LATE DELIVERY RISK ANALYSIS ON GENERAL PRODUCTS USING MACHINE LEARNING METHODS (CLASSIFICATION)

Author: *Satyam Sharma*

ABSTRACT

The aim of this project was to research and develop a machine learning model for predicting the likelihood of a late delivery. I used the DataCo Supply chain dataset available on Kaggle to employ the usage of any three classification methods in python on the dataset and the classification methods used were: Logistic Regression, Gaussian Naïve Bayes, Support Vector Classifier and Decision Tree Classifier. The systematic approach carried included data processing and cleaning, exploratory data analysis (EDA), data preparation for model building, model development, model optimization and conclusion

TABLE OF CONTENTS

INTRODUCTION:

For many manufacturing businesses, getting products to customers in the least amount of time is a difficulty in the current market environment. (Gudum, 2002).

The complexity of a retail-oriented market with a wide range of products mostly results to untimely delivery of purchased products and as such maintaining customer satisfaction and market competitiveness has also faced a downward slope in terms of market success. This team realized the above-mentioned problem and with our research have attempted a solution. Employing machine learning methods and algorithm (classification) to build a predictive model to analyze and possibly curb late deliveries in this market.

METHODOLOGY:

I adopted the KDD (Knowledge Discovery Database) Methodologies which is an iterative process that transforms raw data into useful information. Different steps involved in the KDD Methodology are:

- Data Collection
- Pre-processing and exploring the data
- Data Preparation
- Model Training
- Evaluating Model Performance
- Improving Model Performance

DATA COLLECTION

In this process, I sourced for the data source to be used for our case study whoselink is shown below:

Dataset link: https://data.mendeley.com/datasets/8gx2fvg2k6/5

This is the dataset of Supply Chains used by the company DataCo Global which includes a collection of their products sold, financial details (profit, loss, total sales etc.), Shipping details, and customer details such as sales, demographics, and transaction details.

For this project, I started by importing all the necessary libraries to be used as shown in Fig 1. Subsequently, I began creating functions that would be used for theduration of the project which were shown in the Fig 2-10.

The dataset consists of 180,519 data rows and 53 columns with information about customer details, sales transactions, order and shipping information and many more. A separate metadata file is also provided which has metadata of the various columns in the actual data file.

The tabular dataset (*Appendix - Sample Dataset 1*) was loaded using the ***pd.read_csv()*** function in the first phase of data exploration. The shape function shows the number of columns and rows contained in the dataset. Using the ***head()*** function a brief overview of the complete dataset was deduced.

IMPORTANT LIBRARIES AND PACKAGES

```
import pandas as pd
import numpy as np
import seaborn as sns
import time
import re
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn import model_selection
from sklearn.model_selection import train_test_split, KFold,cross_val_score,GridSearchCV,RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,recall_score,confusion_matrix,f1_score,classification_report
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn import svm

#Hiding the warnings
import warnings
warnings.filterwarnings('ignore')
```

Fig 1

```
[21]
#Function to find features with missing values in the dataset
def Missing_Values_Analysis(data):
    #find percentage of missing data
    count_missing_datapoints = data.isnull().sum().sum()
    total_datapoints = data.shape[0]*data.shape[1]
    missing_value_percentage = (count_missing_datapoints/total_datapoints)*100
    print(f"Total missing values in {total_datapoints} datapoints is {count_missing_datapoints} which is {round(missing_value_percentage,2)}% of total data")

    #find features with missing values
    for i in data.columns:
      missing_values = data[i].isna().sum()
      if  missing_values > 0:
        print(f"Feature '{i}' has {missing_values} missing values")
```

Fig 2

```
[22]
#Function to fill na values
def Fill_Missing_Values(data,cols,custom_value = 0):
  for c in cols:
    data[c] = data[c].fillna(custom_value)

  return data
```

Fig 3

```
#Function to format the headers by removing unnecessary spaces and special characters to make the dataset more understandable
def Format_Headers(cols):
    formatted_cols = []
    for col in cols:
        # Strip unnecessary spaces
        col = col.strip()

        # Convert to lowercase
        col = col.lower()

        # Remove special characters using regex
        col = re.sub(r'[^\w\s]', '', col)

        # Replace spaces with underscore
        col = col.replace(" ","_")

        formatted_cols.append(col)

    return formatted_cols
```

Fig 4

```python
#Function to find duplicate columns in the dataset
def find_duplicate_columns(data):
    duplicate_columns = set()
    for i in range(len(data.columns)):
        for j in range(i + 1, len(data.columns)):
            if data.iloc[:, i].equals(data.iloc[:, j]):
                duplicate_columns.add(data.columns[i])
                duplicate_columns.add(data.columns[j])
                print(f"{data.columns[j]} is a duplicate of column {data.columns[i]}")
    return list(duplicate_columns)
```

Fig 5

```python
#Function to plot correlation heatmap for all the variables in the dataset
def corr_plot(data):
  corrmap = data.corr()
  top=corrmap.index
  plt.figure(figsize=(30,20))
  g=sns.heatmap(data[top].corr(),annot=True,cmap="RdYlGn")
```

Fig 6

```python
#Function to perform label encoding to categorical variables
def label_Encoder(data,features):
  le = preprocessing.LabelEncoder()
  for feature in features:
    data[feature] = le.fit_transform(data[feature])

  return data
```

Fig 7

```python
#Function to apply Classification Models with Hyper-parameter tuning
def Classification_Model(model,X_train, X_test, y_train, y_test,params=0,search_type=""):
  if search_type == "grid":
            clf = GridSearchCV(estimator=model, param_grid=params, scoring=None,
                n_jobs=-1, cv=5, verbose=0,return_train_score=True)
  elif search_type == "random":
            clf = RandomizedSearchCV(estimator=model,param_distributions=params,n_iter=10,
                n_jobs=-1,cv=5,verbose=0,random_state=1,return_train_score=True)
  else:
    clf = model
# Fit the model
  clf.fit(X_train,y_train)
  y_pred = clf.predict(X_test)
  res = []
  res.append(clf)
  res.append(str(clf))

  accuracy_l=accuracy_score(y_pred, y_test) #Accuracy for predection of late delivery
  res.append(round(accuracy_l*100,2))
  recall_l=recall_score(y_pred, y_test)# Recall score for predection of late delivery
  res.append(round(recall_l*100,2))
  conf_l=confusion_matrix(y_test, y_pred)#predection of late delivery
  f1_l=f1_score(y_test, y_pred)#predection of late delivery
  res.append(round(f1_l*100,2))
  print('Model paramters used are :',clf)
  print('Accuracy of late delivery risk analyzer is:', (accuracy_l)*100,'%')
  print('Recall score of late delivery risk analyzer is:', (recall_l)*100,'%')
  print('Conf Matrix of late delivery risk analyzer is: \n',(conf_l))
  print('F1 score of late delivery risk analyzer is:', (f1_l)*100,'%')
  print('Classification Report for late delivery risk analyzer is:\n',classification_report(y_test, y_pred))

  return res
```

Fig 8

```python
#Function to apply cross validation on Gaussian Naive Bayes Classifier
def GNB_CrossVal(X,y,folds):
    gnb = GaussianNB()
    scores = cross_val_score(gnb, X, y, cv=folds, scoring='accuracy')

    print("Cross-validation scores:", scores)
    print("Mean accuracy:", scores.mean())
```

Fig 9

```python
#Function to plot feature important graph of the dataset
def Check_Important_Feature(clf,X):
    #Feature importance
    important_col=clf.feature_importances_.argsort()
    feat_imp=pd.DataFrame({'features':X.columns[important_col],'importance':clf.feature_importances_[important_col]})
    feat_imp=feat_imp.sort_values(by='importance',ascending=False)
    ax = sns.catplot(x='features', y = 'importance', data=feat_imp, height=5, aspect=2,  kind="bar")
    plt.xticks(rotation=90)
```

Fig 10

DATA PRE-PROCESSING AND EXPLORATION

In this process, the first operation carried out was to obtain the information about the data set using the ***info()*** function which showed that the dataset has 53 columns and 180519 rows, *Product Description* field is empty for all the records, *Order Zipcode* and *Customer Lname* has null values and finally, all the column names were not properly formatted as some fields contained parenthesis and spaces.

The second operation carried was the ***describe()*** function which shows *Late delivery risk* field is binary (0 or 1) which means that the orders which have *Days of shipping (real)* greater than the *Days of Shipment (scheduled)* are marked as 1 and rest are marked as 0. I can also noticed that the mean of late delivery is more than 0.5 which means that there are more orders that were delivered late compared to on time deliveries. It can be concluded the business is experiencing loss due to late deliveries and need to understand the root cause and develop a predictive model to flag potential late deliveries for future and support better decision making.

MISSING VALUE ANALYSIS

The next step was to perform **Missing Value Analysis** which showed us that the total datapoints is 9567507 and a total of 336209 datapoints were missing which is 3.51% of the total. I observed that there are 4 fields which have missing values namely ***Customer Lname* (8)**, ***Customer Zipcode* (3)**, ***Order Zipcode* (155679)** and **Product Description (180519)**. Since ***Order Zipcode*** and ***Product Description*** are null for more than 50% of the dataset, I assumed that they will not be relevant for the model development and were dropped from the list of features. Since ***Customer Last Name***

and ***Customer Zipcode*** have missing values for less than 0.001% of the data, where the missing values can be imputed either by using some default values like "NA", 0 or rows with NA values can be dropped since the percentage of missing data is very less.

HANDLING MISSING VALUES

Since ***Product Description*** has all null values and ***Order Zipcode*** has more than 80% null values so we can drop these features as they are not significant for our analysis using the ***drop()*** function. Also since only 3 datapoints have missing value for ***Customer Zipcode,*** I used ***Fillna()*** function to fill the missing value with 0. Since there is a chance different customers might have the same first name or same last name a new column with the ***Customer Full Name*** was created to avoid any ambiguities. The next step was to check for any duplicate record using the ***duplicated().sum()*** function and it showed that there were no duplicates present in thedataset. The next step was to use the ***format_Header()*** function to appropriately space the fields using underscores(_).

PERFORMING EXPLORATORY DATA ANALYSIS

In this process, I analyzed the correlation plot to check the correlation between the different attributes by calling the ***corr_plot()*** function as shown in Fig 11 and from the correlation plot I saw that there are many columns which has very high correlation and upon further analysis they seem to have duplicate values, so I found the columns present in the dataset by calling the ***find_duplicate_column()*** function and dropped them using the ***drop()*** function. after dropping the unwanted columns, I plotted a chart to compare ***delivery status*** against other fields like ***Number of orders***as shown in Fig below.

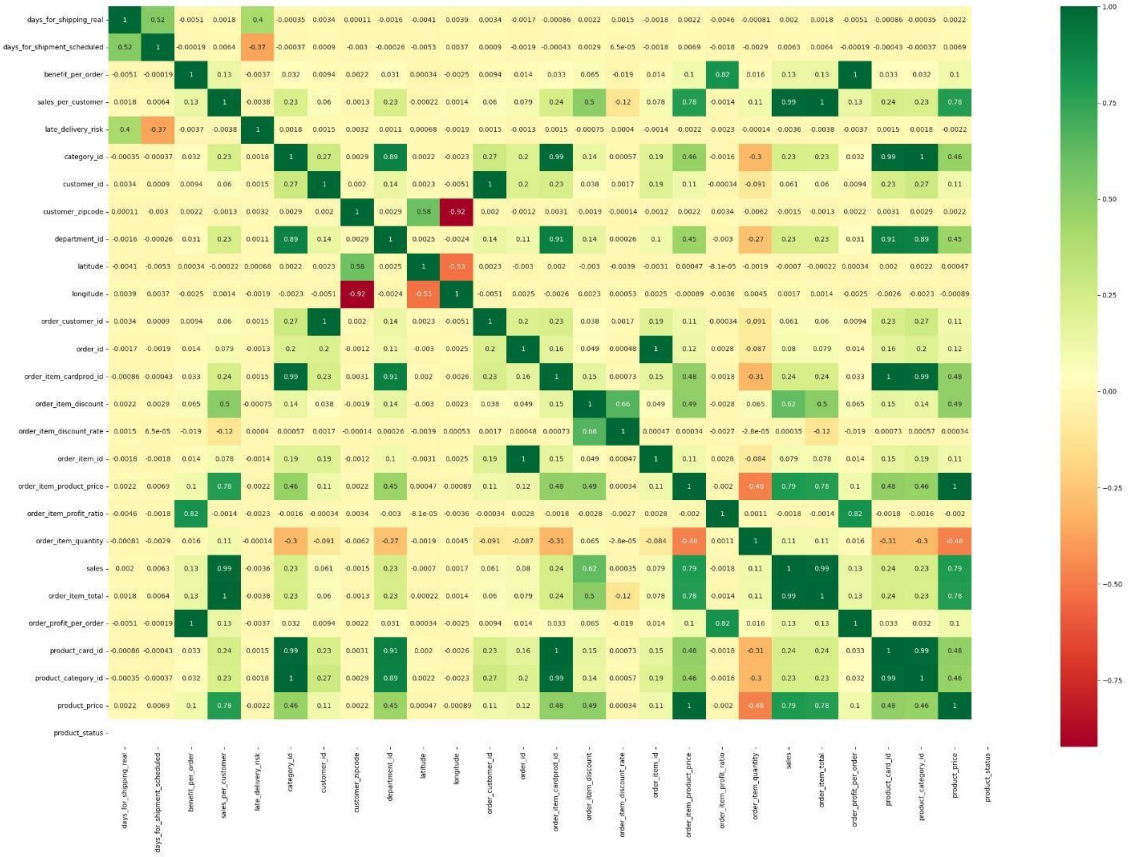

Fig 11

Fig 12

Inference: From the plot I can see that there are more than 50% orders with ***Delivery Status*** as ***Late Delivery***. This makes it imperative to develop machine

learning models to classify orders with high probability of late deliveries (**Late Delivery Risk Analysis**)

I also plotted a chart of ***Number of Orders*** against ***Customer_Country*** to show the ***Delivery Status*** as shown in Fig below:

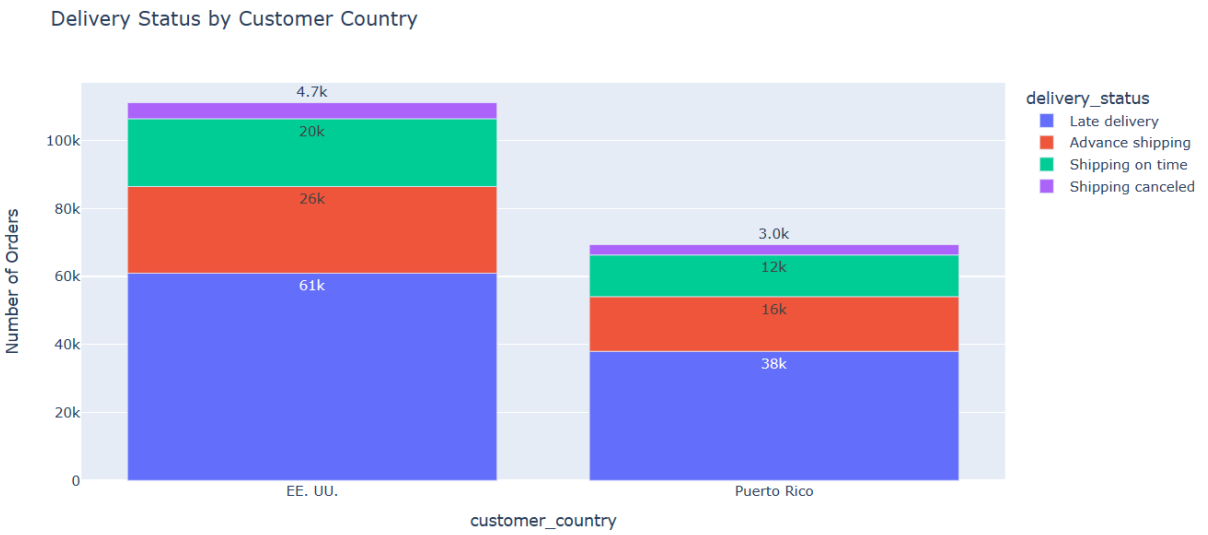

Fig 13

I also plotted a chart of ***Number of Orders*** against ***Market*** to show the ***Delivery Status*** as shown in Fig below:

I also plotted a chart of ***Number of Orders*** against ***Order_Region*** to show the ***Delivery Status*** as shown in Fig below:

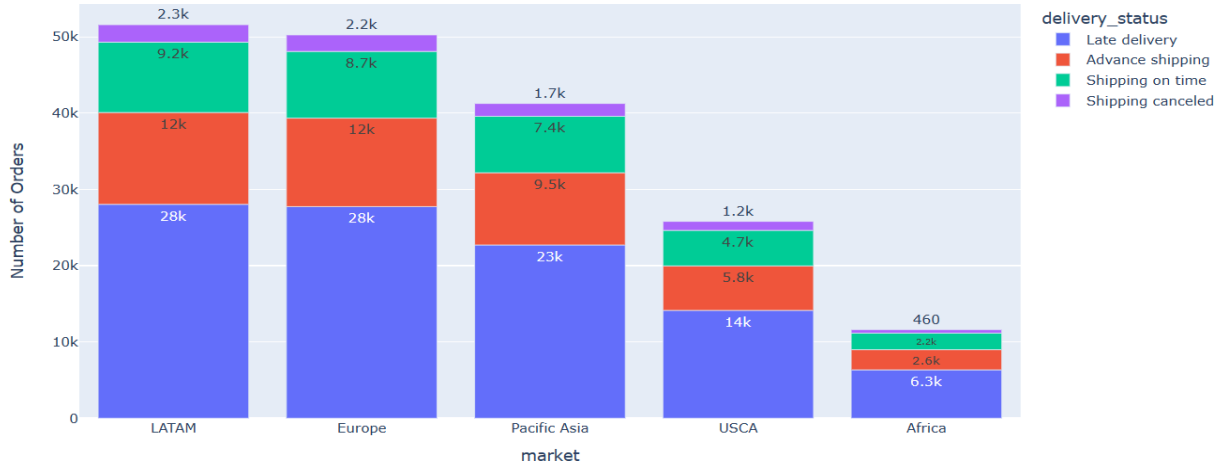

Fig 14

I plotted a pie-chart showing the late deliveries from Customer Segments as shownin the Fig Below

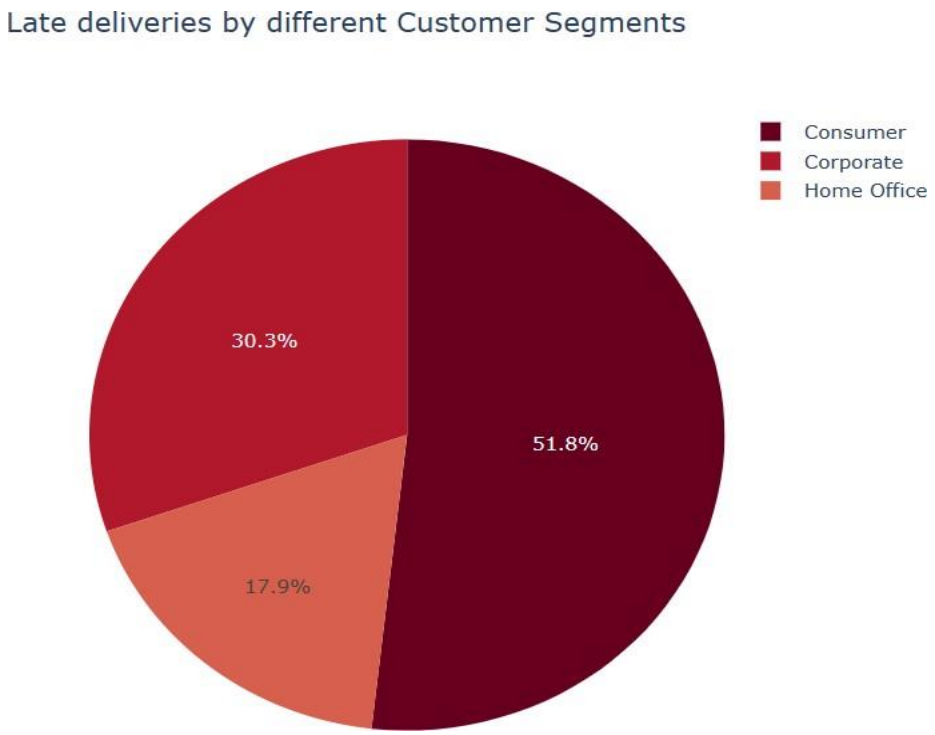

Fig 15

Inference: Consumer segment has the most late deliveries with 51.8% of orders delivered late.

I also plotted a chart of ***Number of Orders*** against ***Shipping_Modes*** to show the ***Delivery Status*** as shown in Fig below:

Fig 16

Inference: From above chart we can visualize that **First Class** and **Second Class** have more than 70% of late deliveries while **Standard Class** have most late deliveries in terms of numbers. This means that in terms of ratio of late deliveries within each group, **First class** has most late deliveries followed by **Second Class** and **Standard Class**

From this we can conclude late delivery is a major problem associated with **First and Second classes** of Shipping modes

I plotted a pie-chart showing the late deliveries by different ***Shipping Modes*** asshown in the Fig below

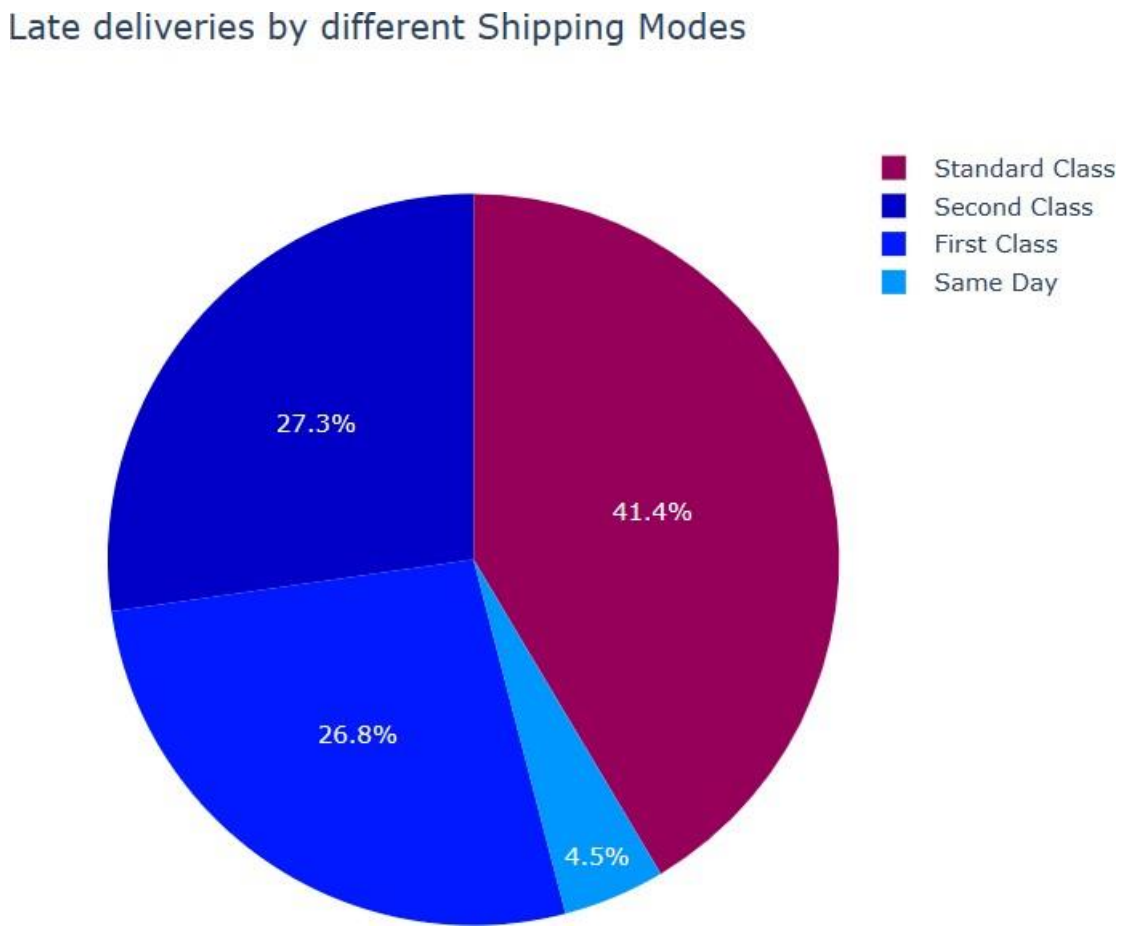

Fig 17

Inference: From above chart we can see that 41% of the late deliveries are from **Standard Class** but in terms of ratio **First Class** and **Second Class** have very high late deliveries

DATA PREPARATION AND FEATURE ENGINEERING

After performing the EDA on the given dataset, I prepared the data and perform Feature Engineering on the data, this occurs as a data transformation step where data is converted from its current state to a format suitable for models to run. Following activities were performed as part of data preparation and feature engineering:

- The columns ***order_date_dateorders*** and ***shipping_date_dateorders*** got converted to datetime format using ***pd.to_datetime()*** function. The Day, month, and year were gathered from the datetime columns and stored in new columns as follows (***'order_day', 'order_month', 'order_year', 'shipping_day', 'shipping_month', 'shipping_year'***). Once done then the original columns for datetime were dropped from the dataframe using ***df.drop().***
- Perform Feature engineering by calculating the abs difference between ***days_for_shipping_real*** and ***days_for_shipment_scheduled*** columns and then storing the resulting data in a new column named ***days_difference*** and now the dataframe has 46 columns.
- Some features which do not contribute to the final model of late delivery risk were dropped from the final features list. Some of these features can be ***order_id***, ***product_card_id***, etc. The shape of the Dataframe after this step was changed to (180519, 30), showing that the number of rows remains the same, but the number of columns has been lowered to 30
- I attempted at applying one-hot encoding to the categorical data as there wasno visible ordinality in the data but since it creates a sparse dataframe with more than 5000 columns making it difficult for the models to interpret the data. Later, Label encoding was applied to all categorical columns with object data type to convert them to into numerical datatype, I use the ***label_Encoder()*** function which was defined in Fig 7.
- The variable 'X' is created by dropping the column ***late_delivery_risk*** from the dataframe **df_final_features** using drop() method & variable ‘y’ is created from ***late_delivery_risk*** column. The ‘X’ and ‘y’ variables are used in machine learning model tasks, where ‘X’ represents the input variables, and ‘y’ represents the target variable.
- I finally performed the split of the dataset into training and testing sets (X_train, X_test, y_train, y_test) with 80% of the data split for training and 20%split for testing.

MODEL DEVELOPMENT AND TRAINING

For this section I defined dependent features and the problem type, and I deduced them as ***late_delivery_risk*** and Classification respectively. The solution is attempted to build a classification model to predict late delivery risk for future deliveries based on various input features like order country, shipping mode, etc.

For making the model development and execution simpler, I created a custom function to train and test the dataset on any classification model and also perform hyper-parameter tunning if required. I used the function **Classification_Model()** which is defined in Fig 8

APPLICATION OF LOGISTIC REGRESSION:

Initially Logistic Regression was applied on the non-scaled dataset to check the accuracy of trained model. It was observed that only **55%** accuracy was achieved with non-scaled training data.

Apply Logistic Regression on non-scaled data

```
[ ]  1 lr = LogisticRegression()
     2 lr_clf = Classification_Model(lr,X_train,X_test,y_train,y_test)
     3 result.append(lr_clf)
```

```
Model paramters used are : LogisticRegression()
Accuracy of late delivery risk analyzer is: 55.118546421449146 %
Recall score of late delivery risk analyzer is: 55.56760777264698 %
Conf Matrix of late delivery risk analyzer is:
 [[ 1970 14337]
 [ 1867 17930]]
F1 score of late delivery risk analyzer is: 68.87676705593117 %
Classification Report for late delivery risk analyzer is:
              precision    recall  f1-score   support

           0       0.51      0.12      0.20     16307
           1       0.56      0.91      0.69     19797

    accuracy                           0.55     36104
   macro avg       0.53      0.51      0.44     36104
weighted avg       0.54      0.55      0.47     36104
```

Fig 18

Next, I tried to fine tune the hyper-parameters of the model by performing hyper-parameter tuning using GridSearchCV and RandomSearchCV which helped the model improve the accuracy by just 1 or 2%.

Logistic Regression with GridSearchCV optimizer

```
[ ]  1 # Logistic Regression
     2 # GridSearch CV params
     3 lr_grid = {}
     4 lr_grid["C"] =  [0.01, 0.1, 10, 100]
     5 lr_grid["fit_intercept"] =  [True, False]
     6 lr_grid["warm_start"] = [True,False]
     7 lr_grid["random_state"] = [1]
     8
     9 #Logistic Regression with GridSearch CV
    10 lr_clf_grid=Classification_Model(lr,X_train,X_test,y_train,y_test,lr_grid,"grid")
    11 result.append(lr_clf_grid)
```

```
Model paramters used are : GridSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 10, 100],
                         'fit_intercept': [True, False], 'random_state': [1],
                         'warm_start': [True, False]},
             return_train_score=True)
Accuracy of late delivery risk analyzer is: 56.971526700642585 %
Recall score of late delivery risk analyzer is: 57.82420326039066 %
Conf Matrix of late delivery risk analyzer is:
 [[ 4820 11487]
 [ 4048 15749]]
F1 score of late delivery risk analyzer is: 66.96999978738333 %
Classification Report for late delivery risk analyzer is:
              precision    recall  f1-score   support

           0       0.54      0.30      0.38     16307
           1       0.58      0.80      0.67     19797

    accuracy                           0.57     36104
   macro avg       0.56      0.55      0.53     36104
weighted avg       0.56      0.57      0.54     36104
```

Fig 19

Logistic Regression with RandomSearchCV optimizer

```
 1 # Random SearchCV params
 2 lr_random = {}
 3 lr_random["C"] = [0.01, 0.1, 10, 100]
 4 lr_random["fit_intercept"] =  [True, False]
 5 lr_random["warm_start"] = [True,False]
 6 lr_random["random_state"] = [1]
 7
 8 #Logistic Regression with RandomSearch CV
 9 lr_clf_random = Classification_Model(lr,X_train,X_test,y_train,y_test,lr_random,"random")
10 result.append(lr_clf_random)
```

```
Model paramters used are : RandomizedSearchCV(cv=5, estimator=LogisticRegression(), n_jobs=-1,
                   param_distributions={'C': [0.01, 0.1, 10, 100],
                                        'fit_intercept': [True, False],
                                        'random_state': [1],
                                        'warm_start': [True, False]},
                   random_state=1, return_train_score=True)
Accuracy of late delivery risk analyzer is: 56.971526700642585 %
Recall score of late delivery risk analyzer is: 57.82420326039066 %
Conf Matrix of late delivery risk analyzer is:
 [[ 4820 11487]
 [ 4048 15749]]
F1 score of late delivery risk analyzer is: 66.96999978738333 %
Classification Report for late delivery risk analyzer is:
              precision    recall  f1-score   support

           0       0.54      0.30      0.38     16307
           1       0.58      0.80      0.67     19797

    accuracy                           0.57     36104
   macro avg       0.56      0.55      0.53     36104
weighted avg       0.56      0.57      0.54     36104
```

Fig 20

Since the accuracy obtained was low, I performed feature scaling using standard scaler and min-max scaler.

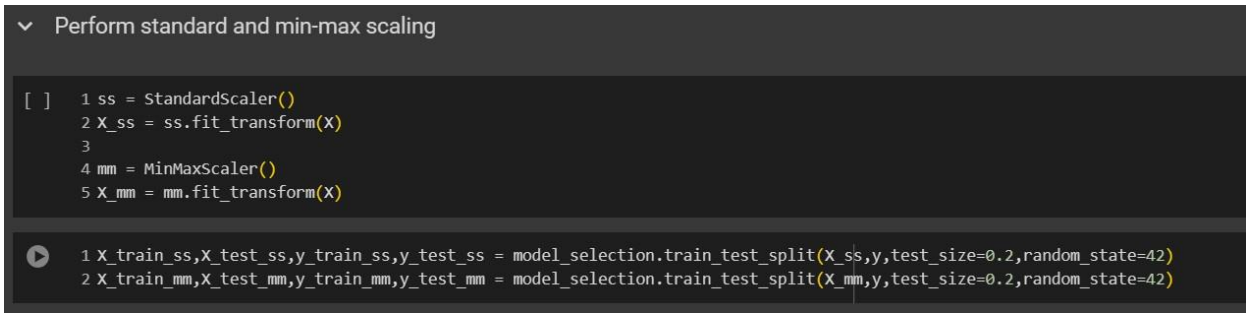

Fig 21

After the data has been scaled, I repeated the process on the scaled data to get the accuracy score and observed a significant improvement in the accuracy on the scaled dataset (Accuracy score ~ 75%)

APPLICATION OF GAUSSIAN NAÏVE BAYES CLASSIFIER:

Gaussian Naive Bayes is a machine learning classification technique based on a probabilistic approach that assumes each class follows a normal distribution. It assumes each parameter has an independent capacity of predicting the output variable

Gaussian Naïve Bayes is applied on the non-scaled data and it was observed that **70%** accuracy was achieved with non-scaled training data.

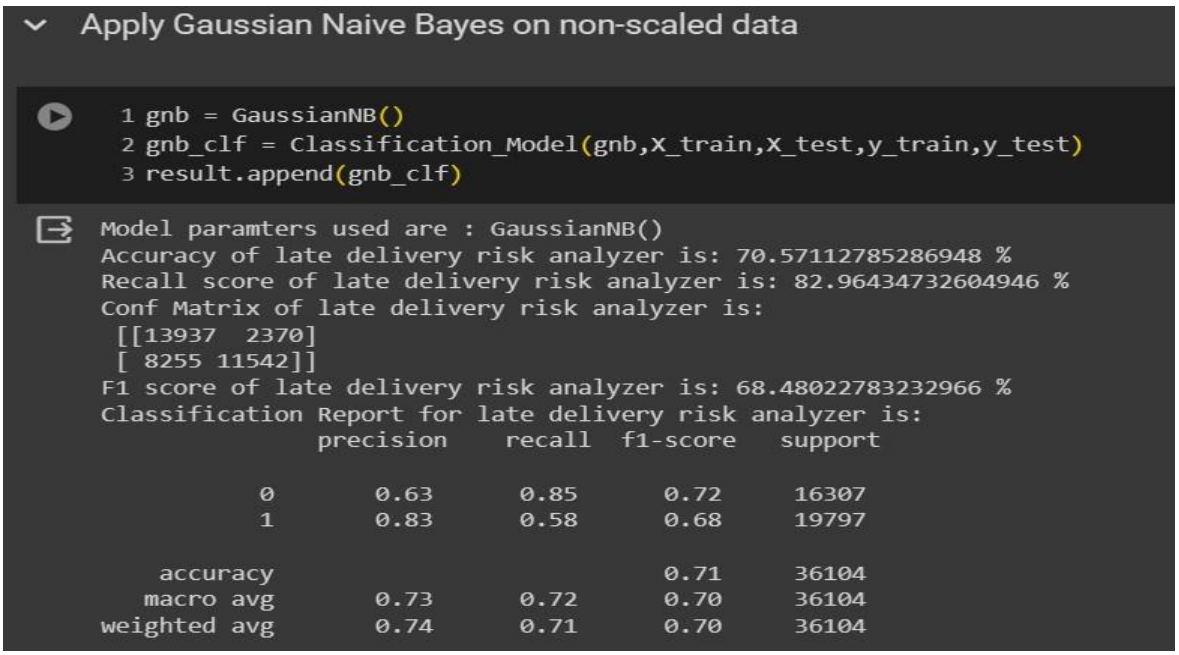

Fig 21

Gaussian Naïve Bayes is then applied on non-scaled data with GridSearchCV optimizer using which resulted in accuracy of 72%, a 2% improvement score. After getting the results Gaussian Naïve Bayes is then applied on the standard scaled data and on the min-max scaled data. I then carried out K-Fold Cross-Validation on the model and it was discovered that the model generated the same results, hence we can conclude that the model is not showing overfitting behavior.

APPLICATION OF SUPPORT VECTOR CLASSIFIER:

I attempted at training the model on Support Vector classifier and achieve an accuracy score of 77% with standard scaled dataset when the train size was reduced to 40%. But when executing the same mode after performing hyper-parameter tuning,the model it taking long to execute and accuracy improvement is not significant. So, SVC is not a suitable model for this problem as it has very high tradeoff associated with model performance.

APPLICATION OF DECISION TREE CLASSIFIER:

Decision Tree Classifier was applied to the non-scaled data and without any hyper-parameter tuning. I achieve a significant increase in the accuracy using Decision Tree Classifier (Accuracy 93%). This improvement in the score triggered us to checkfor overfitting problem which is very common in tree based models.

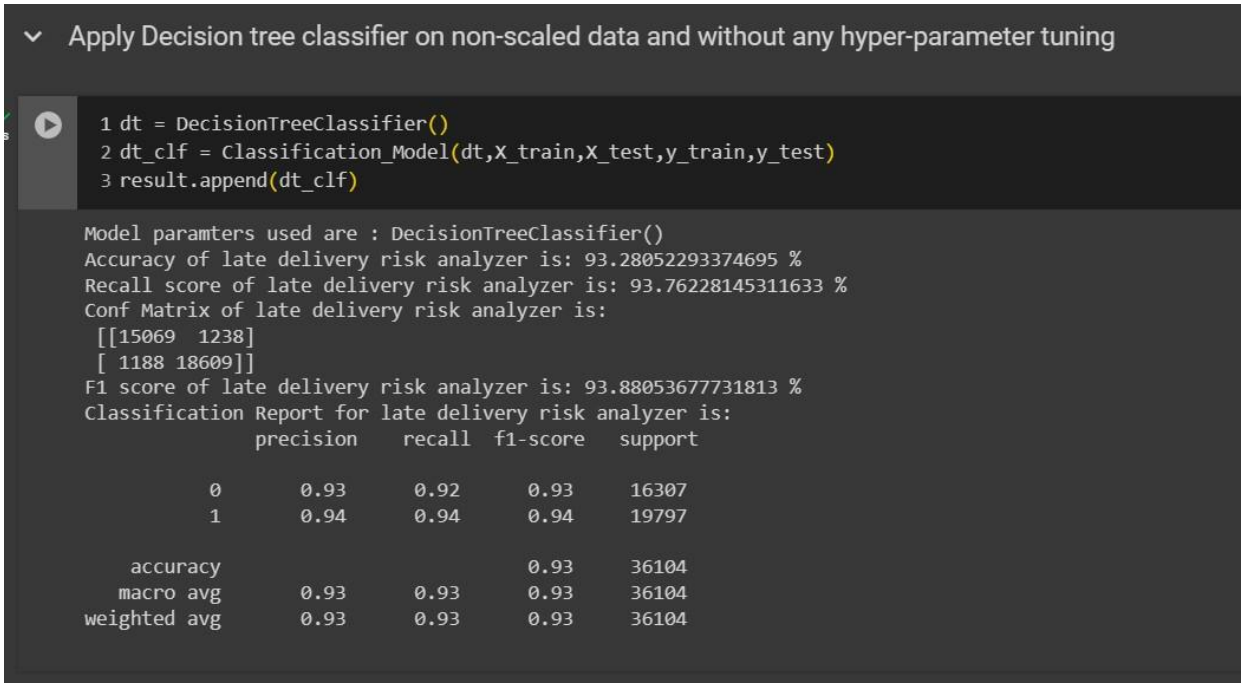

Fig 22

I then proceeded to check for overfitting, and I observed that there is some overfitting done in the training set, I then proceeded to apply GridSearchCV to eliminate possible overfitting of the data. I then discovered that after the application

of GridSearchCV (cv=5) the accuracy score didn't change much, thus I concludedthat the Decision tree classifier model wasn't overfitting.

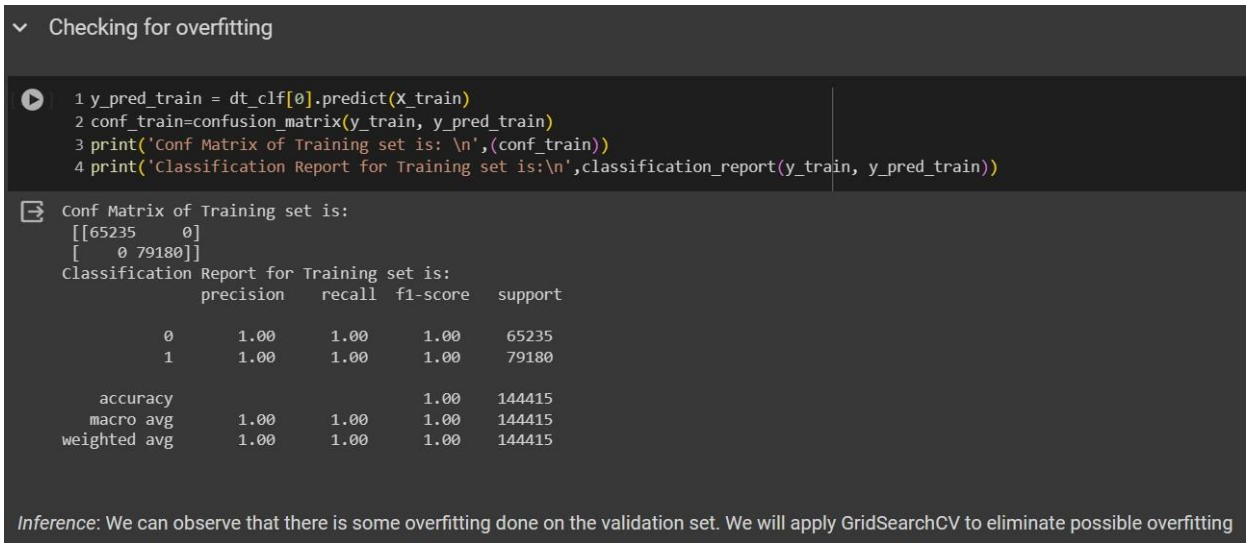

Fig 23

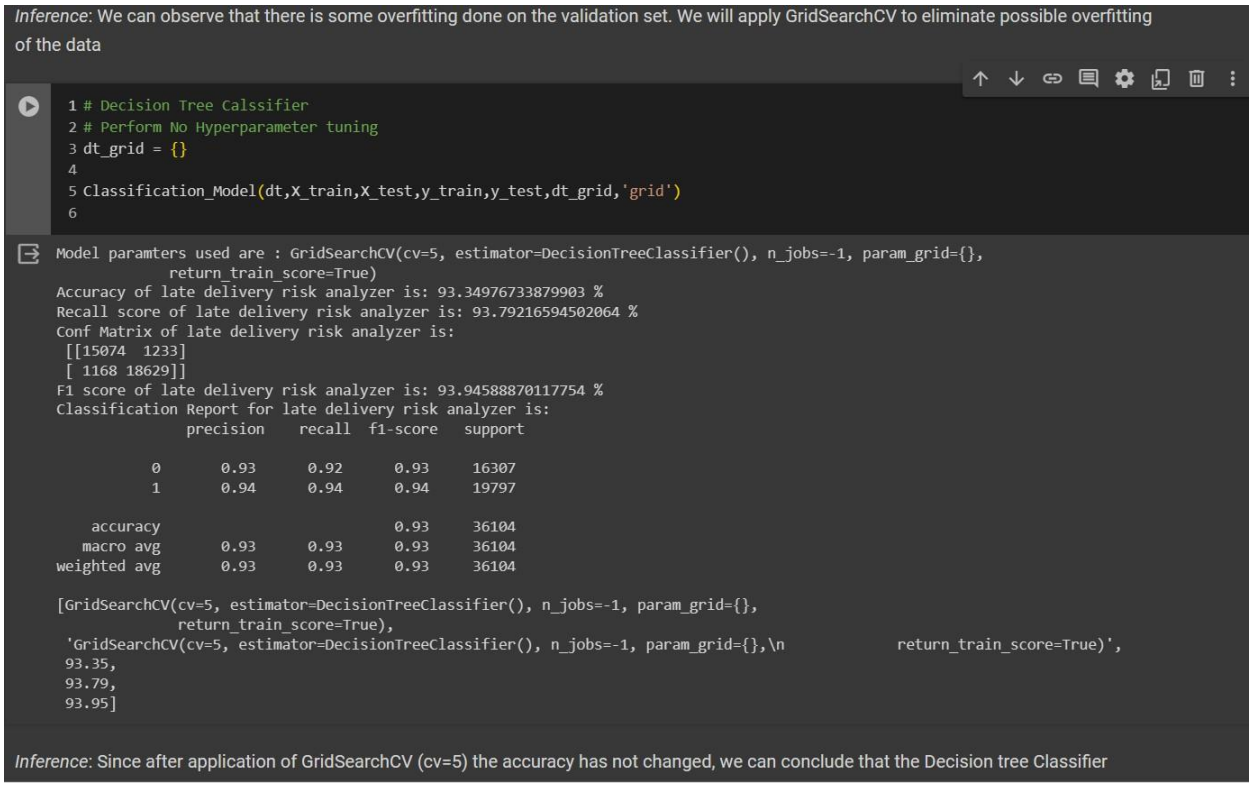

Fig 24

As a next step I tried fine tuning the model with hyper-parameter tuning in Decision Tree Classifier, I observed that the accuracy didn't improve compared to the standard decision tree classifier.

Following feature importance result was also obtained from the analysis done to identify important features for decision tree classifier:

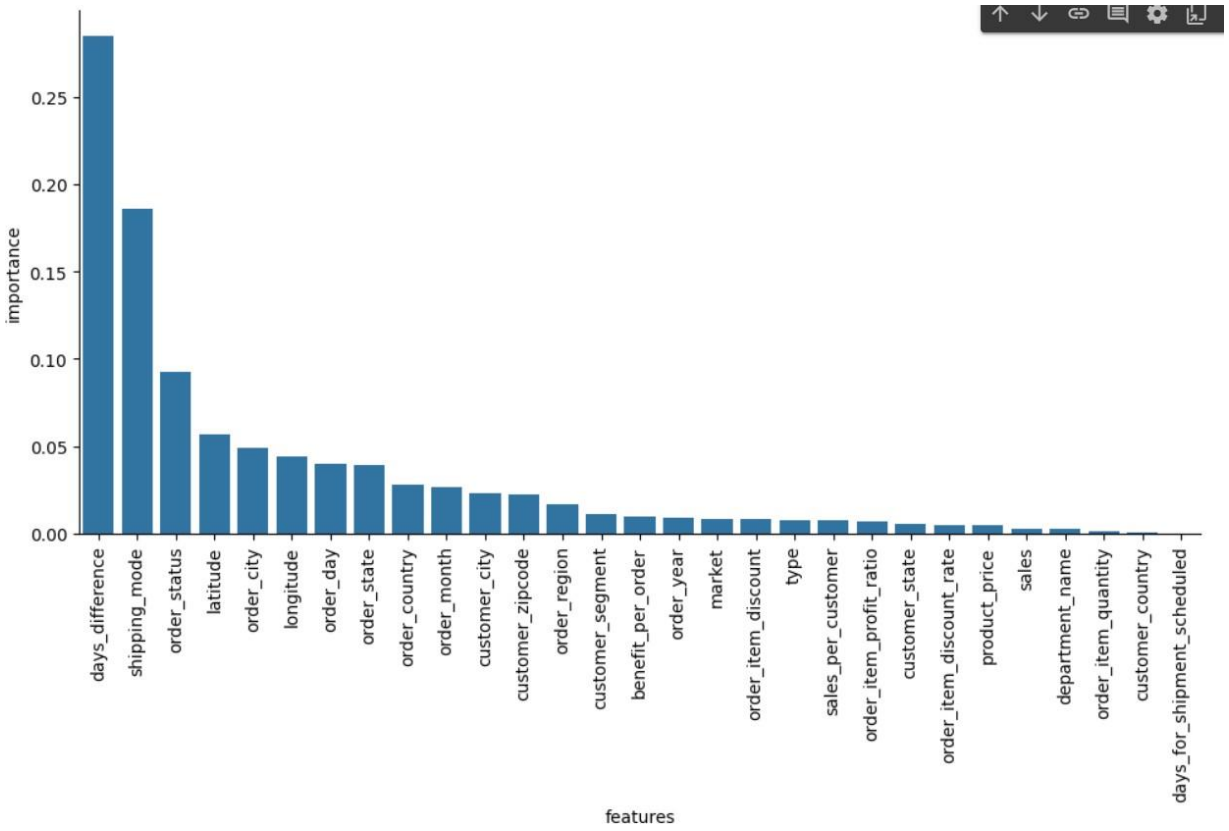

Fig 25

It was discovered that days_difference is the most important and shipping_mode is the second most important feature for decision tree classifier.

RESULTS:

After the creation and the training of each model, the results of each classification model were stored in a list which was transformed into a dataframe for better representation and easy understanding.

	Model	Model_name	Accuracy	Recall	F1_Score
1	LogisticRegression()	LogisticRegression()	55.12	55.57	68.88
2	GridSearchCV(cv=5, estimator=LogisticRegressio...	GridSearchCV(cv=5, estimator=LogisticRegressio...	56.97	57.82	66.97
3	RandomizedSearchCV(cv=5, estimator=LogisticReg...	RandomizedSearchCV(cv=5, estimator=LogisticReg...	56.97	57.82	66.97
4	LogisticRegression()	LogisticRegression()	74.03	76.04	76.45
5	GridSearchCV(cv=5, estimator=LogisticRegressio...	GridSearchCV(cv=5, estimator=LogisticRegressio...	74.03	76.03	76.45
6	RandomizedSearchCV(cv=5, estimator=LogisticReg...	RandomizedSearchCV(cv=5, estimator=LogisticReg...	74.03	76.03	76.45
7	LogisticRegression()	LogisticRegression()	74.03	76.03	76.45
8	GridSearchCV(cv=5, estimator=LogisticRegressio...	GridSearchCV(cv=5, estimator=LogisticRegressio...	74.03	76.03	76.44
9	RandomizedSearchCV(cv=5, estimator=LogisticReg...	RandomizedSearchCV(cv=5, estimator=LogisticReg...	74.03	76.03	76.45
10	GaussianNB()	GaussianNB()	70.57	82.96	68.48
11	GridSearchCV(cv=5, estimator=GaussianNB(), n_j...	GridSearchCV(cv=5, estimator=GaussianNB(), n_j...	72.67	87.86	70.02
12	GaussianNB()	GaussianNB()	72.67	87.84	70.02
13	GridSearchCV(cv=5, estimator=GaussianNB(), n_j...	GridSearchCV(cv=5, estimator=GaussianNB(), n_j...	72.67	87.84	70.02
14	GaussianNB()	GaussianNB()	72.67	87.84	70.02
15	GridSearchCV(cv=5, estimator=GaussianNB(), n_j...	GridSearchCV(cv=5, estimator=GaussianNB(), n_j...	72.74	88.07	70.05
16	SVC()	SVC()	77.31	83.89	77.86
17	SVC()	SVC()	77.82	82.73	78.88
18	DecisionTreeClassifier()	DecisionTreeClassifier()	93.24	93.76	93.84
19	GridSearchCV(cv=5, estimator=DecisionTreeClass...	GridSearchCV(cv=5, estimator=DecisionTreeClass...	82.11	87.45	82.82

Fig 26

Models with their Accuracies, Recall and F1_scores

Result for Best model selection:

From the above matrix, we can see that Decision Tree Classifier model is resulting in the best accuracy for predicting the late delivery risk for our problem. Since we have confirmed that there is no overfitting shown by this model, we can recommend this model usage for predicting late delivery risk.

CONCLUSION:

This project proposed three Machine Learning Classification Technique to predict and help decrease late delivery risk at the advent and with critical analysis, training and development of the models we concluded that in light of the three classification models created which were: Logistic Regression model, Gaussian Naïve Bayes Classifier Model and Decision Tree Classifier Model, the model well suited to aid the main concept of this research was Decision Tree Classifier model. The model demonstrated the highest accuracy rate in predicting late delivery risk and the findings can help reduce the risk of late delivery and improve supply chain efficiency.

REFERENCE:

Gudum, C. K., 2002. On the distribution of lead time delays in supply chains. Working paper, København, 38p.

Pedregosa *et al.*,2011. Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830. Available at: https://scikit-learn.org/stable/about.html#citing-scikit-learn.

Constante, Fabian; Silva, Fernando; Pereira, António (2019), "DataCo SMART SUPPLY CHAIN FOR BIG DATA ANALYSIS", Mendeley Data, V5, doi: 10.17632/8gx2fvg2k6.5. Available at: https://data.mendeley.com/datasets/8gx2fvg2k6/5