that ADASYN helped to mitigate.
- In terms of R^2 on the test set, the non-ADASYN models typically perform badly; negative values denote poor prediction quality.
- Both with and without ADASYN, Logistic Regression shows extremely good balance of overall train and test set results. ADASYN - Logistic regression and logistic regression do not differ that much in the Train-Test gap (non-ADASYN Logistic Regression is merely better), while evaluation scores are generally better for ADASYN - Logistic regression.

It may appear that the **ADASYN - XGBoost** has the lowest test set errors if we disregard the R^2 values, which are unreliable when they are negative, and simply concentrate on MSE, RMSE, and MAE. Alternatively, the **ADASYN - Logistic Regression** is the best when taking into account the evaluation score balance between the test and train sets, while considering evaluation scores.

Since balanced evaluation scores between training and test sets are often chosen since they imply a model that would perform reliably in production, we would base our model selection on the word "balance." Consequently, we would select **ADASYN - Logistic Regression** as the optimal machine learning model for the prediction.

Result of Feature Importance determines the most influential feature



```
coeff_df = pd.DataFrame(sampled_lr.coef_.flatten(), X_resampled.columns, columns=['Coefficient'])
print(coeff_df)
```

```
                 Coefficient
Discount_offered    1.997455
Weight_in_gms      -0.225349
```

Figure 47: Outcomes of coefficients for each feature.

Though a variety of methods are employed for the feature importance in machine learning, the best way on Logistic Regression is to compare the coefficients of each feature. According to the coefficient extraction implemented with above source code, Discount_offered has the highest coefficient of 1.997455, compared to the Weight_in_gms of -0.225349. The result indicates that the Discount_offered feature has more impact on Shipping Arrival on Time than the variable “Weight_in_gms”.

Conclusion

The purpose of our study is to introduce the idea of utilising Machine Learning algorithms and data science principles in order to aid in more accurate estimations of shipment arrival times in terms of the e-commerce sector. In relation to this, we decided to address the underlying issue of inefficient data analysis in the sector due the abundance of its existence. We then proceeded to do some literature reviews which tackle similar problem statements as ours. After doing the required research, we then studied the opportunities we could use to continue this study successfully, and by building a prediction model which would have the potential to provide accurate estimations of shipment arrival times to aid the growth of the business industry. We tested and compared Machine Learning algorithms such as Logistic Regression, Decision Tree, Random Forest, XGBoost and K-Nearest Neighbours, including the base and resampled models in order to identify the model with best performance.

For our results when testing our Machine learning programs with ADASYN to fix imbalanced data as well as without ADASYN. We found that ADASYN Logistic Regression performed the best for prediction of package deliveries that were on time. It showed a balanced result in both testing and training sets. This makes it the most reliable for Real world application. Other models have issues with potential overfitting or negative R-squared values. We also determined the feature importance by comparing the feature coefficients. In our analysis we found that "Discount_Offered" had the highest coefficient of 1.99745, indicating a significant impact on predicting the shipping time or arrival.

Recommendation

According to our findings, immediate improvements should concentrate on the accuracy and openness of delivery schedules, as well as the strengthening of consumer loyalty. Customers may be supplied with real-time information on their shipments by incorporating an expected arrival time function, considerably boosting transparency and confidence. This enhancement is designed to improve the customer experience by matching delivery expectations with the actual service. Furthermore, creating a credit point system to compensate for service gaps not only develops goodwill but also serves as an incentive for return business. Such methods can boost consumer satisfaction in the near run, lowering the likelihood of complaints and returns.

Long term, it is critical to expand our system's analytical capabilities. The inclusion of additional dataset variables, such as customer demographic data or previous delivery performance indicators, will allow for more nuanced insights and forecast accuracy. This would not only simplify decision-making but will also allow services to be tailored to particular client demands. In addition, a monthly operational audit based on the enhanced insights from the increased dataset will ensure continual development and alignment with best practices. This proactive approach to auditing will aid in detecting possible inefficiencies and correcting them before they become major problems, thus preserving the integrity of our operating standards.

It could also be advised to use some additional methods for optimising machine learning models. Hyperparameter tuning is a widely used technique that involves using GridSearchCV to identify the optimal hyperparameters for each machine learning algorithm and then applying those findings to the model in order to potentially obtain more relevant and optimised results from trials.

By putting these ideas into practice, we want to build a strong foundation for ongoing operational excellence in addition to earning our clients' initial trust.

References

1. Abdul Hussien, F. T., S. Rahma, A. M., & Abdul Wahab, H. B. (2021). *Recommendation Systems For E-commerce Systems An Overview*. IOP Science. https://iopscience.iop.org/article/10.1088/1742-6596/1897/1/012024/pdf

2. Al-Saghir, R. (2022). *Predicting Delays in the Supply Chain with the Use of Machine Predicting Delays in the Supply Chain with the Use of Machine Learning Learning*. https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=12627&context=theses

3. Bao, Y., & Datta, A. (2014). Simultaneously Discovering and Quantifying Risk Types from Textual Risk Disclosures. *Management Science*, *60*(6), 1371–1391. https://pubsonline.informs.org/doi/10.1287/mnsc.2014.1930

4. Chen, H., Chiang, R. H. L., & Storey, V. C. (2022). Business Intelligence and Analytics: From Big Data to Big Impact. *MIS Quarterly*, *36*(4), 1165. https://www.jstor.org/stable/41703503

5. CLOUD 7 IT SERVICES INC. (2023, September 5). Big Data in e-commerce: Personalization and recommendations. LinkedIn. https://www.linkedin.com/pulse/big-data-e-commerce-personalization-recommendations

6. Huang, M.-H., & Rust, R. T. (2018). Artificial Intelligence in Service. *Journal of Service Research*, *21*(2), 155–172. https://journals.sagepub.com/doi/10.1177/1094670517752459

7. Huang, Z., & Benyoucef, M. (2013). From e-commerce to social commerce: A close look at design features. *Electronic Commerce Research and Applications*, *12*(4), 246–259. https://www.sciencedirect.com/science/article/abs/pii/S156742231200124X?via%3Dihub

8. Jones, A., & Barrett, A. (2022, July 21). 6 benefits of Big Data Analytics for E-Commerce. Octoparse. https://www.octoparse.com/blog/benefits-of-big-data-analytics-for-e-commerce

9. Jonquais, A., Krempl, F., Adland, R., & Jia, H. (n.d.). *Predicting Shipping Time with Machine Learning*. Retrieved November 5, 2023, from https://ctl.mit.edu/sites/ctl.mit.edu/files/theses/43819602-Executive%20Summary%20Kr

empl%20Jonquais.pdf Millimetric . (2022, March 14). *The 9 Biggest Big Data Challenges For Ecommerce Directors*. Millimetric.ai. https://millimetric.ai/blog/the-9-biggest-big-data-challenges-for-ecommerce-directors/

10. Kusumawati, K. N. (2021, August 22). *Estimated Time Delivery Prediction (Study Case: Ecommerce Dataset)*. Medium. https://medium.com/@katarinanimas20/estimated-time-delivery-prediction-study-case-ecommerce-dataset-fa71adbc2398

11. Ministry of Communications and Digital. (n.d.). *MDEC's Commissioned Study Shows Malaysia's Big Data Analytics Market Expected To Grow To US$1.9b by 2025*. Kkd.gov.my. https://www.kkd.gov.my/en/public/news/18969-mdec-s-commissioned-study-shows-malaysia-s-big-data-analytics-market-expected-to-grow-to-us-1-9b-by-2025

12. Monash University Malaysia. (2023, January 11). *How data science and analytics are boosting Malaysia's digital economy*. Malaysia. https://www.monash.edu.my/news-and-events/trending/how-data-science-and-analytics-are-boosting-malaysias-digital-economy#:~:text=Malaysia

13. Salari, N., Liu, S., & Shen, Z.-J. M. (2020, September 28). *Real-Time Delivery Time Forecasting and Promising in Online Retailing: When Will Your Package Arrive?* Papers.ssrn.com. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3701337

14. Smith, N. (2022, September 14). *How Big Data Can Transform the eCommerce Industry? | Socialnomics*. Socialnomics. https://socialnomics.net/2022/09/14/top-5-ways-big-data-can-transform-the-ecommerce-landscape/

15. Zhang, Y. (2015). Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance*, *15*(8), 1315–1329. https://www.tandfonline.com/doi/full/10.1080/14697688.2015.1032546