CO-586 Data Management and Analytics in Industry 4.0

Prof. Dr. -Ing. Hendro Wicaksono

Exploratory Data Analysis

Handed in by:

Date: 29/05/2023

Nischal Chaulagain

Table of Content

List of Figures

List of Tables

1 Dataset 1: Daily Demand Forecasting Orders Dataset

1.1 Data description

The data set we have is the Daily Demand Forecasting orders data set of a company. Looking at the data set, it seems that the company is a producer of different types of products and the data set given is the record of orders received within a month. The data contains multiple columns which include the week of the months which can range from 1 to 5, day of the week (from Monday to Friday) as where Monday refers to day 1, Tuesday refers to day 2 and so on, then we have the columns for the type of order based on the urgency as urgent order and non-urgent order. The order has also been categorized into three other columns based on which type (A, B, and C) the order falls into from order type A to C. Then we have the columns for fiscal sector orders, orders from the traffic controller sector, banking order (1,2,3) and Target as total orders.

Table 1 Data dictionary of order dataset

Column name	**Definition**	**Data type**	**Possible values (only for nominal and ordinal data types)**	**Required?**
Week of the month	number of weeks in a month on which the order was placed	Nominal	1-5	Yes
Day of the week	Number of days of the week on which order was placed	Nominal	1-5	Yes
Non-urgent order and urgent order	Number of orders according to due date	Discrete		Yes
Order Type (A, B, C)	Types of order based on which categories they fall into	Discrete		Yes
Fiscal and Traffic controller sector orders	Count of orders from fiscal sectors and orders from traffic controller sector	Discrete		Yes
Banking Order (1,2,3)	Count of banking orders	Discrete		Yes
Target (Total Orders)	Total orders to be processed or total orders received	Discrete		Yes

1.2 Exploratory data analysis

1.2.1 Data visualization and exploration

To visualize our dataset set, we first loaded the csv file into python using the code as shown below:

```
# Loading CSV file
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00409/Daily_Demand_Forecasting_Orders.csv"
df = pd.read_csv(url, sep=';')
```

Figure 1. Loading Data

Then, we checked if the file had loaded properly, and the file is the correct file using the python code print(df.head()) which gave us the first five rows of the dataset. Now we were able to make sure that python was reading the dataset correctly and we were good to move on.

Then, we checked if the file had loaded properly, and the file is the correct file using the python code print(df.head()) which gave us the first five rows of the dataset. Now we were able to make sure that python was reading the dataset correctly and we were good to move on.

After that, we used python code print(df.info()) to check the size of our dataset and the datatypes of the dataset. The result we got is as depicted by the picture below. Now our dataset is ready for preprocessing.

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 13 columns):
 #   Column                                                        Non-Null Count  Dtype
---  ------                                                        --------------  -----
 0   Week of the month (first week, second, third, fourth or fifth week  60 non-null     int64
 1   Day of the week (Monday to Friday)                            60 non-null     int64
 2   Non-urgent order                                              60 non-null     float64
 3   Urgent order                                                  60 non-null     float64
 4   Order type A                                                  60 non-null     float64
 5   Order type B                                                  60 non-null     float64
 6   Order type C                                                  60 non-null     float64
 7   Fiscal sector orders                                          60 non-null     float64
 8   Orders from the traffic controller sector                     60 non-null     int64
 9   Banking orders (1)                                            60 non-null     int64
 10  Banking orders (2)                                            60 non-null     int64
 11  Banking orders (3)                                            60 non-null     int64
 12  Target (Total orders)                                         60 non-null     float64
dtypes: float64(7), int64(6)
```

Figure 2 Data Info

```
   Week_of_month  Day_of_week  Non_urgent_order  Urgent_order  Order_type_A  \
0              1            4           316.307       223.270        61.543
1              1            5           128.633        96.042        38.058
2              1            6            43.651        84.375        21.826
3              2            2           171.297       127.667        41.542
4              2            3            90.532       113.526        37.679

   Order_type_B  Order_type_C  Fiscal_sector_orders  \
0       175.586       302.448                 0.000
1        56.037       130.580                 0.000
2        25.125        82.461                 1.386
3       113.294       162.284                18.156
4        56.618       116.220                 6.459

   Traffic_controller_orders  Banking_orders_1  Banking_orders_2  \
0                      65556             44914            188411
1                      40419             21399             89461
2                      11992              3452             21305
3                      49971             33703             69054
4                      48534             19646             16411

   Banking_orders_3  Target_orders
0             14793        539.577
1              7679        224.675
2             14947        129.412
3             18423        317.120
4             20257        210.517
```

Figure 3 Data Heads

```
       Week_of_month  Day_of_week  Non_urgent_order  Urgent_order  \
count      60.000000    60.000000         60.000000     60.000000
mean        3.016667     4.033333        172.554933    118.920850
std         1.282102     1.401775         69.505788     27.170929
min         1.000000     2.000000         43.651000     77.371000
25%         2.000000     3.000000        125.348000    100.888000
50%         3.000000     4.000000        151.062500    113.114500
75%         4.000000     5.000000        194.606500    132.108250
max         5.000000     6.000000        435.304000    223.270000

       Order_type_A  Order_type_B  Order_type_C  Fiscal_sector_orders  \
count     60.000000     60.000000     60.000000             60.000000
mean      52.112217    109.229850    139.531250             77.396133
std       18.829911     50.741388     41.442932            186.502470
min       21.826000     25.125000     74.372000              0.000000
25%       39.456250     74.916250    113.632250              1.243250
50%       47.166500     99.482000    127.990000              7.831500
75%       58.463750    132.171000    160.107500             20.360750
max      118.178000    267.342000    302.448000            865.000000

       Traffic_controller_orders  Banking_orders_1  Banking_orders_2  \
count                  60.000000         60.000000         60.000000
mean                44504.350000      46640.833333      79401.483333
std                 12197.905134      45220.736293      40504.420041
min                 11992.000000       3452.000000      16411.000000
25%                 34994.250000      20130.000000      50680.500000
50%                 44312.000000      32527.500000      67181.000000
75%                 52111.750000      45118.750000      94787.750000
max                 71772.000000     210508.000000     188411.000000

       Banking_orders_3  Target_orders
count         60.000000      60.000000
mean       23114.633333     300.873317
std        13148.039829      89.602041
min         7679.000000     129.412000
25%        12609.750000     238.195500
50%        18011.500000     288.034500
75%        31047.750000     334.237250
max        73839.000000     616.453000
```

Figure 4 Data Statistics

1.2.2 Brainstorming and discussions

i. Looking at the data statistics, we can see that the mean of the non-urgent orders is higher than urgent orders, which also signifies that the company has higher numbers of non-urgent as compared to the urgent orders.
ii. The average number of orders placed each day is around 300,000.
iii. Among the orders placed, most of the orders in the 60-day period are order type C.

1.3 Data preprocessing

As we investigated our dataset, we found that the columns had been named with exceptionally long information which made it hard to read and do the analysis. So, we renamed each column such that the columns are easier to access and better readable. It was done using the python code shown below:

df.rename(column={“column_name”: ”new_column_name”}, inplace=True).

```python
# Renameing columns

df.rename(columns={'Week of the month (first week, second, third, fourth or fifth week': 'Week_of_month',

                    'Day of the week (Monday to Friday)': 'Day_of_week',

                   'Non-urgent order': 'Non_urgent_order',

                  'Urgent order': 'Urgent_order',

                  'Order type A': 'Order_type_A',

                  'Order type B': 'Order_type_B',
                    'Order type C' : 'Order_type_C',

             'Fiscal sector orders': 'Fiscal_sector_orders',

                   'Orders from the traffic controller sector': 'Traffic_controller_orders',

                  'Banking orders (1)': 'Banking_orders_1',

                 'Banking orders (2)': 'Banking_orders_2',

                'Banking orders (3)': 'Banking_orders_3',
                   'Target (Total orders)': 'Target_orders'}, inplace=True)
```

Figure 5 Renaming Columns

```
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Week_of_month              60 non-null     int64
 1   Day_of_week                60 non-null     int64
 2   Non_urgent_order           60 non-null     float64
 3   Urgent_order               60 non-null     float64
 4   Order_type_A               60 non-null     float64
 5   Order_type_B               60 non-null     float64
 6   Order_type_C               60 non-null     float64
 7   Fiscal_sector_orders       60 non-null     float64
 8   Traffic_controller_orders  60 non-null     int64
 9   Banking_orders_1           60 non-null     int64
 10  Banking_orders_2           60 non-null     int64
 11  Banking_orders_3           60 non-null     int64
 12  Target_orders              60 non-null     float64
dtypes: float64(7), int64(6)
```

Figure 6 Renamed Columns

1.3.1 Handling missing values

Checking if missing values were done using two steps. At first, we checked if we had any NULL values in our dataset which we found out we did not and later we replaced all the zeros with NULL and found that we had 13 missing values as shown in MSNO matrix below:

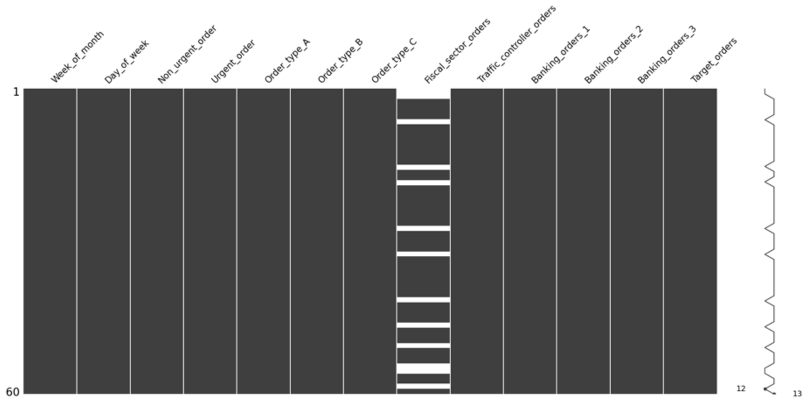


Figure 7 MSNO Matrix

```
Missing values:
Week_of_month                0
Day_of_week                  0
Non_urgent_order             0
Urgent_order                 0
Order_type_A                 0
Order_type_B                 0
Order_type_C                 0
Fiscal_sector_orders        13
Traffic_controller_orders    0
Banking_orders_1             0
Banking_orders_2             0
Banking_orders_3             0
Target_orders                0
dtype: int64
```

Figure 8 Missing Values Count

To handle missing values now we need to find if our data are Missing at Random or Missing Completely at Random or Missing Not at Random. To recognize these, we created a correlation matrix with the missing values. We found that only 3 columns namely Urgent_Order, Order_type_C and Banking_orders_2 showed positive correlation with lower than 0.3 points. By this statistic we could conclude that out data was MAR (Missing at Random)

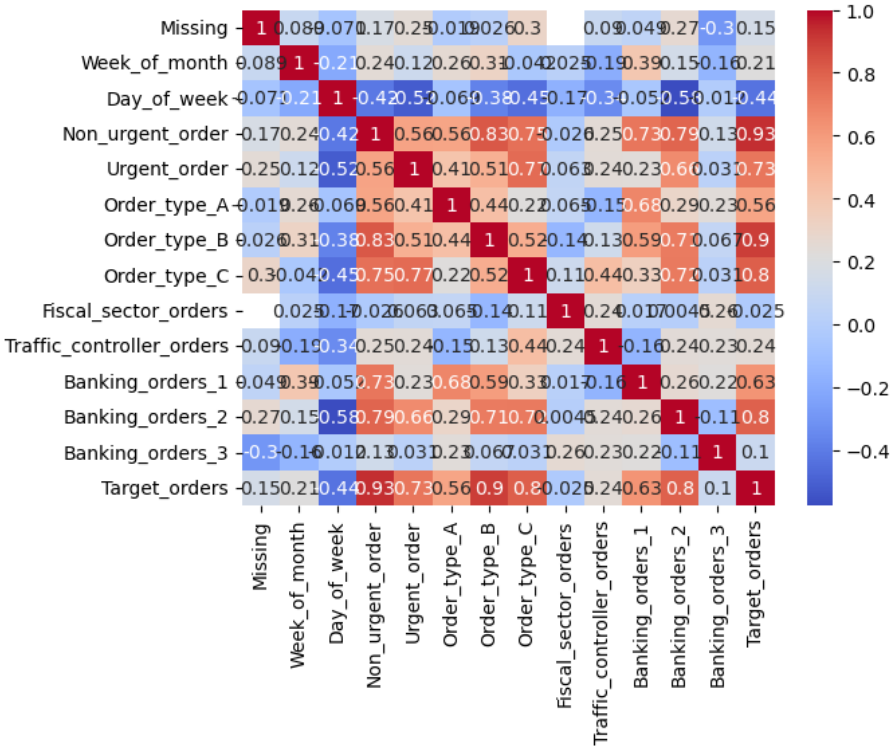


Figure 9 Correlation Matrix Heat Map

Now we could successfully apply Simple Mean Imputer to handle the MAR values. We performed the imputation and transformed the data as shown below. This is how we successfully handled the missing values.

```
# Create an instance of SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# Fit and transform the data
df_imputed = imputer.fit_transform(df)

df= pd.DataFrame(df_imputed, columns=df.columns)
```

```
missing_values = df.isnull().sum()
print("Missing values:")
print(missing_values)
```

```
Missing values:
Week_of_month                0
Day_of_week                  0
Non_urgent_order             0
Urgent_order                 0
Order_type_A                 0
Order_type_B                 0
Order_type_C                 0
Fiscal_sector_orders         0
Traffic_controller_orders    0
Banking_orders_1             0
Banking_orders_2             0
Banking_orders_3             0
Target_orders                0
dtype: int64
```

Figure 10 Mean Imputer Method

1.3.2 Encoding categorical data

We had two categorical columns in our dataset. We decided to use one hot coding for weeks of month and days of the week which are our categorical columns because it allows capturing interactions or nonlinear relationships between categories. It also ensures that encoded features are comparable in scale and avoids giving numerical advantages to any class.

```
# selecting categorical columns
cat_cols = ['Week_of_month', 'Day_of_week']

# creation of one-hot encoding
one_hot = pd.get_dummies(df[cat_cols],)

# merging one-hot encoding with original data
df_encoded = pd.concat([one_hot, df], axis=1)

df = df_encoded.iloc[:, 2:].copy()
```

Figure 11 One Hot Encoding

	Week_of_month	Day_of_week
0	1.0	4.0
1	1.0	5.0
2	1.0	6.0
3	2.0	2.0
4	2.0	3.0

Figure 12 Encoded Data

1.3.3 Feature scaling

Our large data may dominate our final finding and small data may lose its importance when machine learning tries to analyze our data. To make sure that this does not happen we have to scale our data all to a fixed scale without losing its distribution. To do that we used a Standard Scaling Method.

We used standard scaling because as our data forms a normal distribution, it can help us to scale our data to have a mean of 0 and standard deviation of 1, which helps us to reduce the impact of outliers and make our data more interpretable as we now have a scaled data distribution centered to around 0.

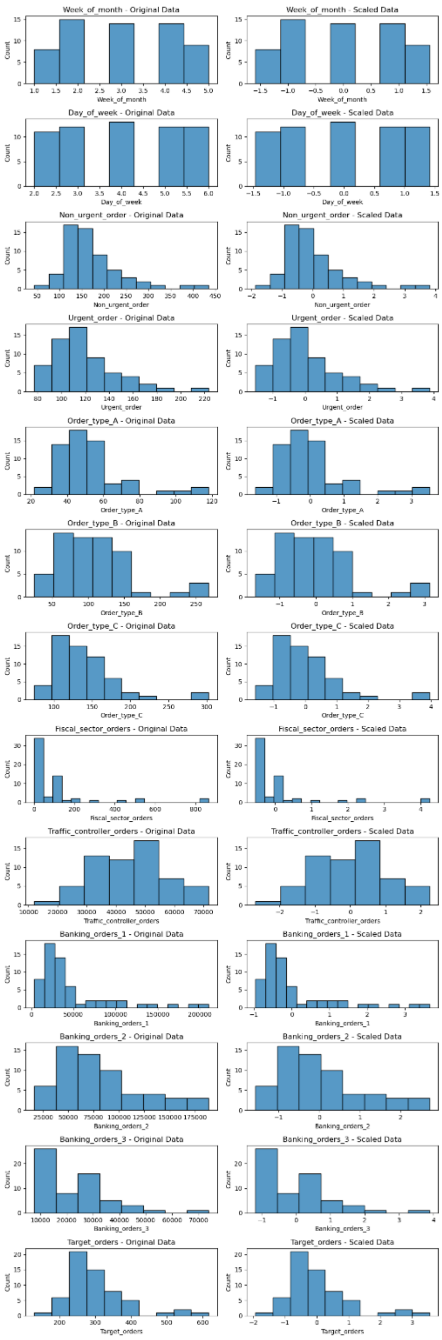


Figure 13 Standard Scaled Bar Chart

1.3.4 Other preprocessing

After finishing our scaling, we checked to see if we needed to perform other preprocessing methods. Based on the results of EDA our data does not show any significant class imbalance in the dataset, Hence, oversampling or under sampling is not necessary.

However, we can use dimension-reduction techniques for our data set. Columns such as order types and banking orders can be reduced using the PCA method of reduction. PCA is a useful technique when we must reduce our data without losing information of the original dataset and reduce the dimension and the correlation in between PC1 and PC2.

```
df.head()
```

	Week_of_month	Day_of_week	Non_urgent_order	Urgent_order	Fiscal_sector_orders	Traffic_controller_orders	Target_orders	PC1	PC2
0	-1.586212	-0.023980	2.085656	3.872880	0.000000	1.740406	2.686525	3.713013	-2.842836
1	-1.586212	0.695422	-0.637250	-0.849140	0.000000	-0.337749	-0.857585	-1.168587	-1.176522
2	-1.586212	1.414823	-1.870229	-1.282156	-0.539988	-2.687898	-1.929736	-3.215757	-0.172652
3	-0.799661	-1.462783	-0.018251	0.324610	-0.447032	0.451945	0.182851	-0.224301	-0.653677
4	-0.799661	-0.743382	-1.190047	-0.200228	-0.511869	0.333144	-1.016928	-2.074444	0.281667

Figure 14 PCA Reduction Technique

1.4 Development of predictive models

Now we classified our Target variable as Target Orders as our final analysis is for prediction of targets and we classified the rest of columns as predictive variables or models.

```
# Defining the predicting variables X and response Y

X = df[['Week_of_month', 'Day_of_week', 'Non_urgent_order', 'Urgent_order',
        'Fiscal_sector_orders', 'Traffic_controller_orders', 'PC1','PC2']]
y = df['Target_orders']
```

Figure 15 Defining Predicting Variable and Response

1.4.1 Dataset splitting

After successful preprocessing methods, we can now begin our analysis of regression models starting with splitting our dataset. We are using 80/20 data split on training and test dataset respectively because the larger training portion provides sufficient data to identify relationships, preprocess variables, and create new features. Therefore, we want to use 80 percent of our data to train the model and 20 percent to test the performance. Furthermore, this split allows us to separate enough data to train our model while we still will have enough data to test performance.

1.4.2 Model selection and parameter settings

To choose the best parameter we used GridSearchCV method which is an algorithm that returns us the best parameter for our model with the help of negative mean squared error and after that we performed multiple regression analysis models to our dataset.

We used the following models for our regression analysis.

I. Linear regression model: It is a widely used regression method which makes linear relationship between the variables. We chose linear because it is quite easy to interpret and provides baseline comparison with complex models.

Table 2 Parameter setting of Linear regression.

Parameter	**Value**
Copy_x	True
Fit_intercept	False

II. Random Forest Regression: We used Random Forest Regression because it combines multiple decision trees to improve our accuracy and reduce overfitting. Furthermore, it has high predictive accuracy.

Table 3 Parameter setting of random forest regression.

Parameter	**Value**
max_depth	5
min_samples_leaf	1
min_samples_split	2
n_estimators	150

Decision Tree Regression: Decision Tree uses an intuitive method which can handle both continuous and categorical data. It is easy to interpret and can provide insights into important features.

Table 4 Parameter setting of decision tree regression.

Parameter	**Value**
criterion	squared_error
max_depth	8
min_samples_leaf	1
min_samples_split	5
splitter	random

III. SVM Regression: SVM is a powerful method which is useful when data is complex and there is a non-linear relationship between target variable and features. SVM can handle high dimensional data and has a good balance between Bias and Variance.

Table 5 Parameter setting of SVM regression.

Parameter	**Value**
C	0.1
gamma	scale
kernel	linear

1.5 Models' evaluation

For each model, we plotted the regression line to check the relationship between the predicted vs actual values and the residual vs actual values. We used both testing dataset and test dataset to check the relation. The results of each model analysis are shown and discussed below.

I. **Linear Regression Analysis**: Below, we can see the linear regression plot of the Predicted value with Actual Values and Residual values with Actual Values. We can clearly see that there is a strong linear relationship between the predicted values and the actual value while the linear relationship between the residual and actual values seems to be weak.

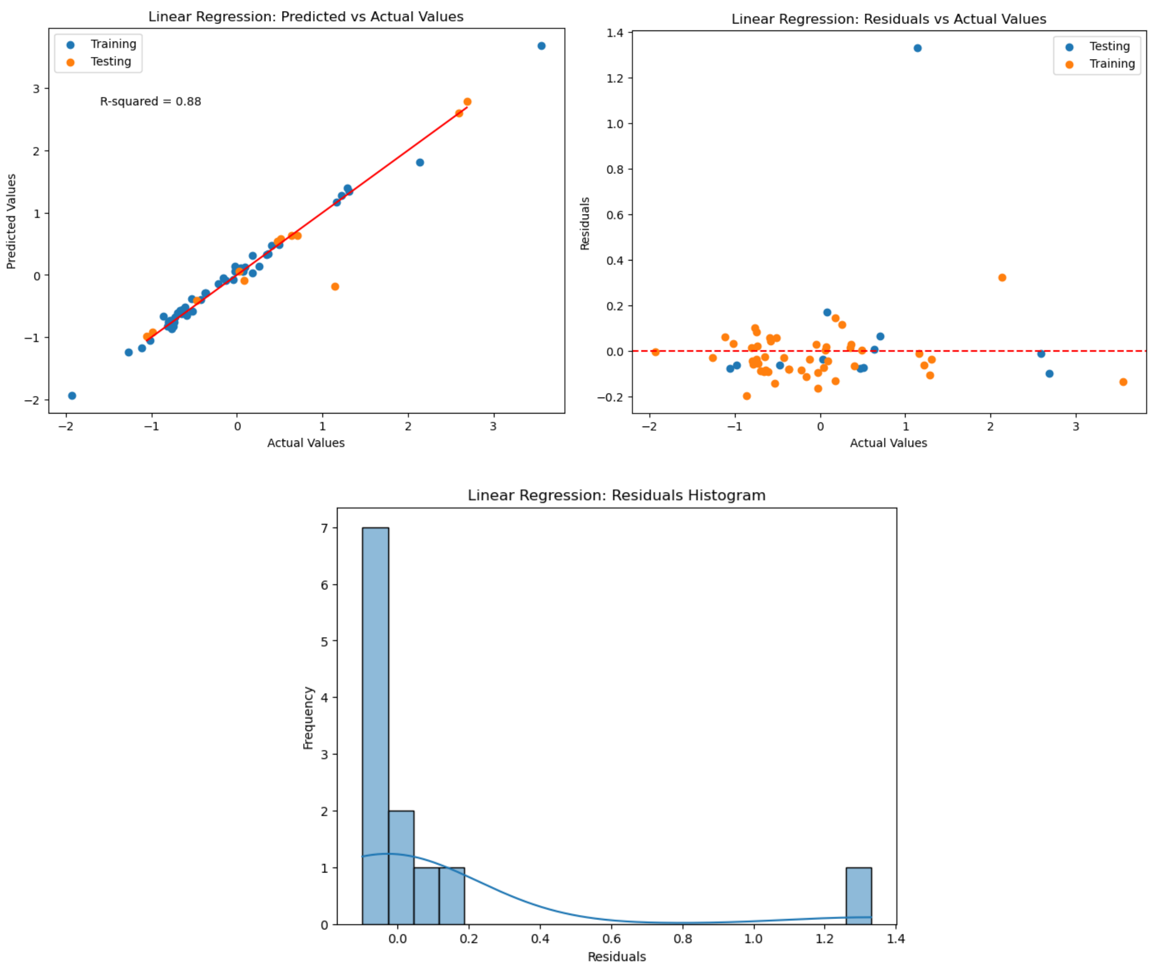


Figure 16 Linear Regression

Linear Regression:	Linear Regression Metrics (k=10):
MAE: 0.17	MAE: 0.14 ± 0.06
MSE: 0.15	MSE: 0.06 ± 0.09
RMSE: 0.39	RMSE: 0.20 ± 0.13
MAPE: 0.44	MAPE: 0.82% ± 0.99%

Figure 17 Errors of Linear Regression

II. **Random Forest Regression**: This model shows the relation between the Predicted vs Actual Values is weak linear, and the relation between the residual and actual values is also weak linear.

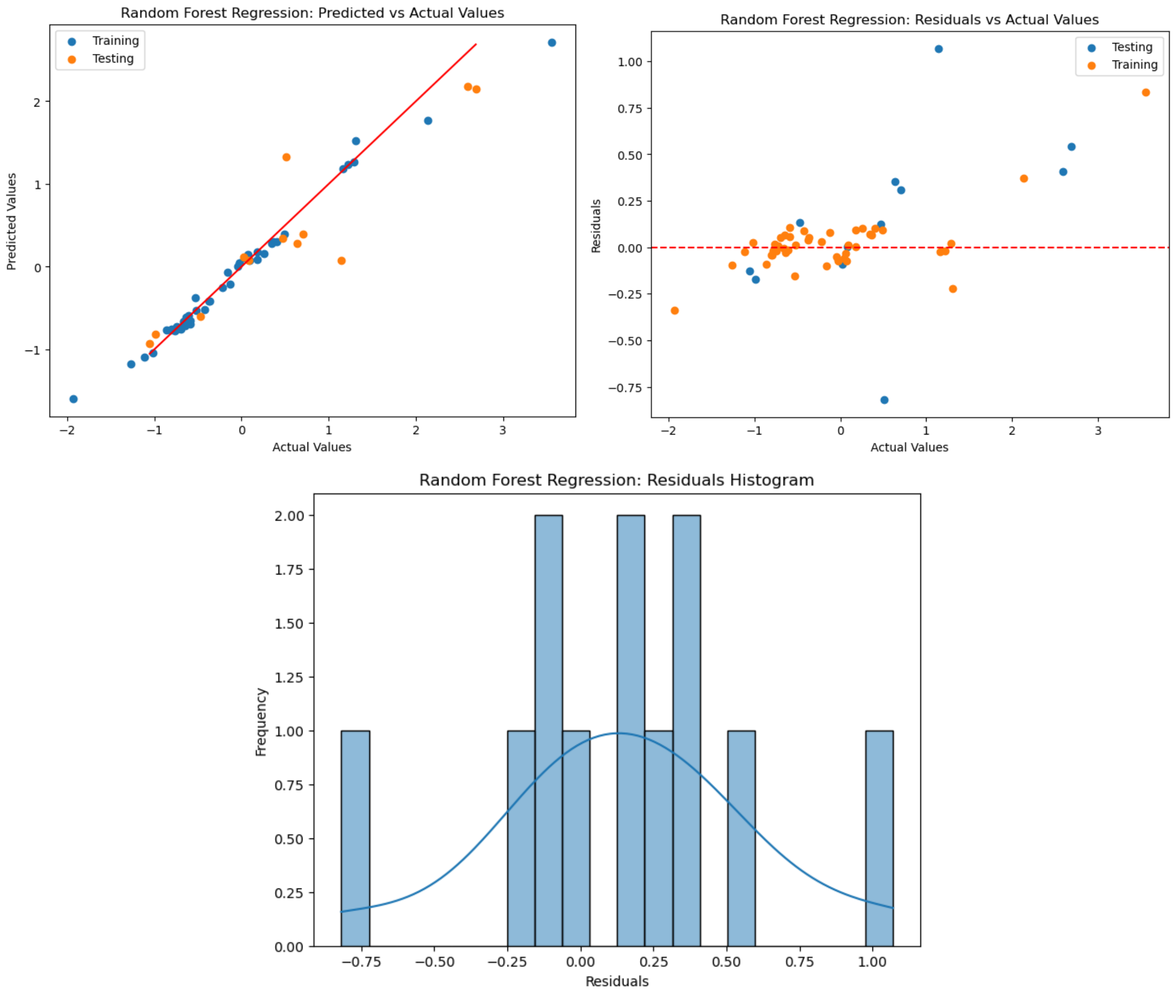


Figure 18 Random Forest Regression

```
Random Forest Tree Regression:
MAE: 0.19
MSE: 0.15
RMSE: 0.39
MAPE: 0.66
```

```
Random Forest Metrics (k=10):
MAE: 0.21 ± 0.08
MSE: 0.10 ± 0.09
RMSE: 0.29 ± 0.14
MAPE: 1.12% ± 1.60%
```

Figure 19 Errors on Random Forest Regression

III. **Decision Tree Regression**: With this model, the relation between the predicted vs actual and the relation between the residual vs actual values both seem to be weak linear.

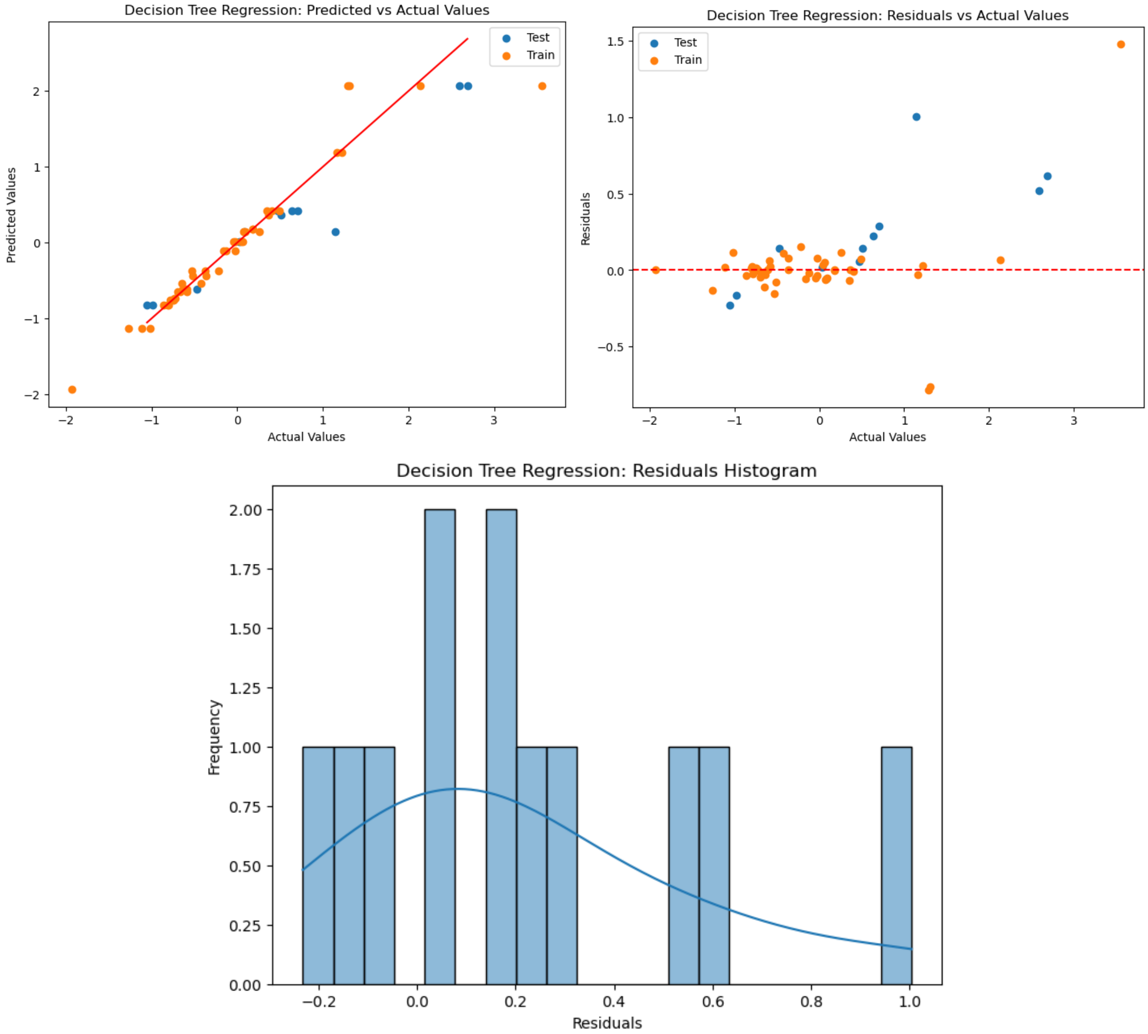


Figure 20 Decision Tree Regression

```
Decision Tree Regression:
MAE: 0.29
MSE: 0.16
RMSE: 0.40
MAPE: 0.38
```

```
Decision Tree Metrics (k=10):
MAE: 0.27 ± 0.08
MSE: 0.15 ± 0.12
RMSE: 0.36 ± 0.16
MAPE: 1.57% ± 2.39%
```

Figure 21 Errors on Decision Tree Regression

IV. **SVM Regression**: With the SVM regression the relation between the predicted vs actual values seems to be strong linear and the relation between the residual vs actual values seem to be weak linear.

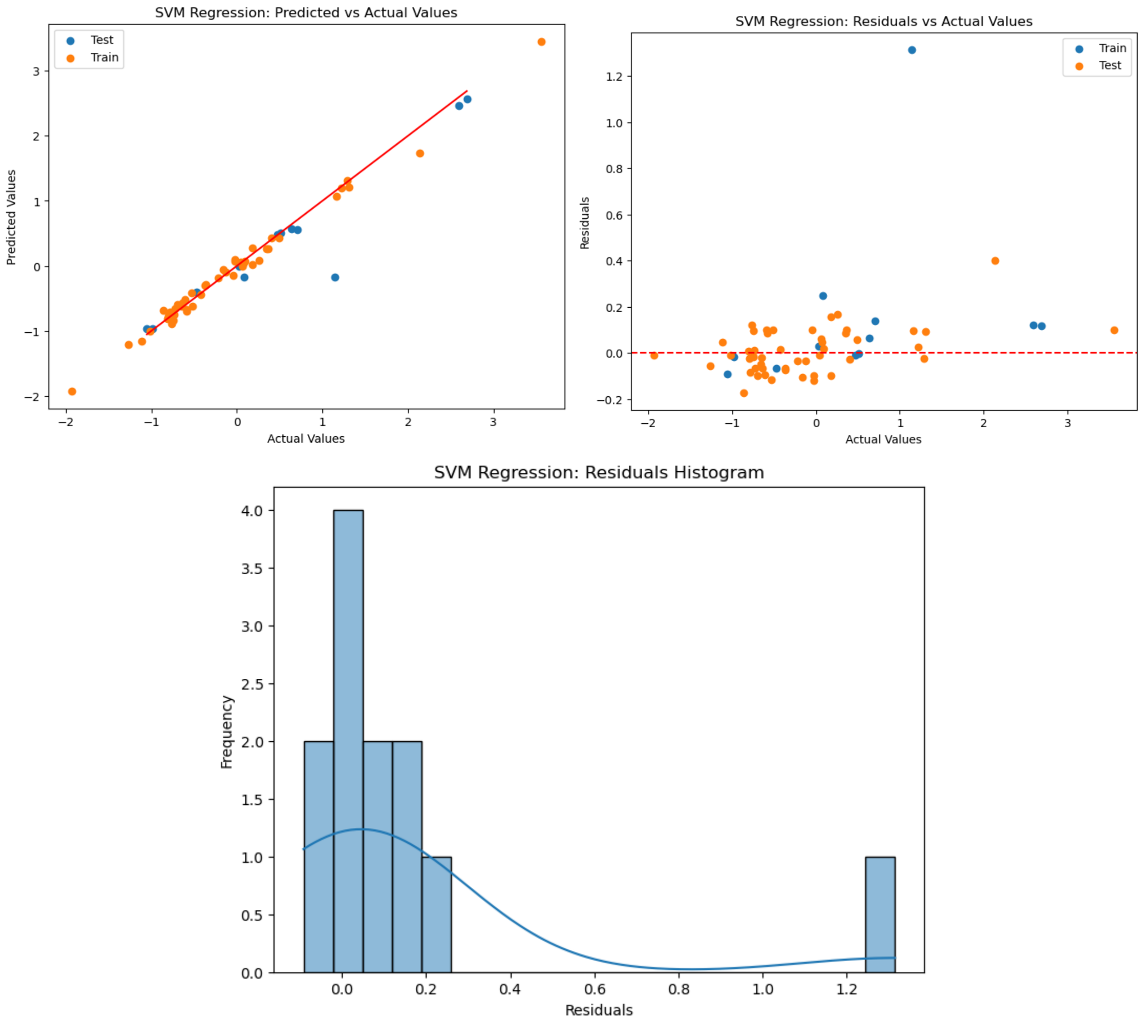


Figure 22 SVM Regression

SVM Regression:
MAE: 0.19
MSE: 0.15
RMSE: 0.39
MAPE: 0.49

SVM Metrics (k=10):
MAE: 0.32 ± 0.23
MSE: 0.44 ± 0.57
RMSE: 0.54 ± 0.40
MAPE: 0.84% ± 1.02%

Figure 23 Errors on SVM Regression

Based on our K-fold cross validation, we see that Linear regression has lowest MAE, MSE, RMSE, and MAPE values compared to other models. This predicts that linear regression model performs best in predicting our target variable which is target orders. Random Forest model is second best performer, but it has higher variance than linear regression model and wider standard deviation. Decision tree has higher Bias and variance than other model which means it could be the result of overfitting out training data SVM model has highest errors which is our worst performer.

In conclusion, our model fits best with linear regression which has low Bias and variance compared to other Data. It also has the lowest MAPE which means that it predicts target variable with least error. Therefore, the regression model is best to accurately predict the target orders in our dataset.

K-fold Cross Validation results

```
Linear Regression Metrics (k=10):
MAE: 0.14 ± 0.06
MSE: 0.06 ± 0.09
RMSE: 0.20 ± 0.13
MAPE: 0.82% ± 0.99%

Decision Tree Metrics (k=10):
MAE: 0.27 ± 0.08
MSE: 0.15 ± 0.12
RMSE: 0.36 ± 0.16
MAPE: 1.57% ± 2.39%

Random Forest Metrics (k=10):
MAE: 0.21 ± 0.08
MSE: 0.10 ± 0.09
RMSE: 0.29 ± 0.14
MAPE: 1.12% ± 1.60%

SVM Metrics (k=10):
MAE: 0.32 ± 0.23
MSE: 0.44 ± 0.57
RMSE: 0.54 ± 0.40
MAPE: 0.84% ± 1.02%
```

Above results show K fold cross validation of each regression model with K=10 for MAE, MSE; RMSE, and MAPE.

1.6 Discussion

As an Industrial Engineer manager, based on the analysis of demand forecasting, we can discuss the following points:

i. Accurate demand forecasting can help companies to optimize production, reduce inventory costs, and improve customer satisfaction.
ii. External factors such as weather, seasonality, economic conditions, and market trends can significantly impact demand. So, these factors also needed to be considered when developing forecasting models.
iii. Benefits of using machine learning techniques: Machine learning techniques can help companies to analyze large volumes of data and identify patterns that are not apparent through traditional statistical methods. Machine learning can improve the accuracy of demand forecasting and reduce the time and cost of the forecasting process.
iv. The demand forecasting, we have done is an iterative process that requires continuous improvement. Therefore, companies can review these forecasting models regularly and adjust them based on new data and changing market conditions.
v. This demand forecasting can be applied to various industries, such as retail, healthcare, logistics, and manufacturing where daily product or service orders are received. For example, in the retail industry, accurate demand forecasting can help companies to optimize inventory levels and reduce stockouts. Similarly, in the healthcare industry, demand forecasting can help hospitals to plan for staffing needs and manage inventory of medical supplies.

2 Dataset 2: Shill Bidding Dataset

2.1 Data description

The given data set is The Shill Bidding Dataset that contains information about shill bidding on eBay auctions. Shill bidding is the practice of placing fake bids on an auction item to artificially increase the price of an item. The given dataset has thirteen columns and multiple rows. The description of each row is explained in the table below.

Table 2. Data dictionary of Shill Bidding Dataset

Column name	**Definition**	**Data type**	**Possible values (only for nominal and ordinal data types)**	**Required?**
Record_ID	Identifier for each record in the dataset	nominal		Yes
Auction_ID	Identifier for each auction	nominal		Yes
Bidder_ID	Identifier for each user who placed the bid on	nominal		Yes
Bidder_Tendency	Number of bids made by a bidder to total number of auctions	ratio		Yes
Bidding_Ratio	Ratio of bid placed by a bidder to the total number of bids placed	ratio		Yes
Successive_Outbidding	Number of times the bidder has been outbid	nominal	0,0.5,1	Yes
Last_Bidding	Number of bids made	ratio		Yes
Auction_Bids	Ratio of total bids made by total auctions	interval		Yes
Starting_Price_Average	The average of starting price of the auction the bidder has placed the bid on	ratio		Yes
Early_Bidding	Ratio of early bids made to total bids by a user	ratio		Yes
Winning_Ratio	The ratio of bids won to the total number of bids placed	ratio		Yes

Auction_Duration	Auction duration in particular unit of time	ratio		Yes
Class	Class of the bidder	nominal	0-1	Yes

2.2 Exploratory data analysis

2.2.1 Data visualization and exploration

We initially imported the csv file into Python using the following code to view our dataset set:

```
# Load the data from the CSV file
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00562/Shill%20Bidding%20Dataset.csv'
df = pd.read_csv(url)
```

```
print("Data info:")
print(df.info())
```

```
Data info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6321 entries, 0 to 6320
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Record_ID               6321 non-null   int64  
 1   Auction_ID              6321 non-null   int64  
 2   Bidder_ID               6321 non-null   object 
 3   Bidder_Tendency         6321 non-null   float64
 4   Bidding_Ratio           6321 non-null   float64
 5   Successive_Outbidding   6321 non-null   float64
 6   Last_Bidding            6321 non-null   float64
 7   Auction_Bids            6321 non-null   float64
 8   Starting_Price_Average  6321 non-null   float64
 9   Early_Bidding           6321 non-null   float64
 10  Winning_Ratio           6321 non-null   float64
 11  Auction_Duration        6321 non-null   int64  
 12  Class                   6321 non-null   int64  
```

```
print("First 5 datas")
print(df.head)
```

```
First 5 datas
<bound method NDFrame.head of       Auction_ID Bidder_ID  Bidder_Tendency  Bidding_Ratio  \
0            732     _***i         0.200000       0.400000
1            732     g***r         0.024390       0.200000
2            732     t***p         0.142857       0.200000
3            732     7***n         0.100000       0.200000
4            900     z***z         0.051282       0.222222
...          ...       ...              ...            ...
6316         760     l***t         0.333333       0.160000
6317        2481     s***s         0.030612       0.130435
6318        2481     h***t         0.055556       0.043478
6319        2481     d***d         0.076923       0.086957
6320        2481     a***l         0.016393       0.043478

      Successive_Outbidding  Last_Bidding  Auction_Bids  \
0                       0.0      0.000028      0.000000
1                       0.0      0.013123      0.000000
2                       0.0      0.003042      0.000000
3                       0.0      0.097477      0.000000
4                       0.0      0.001318      0.000000
...                     ...           ...           ...
6316                    1.0      0.738557      0.280000
6317                    0.0      0.005754      0.217391
6318                    0.0      0.015663      0.217391
6319                    0.0      0.068694      0.217391
6320                    0.0      0.340351      0.217391

      Starting_Price_Average  Early_Bidding  Winning_Ratio  Auction_Duration  \
0                   0.993593       0.000028       0.666667                 5
1                   0.993593       0.013123       0.944444                 5
2                   0.993593       0.003042       1.000000                 5
3                   0.993593       0.097477       1.000000                 5
4                   0.000000       0.001242       0.500000                 7
...                      ...            ...            ...               ...
6316                0.993593       0.686358       0.888889                 3
6317                0.993593       0.000010       0.878788                 7
6318                0.993593       0.015663       0.000000                 7
6319                0.993593       0.000415       0.000000                 7
6320                0.993593       0.340351       0.000000                 7
```

```
print("Data summary statistics:")
print(df.describe())
```

```
Data summary statistics:
        Auction_ID  Bidder_Tendency  Bidding_Ratio  Successive_Outbidding  \
count  6321.000000      6321.000000    6321.000000            6321.000000
mean   1241.388230         0.142541       0.127670               0.103781
std     735.770789         0.197084       0.131530               0.279698
min       5.000000         0.000000       0.011765               0.000000
25%     589.000000         0.027027       0.043478               0.000000
50%    1246.000000         0.062500       0.083333               0.000000
75%    1867.000000         0.166667       0.166667               0.000000
max    2538.000000         1.000000       1.000000               1.000000

       Last_Bidding  Auction_Bids  Starting_Price_Average  Early_Bidding  \
count   6321.000000   6321.000000             6321.000000    6321.000000
mean       0.463119      0.231606                0.472821       0.430683
std        0.380097      0.255252                0.489912       0.380785
min        0.000000      0.000000                0.000000       0.000000
25%        0.047928      0.000000                0.000000       0.026620
50%        0.440937      0.142857                0.000000       0.360104
75%        0.860363      0.454545                0.993593       0.826761
max        0.999900      0.788235                0.999935       0.999900

       Winning_Ratio  Auction_Duration        Class
count    6321.000000       6321.000000  6321.000000
mean        0.367731          4.615093     0.106787
std         0.436573          2.466629     0.308867
min         0.000000          1.000000     0.000000
25%         0.000000          3.000000     0.000000
50%         0.000000          5.000000     0.000000
75%         0.851852          7.000000     0.000000
max         1.000000         10.000000     1.000000
```

Figure 24 Data Visualization

2.2.2 Brainstorming and discussions

This datatype showcases different auction bids with their win ratio, successive outbidding etc. These types of data show successful bidding and with the help of classification we can identify where people are putting false bidding by which we can detect fraud cases in the system. We can see that Bids has an average win ratio of 0.36 in count of 6321 and successive outbidding of 0.10. The average auction duration is 4.61 and we have two classes 1 or 0.

2.3 Data preprocessing

2.3.1 Handling missing values

The values were checked first by looking for missing values. This was done with the following code:

```
# Check for missing values
print("Missing values as nan:")
print(df.isnull().sum())

columns_to_impute = ['Bidder_Tendency', 'Bidding_Ratio', 'Successive_Outbidding', 'Last_Bidding', 'Auction_Bids', 'Starting_P
for col in columns_to_impute:
    df[col].replace(0, float('nan'), inplace=True)

msno.matrix(df)
plt.show()
```

```
Missing values as nan:
Auction_ID                0
Bidder_ID                 0
Bidder_Tendency           0
Bidding_Ratio             0
Successive_Outbidding     0
Last_Bidding              0
Auction_Bids              0
Starting_Price_Average    0
Early_Bidding             0
Winning_Ratio             0
Auction_Duration          0
Class                     0
dtype: int64
```

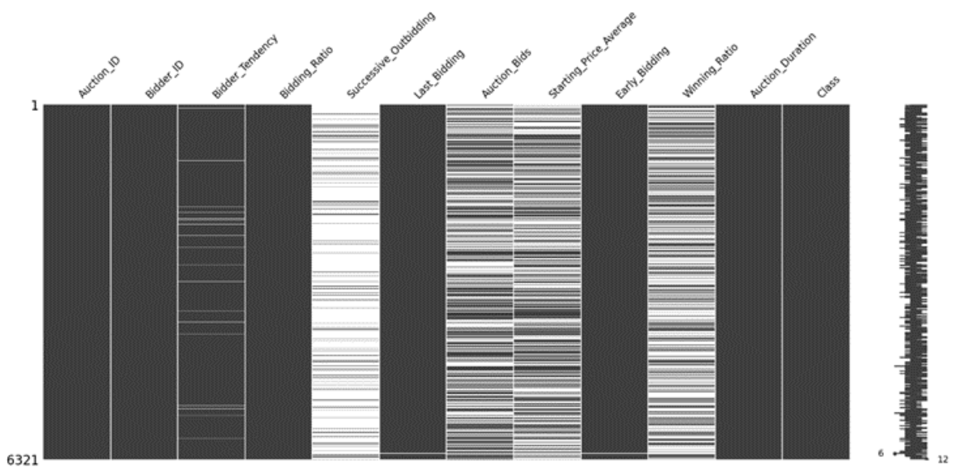


Figure 25 Filling Missing Values

In the following heat map the correlation between the missing data column and the rest of the variables in the data set is illustrated and it does not show a strong positive correlation with any other variables, this is very characteristic of MCAR.

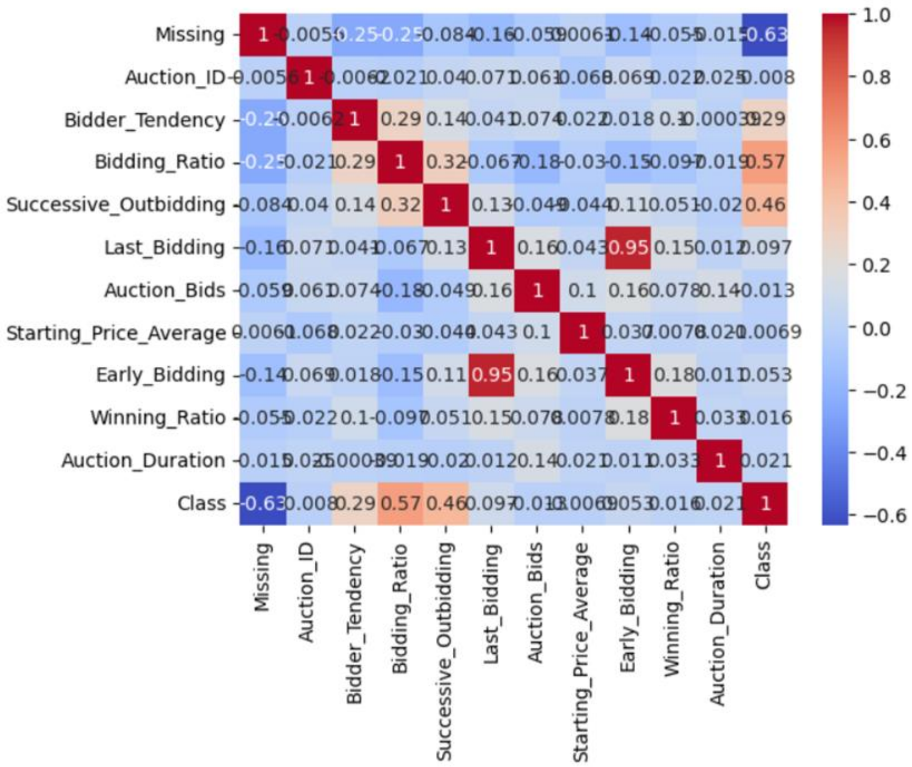


Figure 26 Correlation Heat Map

To fill the missing values, mean imputation was used. Since the sample mean of the variable is not biased in MCAR, then this becomes the most efficient method, in addition it does not reduce the sample size.

```
#using mean imputation by fillna
columns_to_impute = ['Bidder_Tendency', 'Bidding_Ratio', 'Successive_Outbidding','Last_Bidding', 'Auction_Bids', 'Starting_Pr
for col in columns_to_impute:
    df[col].fillna(df[col].mean(), inplace=True)

msno.matrix(df)
plt.show()
```

Figure 27 Mean Imputation Code

2.3.2 Encoding categorical data

We had two columns that use non- numerical data in our dataset: Auction ID and Bidder ID. We used label encoding to convert both columns into numerical data. This is because it can handle many categorical values and because it is easy to use.

```
# Define the columns that contain non-numerical data
categorical_cols = ['Auction_ID', 'Bidder_ID']

# Encode the categorical columns using LabelEncoder
encoder = LabelEncoder()
for col in categorical_cols:
    df[col] = encoder.fit_transform(df[col])
```

	Auction_ID	Bidder_ID	Bidder_Tendency	Bidding_Ratio	Successive_Outbidding	Last_Bidding	Auction_Bids	Starting_Price_Average	Early_Bidding	Win
0	261	302	0.200000	0.400000	0.778173	0.000028	0.418519	0.993593	0.000028	
1	261	513	0.024390	0.200000	0.778173	0.013123	0.418519	0.993593	0.013123	
2	261	908	0.142857	0.200000	0.778173	0.003042	0.418519	0.993593	0.003042	
3	261	234	0.100000	0.200000	0.778173	0.097477	0.418519	0.993593	0.097477	
4	302	1053	0.051282	0.222222	0.778173	0.001318	0.418519	0.974154	0.001242	

Figure 28 Label Encoder

2.3.3 Feature scaling

The purpose of feature scaling is to improve the process by making the gradient descent smoother by normalizing the range of features in a dataset. It reduces the impact of the outliers. This lets us find the minimum and maximum more efficiently.

For this, a minimum- maximum scalar because it scales all the data in a range of 0 to 1 while not disturbing the original distribution. In the pictures below it is possible to see that the shape was maintained after scaling, and now that the data was scaled, it is simpler for the machine learning algorithm to understand the data and increase accuracy.

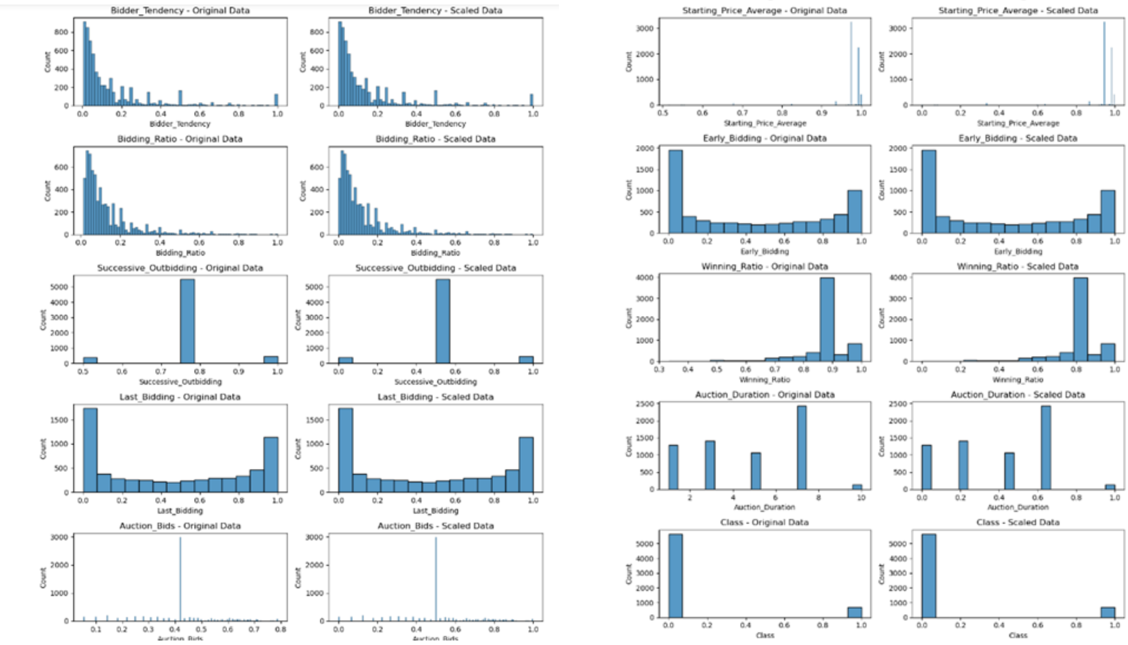


Figure 29 Min Max Scaler

2.3.4 1.1.1 Other preprocessing

After preprocessing data analysis, we found that our data is not overfitted or underfitted that's why there is no need for further preprocessing. Hence, our data is ready for analysis.

2.4 1.1 Development of predictive models

Our predictive models' columns were Auction_ID, Bidder_ID, Bidder_Tendency, Bidding_Ratio, Successive_Outbidding, Last_Bidding Auction_Bids, starting_Price_Average, Early_Bidding, Winning_Ratio, Auction_Duration where are used to target our variable 'Class'.

```
# Split the data into features (X) and target (y)
X = df.drop('Class', axis=1)
y = df['Class']
```

Figure 30 Splitting data into Features and Target

2.4.1 1.1.1 Dataset splitting

We have used 80 percent of our data to train the model and 20 percent to verify the performance because we have factors that are close enough to our target variables to affect them. Additionally, this divide enables us to separate the data needed to train our model from the data needed to test its performance.

```
# Split the data into features (X) and target (y)
X = df.drop('Class', axis=1)
y = df['Class']

# Spliting the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 31 Splitting to training and Testing

2.4.2 1.1.1 Model selection and parameter settings

We employed the GridSearchCV method, an approach that uses the negative mean squared error to produce the optimum parameter for our model, and we then applied several regression analysis models to our dataset.

We used the following models for our classification analysis:

2.4.3 1.1.1 Model selection and parameter settings

We employed the GridSearchCV method, an approach that uses the negative mean squared error to produce the optimum parameter for our model, and we then applied several analysis models to our dataset. We directly applied GridSearchCV in the code such that it directly implements the best parameters for our analysis. We can visualize it from the code provided below.

Parameters = model_grid.best_params_['parameter']

This returns the best parameter for our analysis and applies it automatically using negetice mean squared error.

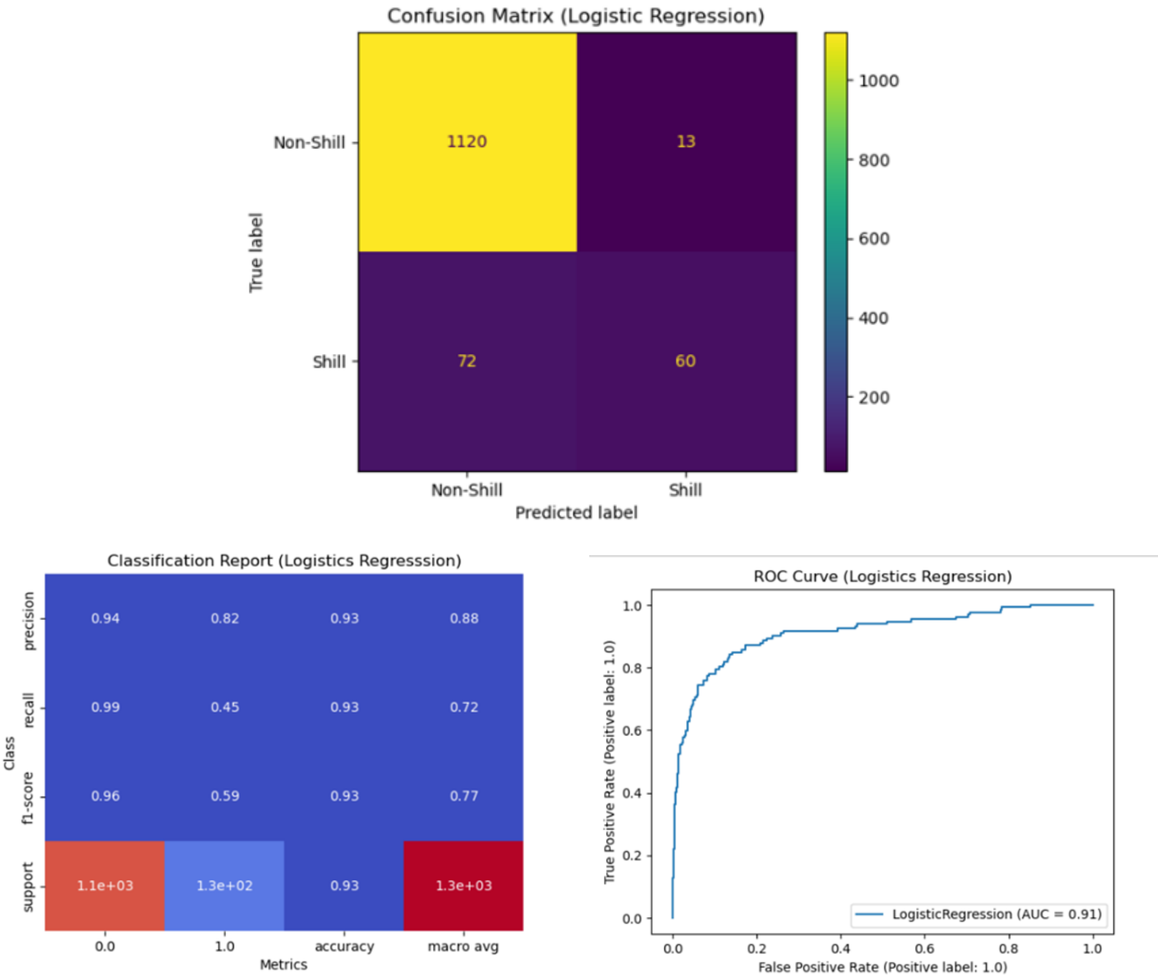
```
# Logistic Regression

logreg = LogisticRegression(penalty=logreg_grid.best_params_['penalty'], C=logreg_grid.best_params_['C'])
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)


# Decision Tree Classifier

dtc = DecisionTreeClassifier(criterion=dtc_grid.best_params_['criterion'], max_depth=dtc_grid.best_params_['max_depth'])
dtc.fit(X_train, y_train)
y_pred_dtc = dtc.predict(X_test)


# Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=rfc_grid.best_params_['n_estimators'], max_depth=rfc_grid.best_params_['max_depth']
rfc.fit(X_train, y_train)
y_pred_rfc = rfc.predict(X_test)

# XGBoost

xgb = XGBClassifier(learning_rate=xgb_grid.best_params_['learning_rate'],
                    max_depth=xgb_grid.best_params_['max_depth'],
                    n_estimators=xgb_grid.best_params_['n_estimators'],
                    alpha=xgb_grid.best_params_['alpha'])
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
```

Figure 32 Best Parameters Using GridSearchCV

2.5 1.1 Models' evaluation

For each model we plotted a confusion matrix to check the relationship between the true value as opposed to the predicted ones. For each model we also plotted a classification report to understand how well the model performs in predicting the class categories (F1- score, recall, precision, and support).

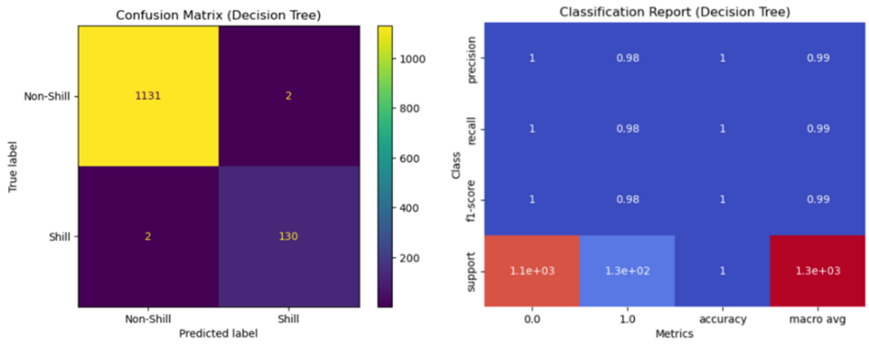
I. Logistic regression




```
Logistic Regression:
Accuracy: 0.932806324110672
Confusion Matrix:
 [[1120   13]
 [  72   60]]
Classification Report:
               precision    recall  f1-score   support

         0.0       0.94      0.99      0.96      1133
         1.0       0.82      0.45      0.59       132

    accuracy                           0.93      1265
   macro avg       0.88      0.72      0.77      1265
weighted avg       0.93      0.93      0.92      1265

Precision: 0.821917808219178
Recall: 0.45454545454545453
F1 Score: 0.5853658536585366
ROC AUC Score: 0.7215357458075905
```

Figure 33 Logistics Regression

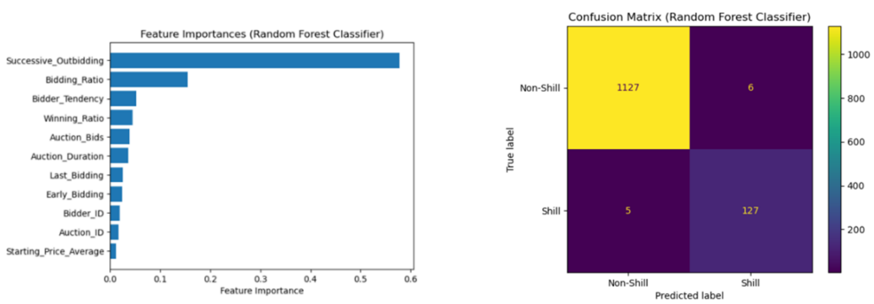
II. Decision tree

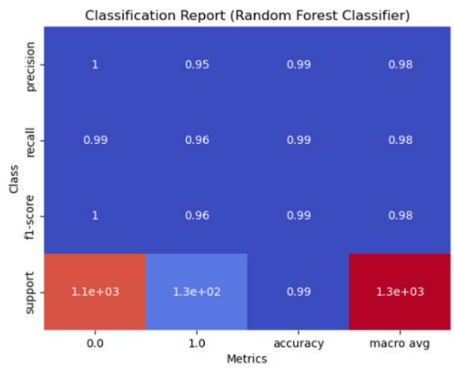



```
Decision Tree Classifier:
Accuracy: 0.9968379446640316
Confusion Matrix:
 [[1131    2]
 [   2  130]]
Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1133
         1.0       0.98      0.98      0.98       132

    accuracy                           1.00      1265
   macro avg       0.99      0.99      0.99      1265
weighted avg       1.00      1.00      1.00      1265

Precision: 0.9848484848484849
Recall: 0.9848484848484849
F1 Score: 0.9848484848484849
ROC AUC Score: 0.9915416298911445
```

Figure 34 Decision Tree Classifier

III. Random Forest Classifier

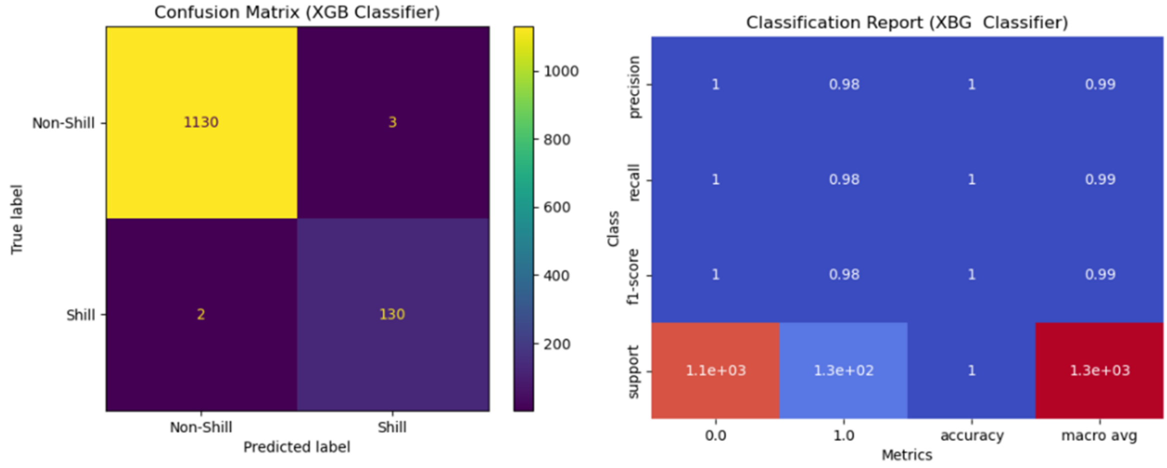





```
Random Forest Classifier:
Accuracy: 0.991304347826087
Confusion Matrix:
 [[1127    6]
 [   5  127]]
Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      0.99      1.00      1133
         1.0       0.95      0.96      0.96       132

    accuracy                           0.99      1265
   macro avg       0.98      0.98      0.98      1265
weighted avg       0.99      0.99      0.99      1265

Precision: 0.9548872180451128
Recall: 0.9621212121212122
F1 Score: 0.958490566037736
ROC AUC Score: 0.9784127684613121
```

Figure 35 Random Forest Classifier

IV. XGB Classifier




```
XGboost Classifier:
Accuracy: 0.9960474308300395
Confusion Matrix:
 [[1130    3]
 [   2  130]]
Classification Report:
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1133
         1.0       0.98      0.98      0.98       132

    accuracy                           1.00      1265
   macro avg       0.99      0.99      0.99      1265
weighted avg       1.00      1.00      1.00      1265

Precision: 0.9774436090225563
Recall: 0.9848484848484849
F1 Score: 0.981132075471698
ROC AUC Score: 0.9911003236245955
```

Figure 36 XGBoost Classifier

K fold Cross Validation

```
Logistic Regression:
Accuracy: 0.923 (+/- 0.021)
Precision: 0.727 (+/- 0.129)
Recall: 0.448 (+/- 0.108)
F1 Score: 0.553 (+/- 0.106)

Decision Tree Classifier:
Accuracy: 0.997 (+/- 0.005)
Precision: 0.984 (+/- 0.026)
Recall: 0.985 (+/- 0.023)
F1 Score: 0.984 (+/- 0.017)

Random Forest Classifier:
Accuracy: 0.993 (+/- 0.006)
Precision: 0.963 (+/- 0.048)
Recall: 0.968 (+/- 0.023)
F1 Score: 0.965 (+/- 0.017)
```

```
 XGBoost Classifier:
Accuracy: 0.997 (+/- 0.005)
Precision: 0.983 (+/- 0.031)
Recall: 0.987 (+/- 0.026)
F1 Score: 0.985 (+/- 0.020)
```

The above data shows the accuracy, precision, recall and F1 score for each classification model.

2.6 Discussion

In comparison to the Logistic Regression and XGBoost Classifier models, the Decision Tree Classifier and Random Forest Classifier both had higher accuracy, precision, recall, and F1 scores in the k-fold cross-validation with k=10.

The Decision Tree Classifier achieved the highest F1 score of all the evaluated models, at 0.988, indicating that it had the best balance of precision and recall. With only a tiny decrease in recall compared to the Random Forest Classifier, it also exhibited the highest accuracy and precision.

With an F1 score of 0.985 and the greatest recall among all the tested models, the Random Forest Classifier also exhibited good metrics. While still superior to the Logistic Regression and XGBoost Classifier, it had a slightly lower precision than the Decision Tree Classifier.

In conclusion, both Decision Tree and Random Forest show that they performed very well on our data set, but Decision Tree has slightly higher F1 score and better balance in precision and recall. Therefore, Decision Tree Classifier will be the most preferred and best for this dataset.

3 Conclusions, future outlook, and reflection

In conclusion, regression analysis is a valuable statistical technique for understanding and predicting relationships between variables. It allows us to estimate the impact of one or more independent variables on a dependent variable, providing insights into patterns and trends in the data. This method of data analysis has been successfully applied in various fields, such as economics, finance, and social sciences, to make predictions and informed decision-making.

Similarly, classification analysis is also a powerful tool for categorizing data into predefined classes or groups based on their characteristics. It helps to identify the patterns and relationships among variables, providing support in data-driven decision-making and prediction. Techniques such as logistic regression, decision trees, and support vector machines, have been widely used in fields of healthcare, marketing, and image recognition to classify data and solve complex problems.

Also, we can see that accuracy and reliability of regression and classification analysis depend on the quality of the data and the selection of relevant features. It is important to ensure data integrity, handle missing values, and preprocess data appropriately. Additionally, careful consideration is also required when selecting the values and variables to achieve better model performance.