BackOrder Prediction

By

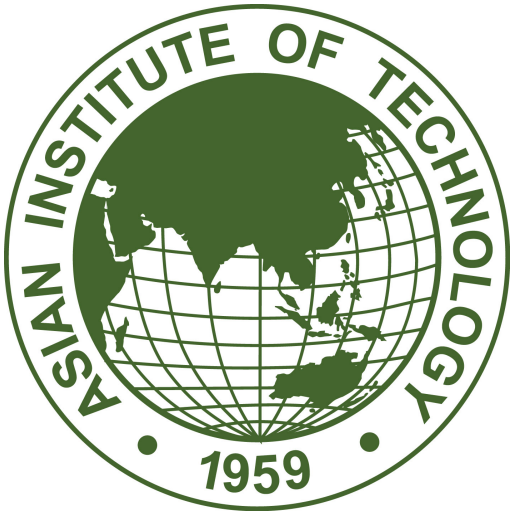
Group 13

Ashmita Phuyal st124454

Sitthiwat Damrongpreechar st123994

A Proposal Submitted for Computer Programming for

Data Science and Artificial Intelligence Project



Asian Institute of Technology

School of Engineering and Technology

Thailand

1st November, 2023

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Background and Motivation

When a customer orders a product, which is not available in the inventory or temporarily out of stock, and the customer decides to wait until the product is available and guaranteed to be dispatched, then this scenario is called backorder. Backorders if not properly handled will negatively affect a company's income, share price, and consumer trust which result in the loss of a client or selling order. This is a crucial component of supply chain management and inventory optimization, and it involves forecasting the likelihood of certain products being temporarily unavailable due to insufficient stock levels. Due to traditional inventory management systems, frequently caused stockouts, manufacturing delays, and dissatisfied clients have created the impact in business leading to heavy loss. With the advancement in technology and machine learning, it is possible to develop more accurate and proactive backorder prediction models.

The objective for the backorder prediction implementation is to streamline the inventory and supply chain management process. Companies' target is to avoid the expensive and ineffective practice of either overstocking or understocking their products, as this may result in lower market competencies, earnings and lost opportunities. Predicting backorders help companies to maintain the balance in the inventory to satisfy consumer needs eliminating surplus stock and related holding costs.

1.2 Business Overview

The challenge of precisely forecasting and managing inventory to avoid surpluses in order to minimize the costs associated with carrying excess inventory is the main issue with backorder prediction. Variable customer demand, unstable times for suppliers, and the balance between inventory levels and their cost constraints are the highlights of this issue. The main purpose is to maintain high customer satisfaction while avoiding the expense of carrying excess inventory by preventing stockouts. It takes sophisticated data analysis, predictive modeling, and efficient supply chain process integration to achieve this balance, which makes it a difficult and crucial task for companies in a variety of sectors.

1.3 Impacts

Implementing a backorder prediction provides advantages on a large scale in the business areas. The end users like companies can effectively manage expectations of the customers by communicating proactively in order to meet the potential delivery delays. By reducing excess inventory, the companies can generate significant savings in terms of storage. As a result, businesses can ensure that products are available when needed, which could increase sales and overall revenue and can be able to better meet customer demand. The system enhances overall supply chain efficiency and fosters better supplier relations.

2. PROBLEM STATEMENT

The customers purchasing the product are unfamiliar about the behavior of the backorder policies. The e-commerce companies usually face the challenges of frequent backorders which created a huge impact on customer satisfaction, substantial costs and inventory management. The dynamic market conditions, fluctuation in customer demands, and diverse products are the main problems of backorder. In order to solve this problem, we need to develop a robust predicting system selecting the best possible models to predict the orders.

2.1 Objective

Creating a system that can precisely predict and handle situations in which product stock levels are anticipated to drop below customer demand is the main goal of backorder prediction. This involves the proactive identification of potential backorders, allowing businesses to take timely and informed actions to prevent or mitigate stockouts. The key objectives include:

- Preventing Stockouts: The main goal is to stop or reduce the number of times that products are out of stock when customers place orders. This will help to keep them happy and prevent revenue loss.

- Reducing Superfluous Stock: In addition to avoiding stockouts, the goal is to steer clear of overstocking, which can lead to needless holding expenses, space usage, and capital invested in inventory.

- Improving Inventory Management: Creating data-driven models and procedures to guarantee that the appropriate number of goods is available when needed, taking into account variables such as lead times for supplies and fluctuations in demand.

- Improving Customer Satisfaction: Keeping high levels of customer satisfaction requires managing customer expectations through efficient communication in backorder situations.

2.2 Scope

Backorder prediction covers a wide range of topics and factors in the context of inventory control and supply chain management. It has a dynamic application that evolves as technology advances and the market changes. It includes an in-depth approach to deal with the issues related to customer satisfaction and inventory management in a variety of industries. It comprises:

- Demand Forecasting: Using demand forecasting methods to project future trends and requirements from customers.

- Real-Time Data Integration: For more dynamic forecasts, the scope includes integrating real-time data sources, such as supplier performance metrics and market conditions.

- Cost optimization: Process of determining the best course of action for inventory management by weighing the costs of backorders against the costs of keeping inventory.

- Market dynamics : Understanding the effects of shifting consumer preferences and market conditions on backorder projections

3. RELATED WORKS

Prediction of backorder is a challenging task as it is uncertain and varies with time. In recent times, numerous research papers and studies have contributed to the advancement of backorder prediction in the fields of supply chain management, and inventory control. Several machine learning models have been developed to find the best prediction for the backorders.

In a recent study by Rodrigo, Eduardo , and Leonardo [1], the researcher provides insights into predictive modeling techniques for backorder prediction, emphasizing the use of machine learning algorithms to improve accuracy. The paper addresses the complexities of model building and the importance of precise backorder forecasts for companies looking to maximize consumer satisfaction and inventory control.

The latest research study have been conducted by Samiul Islam and Saman Hassanzadeh Amin using Distributed Random Forest (DRF) and Gradient Boosting Machine (GBM) [2] and have observed the performances of the machine learning models to showcase how these models can be used to predict the probable backorder products before actual sales take place. In this study, a ranged technique is proposed to solve this issue. Both machine learning models have been trained on actual and ranged data, and their respective performances have been compared. According to the comparison result, training the models with the ranged data improves their performance by about 20%.

In a broader context and based on real life, Hui Gao, Quanhui Ren and Chunfeng Lv [3] have explored the potential of neural networks and naive bayes, to predict backorder products. In this study, the likely backorder goods have been forecasted using two machine learning algorithms. Upon receiving the experimental results using three statistical criteria—accuracy, precision, misclassification rate, and ROC graphs, each of the results were examined to be proved that it attains a new level of state-of-the-art performance.

In order to improve inventory management, customer satisfaction, and operational efficiency, these research papers collectively offer insightful perspectives and methodologies to the field of backorder prediction. They provide insights on light on a variety of topics, including machine learning techniques, data mining, neural network models, and real-time strategies.

4. DATASETS

The primary source of the dataset is from Kaggle Datasets [4] which contains information of an company's products for the 8 weeks prior to the week that is going to be predicted. he dataset consists of two main files: train.csv, which serves as the training set, and test.csv, used for testing purposes. These products are characterized by various attributes that are crucial for inventory management and risk assessment as they contain various attributes related to the company's products.

This train dataset contains information on various health-related attributes with 23 features of data and consists of records. Eight features—including the target variable—are categorical, while the remaining 15 are numerical, according to the dataset. The product identifier, or sku, appears in the first column and is distinct for every dataset.

5. METHODOLOGY

5.1 Exploratory Data Analysis (EDA)

- Data Considerations:

The dataset consists of 1,929,937 rows and 23 columns. These columns contain a mix of numerical and categorical features, with one column containing missing values that we will address in the imputation section.

Here's a breakdown of the data considerations:

- Data Shape: The dataset contains 1,929,937 rows and 23 columns.

- Column Types: Among the columns, 15 are numerical features, and 8 are categorical features.
- Categorical Features: The categorical features are 'sku', 'potential_issue', 'deck_risk', 'oe_constraint', 'ppap_risk', 'stop_auto_buy', 'rev_stop', and 'went_on_backorder.' The field: 'sku' is considered a product identifier and is not used in further analysis.
- Numerical Features: The numerical features include 'national_inv,' 'lead_time,' 'in_transit_qty,' 'forecast_3_month,' 'forecast_6_month,' 'forecast_9_month,' 'sales_1_month,' 'sales_3_month,' 'sales_6_month,' 'sales_9_month,' 'min_bank,' 'pieces_past_due,' 'perf_6_month_avg,' 'perf_12_month_avg,' and 'local_bo_qty.'
- Missing Values: The features 'perf_6_month_avg' and 'perf_12_month_avg' contain missing values denoted by -99.0. We have done imputation to remove the missing values from these two fields.
- Data Cleaning: The two rows with all missing values for all columns have been removed from the dataset. After the removal of these rows, there are no more missing values in the dataset. The data cleaning process has ensured the quality, integrity, and usefulness of the data for further analysis and modeling.

- Univariate Analysis:

The subplot we used showcases the data distribution for numerical and categorical features in the dataset. We used Kernel Density Estimate (KDE) Plots for numerical features and count plots for categorical features. Most of the numerical features are right-skewed, with a tail extending to the right. Hence, the majority of data points are concentrated on the lower end of the scale, while some data points have higher values, creating a long right tail.

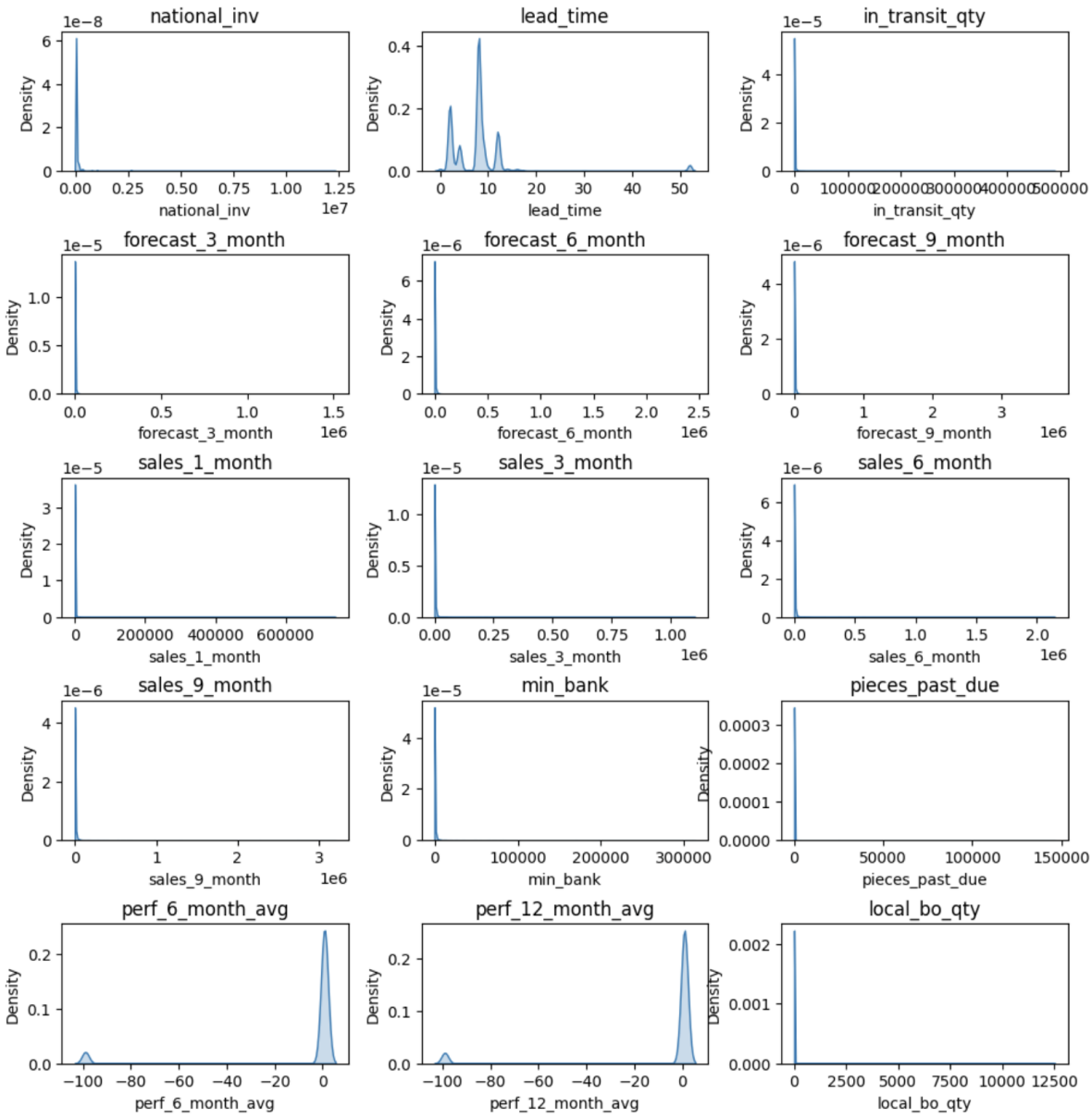


Figure 2: KDE plot for numerical features

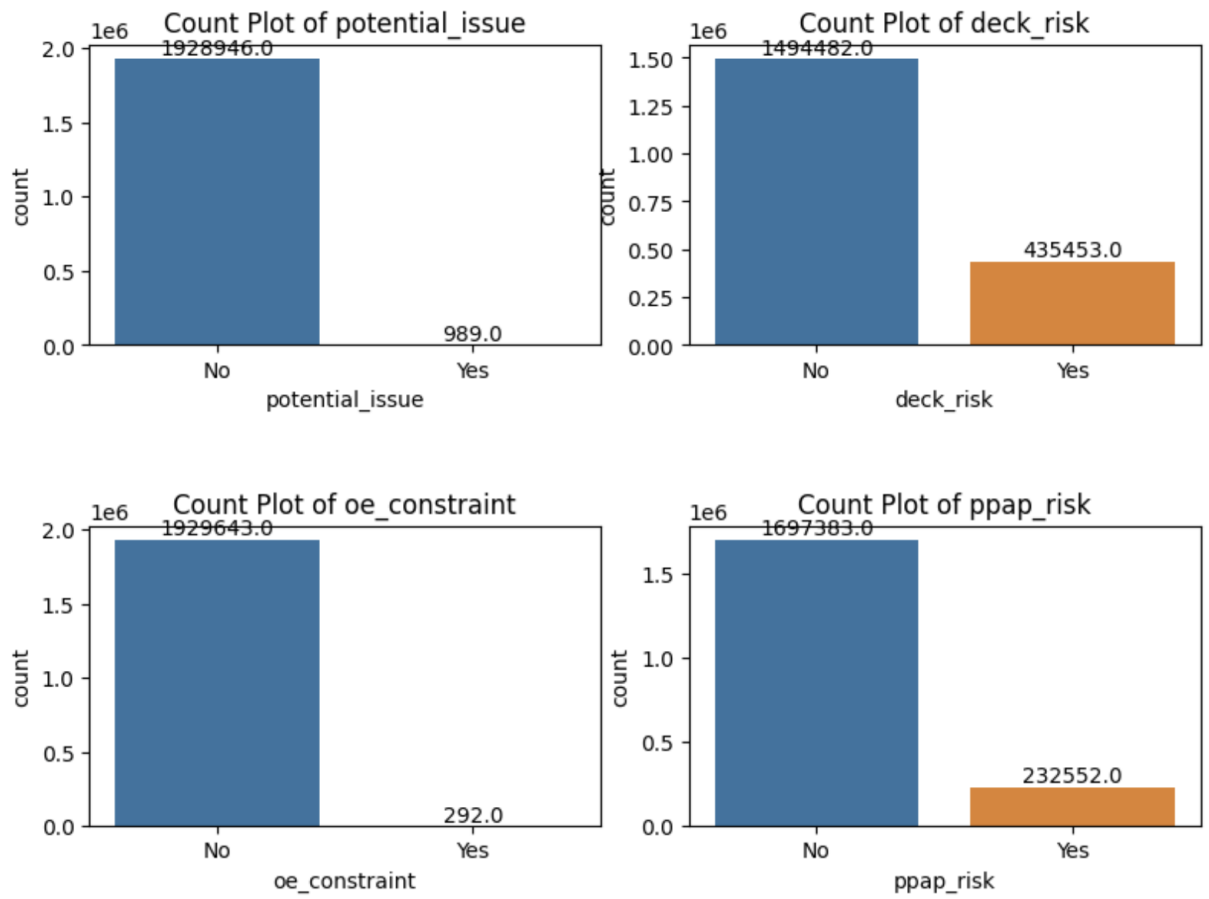


Figure 3: Countplot for categorical features

- Multivariate Analysis
    - Correlation Matrix and Predictive Power Score (PPS)

The correlation matrix heatmap focuses on linear relationships and is useful for feature selection. The below heat map shows that the significant correlations in your data are positive. Forecast_3_month, forecast_6_month and forecast_9_month have strong correlations between each other with the correlation coefficient as 0.99. Also, sales_1_month, sales_3_month, sales_6_month, and sales_9_month are strongly correlated with each other, with correlation coefficients ranging from 0.82 to 0.98. The two variables : perf_6_month_avg and perf_12_month_avg are highly correlated with a coefficient of 0.97

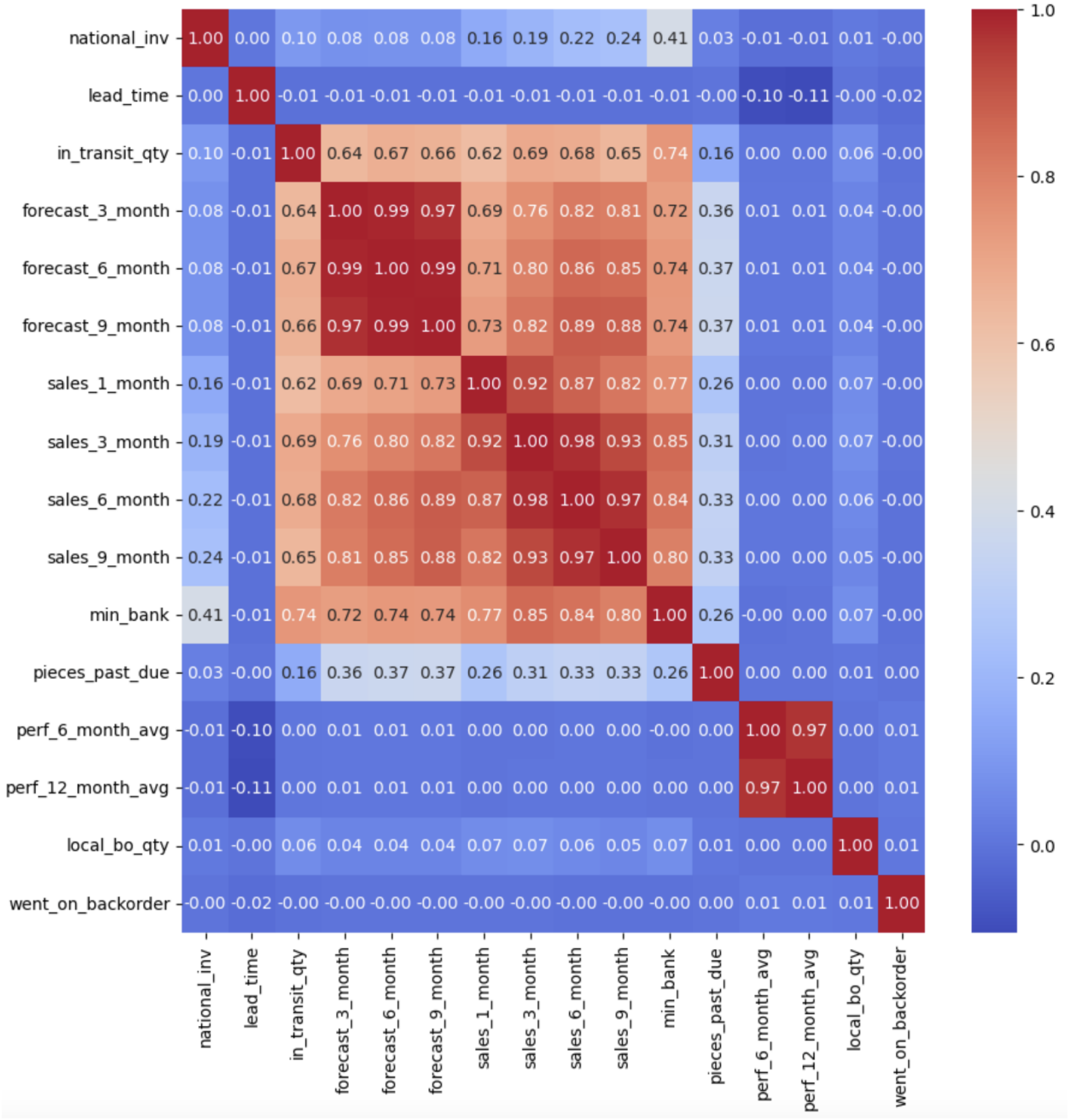


Figure 4: The Correlation HeatMap

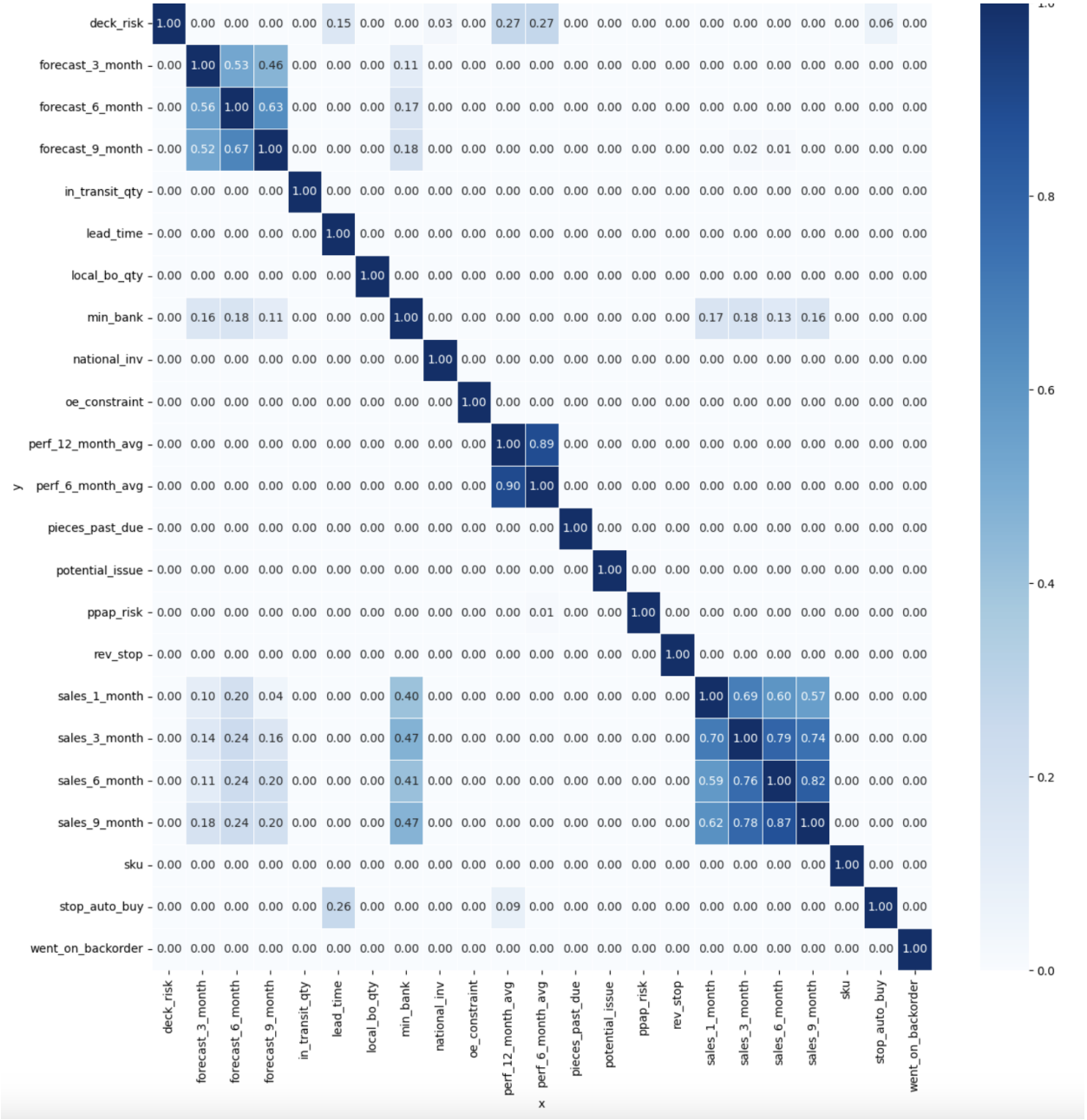


Figure 5: The HeatMap of Power Predictive Score

5.2 Preprocessing

Several crucial steps are carried out in the data preprocessing process for modeling. First, feature scaling is done with standardization techniques in order to ensure that numerical variables are on a common scale. Secondly, we convert categorical variables into a numerical representation that

machine learning algorithms can comprehend, feature encoding is required. For this, we use one-hot encoding or label encoding based on the data. Then, feature selection uses methods such as feature importance ranking or domain expertise to determine which variables are most pertinent to the particular predictive task. Last but not least, data splitting is an essential stage that involves splitting the dataset into training and testing sets (such as an 80-20 or 70-30 split) to assess the model's performance and make sure it can adapt to new situations.

- Data Splitting - Due to a large dataset of backorder, we've chosen to perform a 70:30 data split.
- Imputation - We have performed imputation to handle missing and special values (-99.0) in your dataset, specifically for the features 'lead_time,' 'perf_6_month_avg,' 'perf_12_month_avg,' and 'national_inv.' Here's a summary of the imputation process:
    - We're using the `SimpleImputer` from scikit-learn with the `strategy='most_frequent'` to fill missing values (NaN) in the 'lead_time' feature.
    - For the 'perf_6_month_avg', 'perf_12_month_avg', 'national_inv' features, you're also using the `SimpleImputer` with the 'most_frequent' strategy to replace the special value -99.0 with the most frequent value.
- Standardization and Scaling - We are planning to apply Min-Max scaling to a set of numerical features in our backorder dataset and follow the below steps:
- We need to first identify numerical features and select the features for scaling. We will then apply the Min-Max scaling to both the training and testing datasets (`X_train` and `X_test`) by using MinMaxScaler. If the values in the scaled features now fall within the range of 0 to 1, then it would be suitable for modeling.

In summary, Min-Max scaling is essential for ensuring that numerical features with different ranges are brought to a consistent scale, which is a fundamental step in data preprocessing to improve the performance and stability of machine learning models. The `ColumnTransformer` aids in streamlining this process and maintaining data integrity.

- Encoding - We will create a comprehensive data preprocessing pipeline, combining several steps of 'imputation',' scaling', 'encoding' and 'prediction' into a single pipeline.

This preprocessing pipeline allows you to efficiently and consistently prepare your data for machine learning, ensuring that it is in a suitable format for model training and evaluation. We will experiment with various machine learning models and algorithms to identify the one that provides the best backorder prediction.

- Resampling - Resampling techniques are used to address the issue of class imbalance in machine learning, particularly in this binary classification project where one class has significantly fewer examples than the other. Both oversampling and undersampling techniques are considered to balance the class distribution in the training data, enabling the algorithms to learn from the training data effectively.

- Pipeline - The pipeline process is employed to consolidate all of the preprocessing and modeling steps into a single workflow. This pipeline facilitates the ease of maintenance, reproducibility, and deployment of the model. It also helps prevent data leakage, which can occur accidentally during preprocessing.

6. PRELIMINARY RESULTS

In the preliminary results, the dataset was utilized to conduct exploratory data analysis, perform basic preprocessing, and make simple model predictions. These preliminary results were achieved using Jupyter Notebook with the Python programming language. The steps for the preliminary results are as follows.

1. Import Libraries:

   In the first step, the libraries that were used to perform the exploratory data analysis, preprocessing, pipelining, and prediction (as illustrated in Figure 6). The main libraries are pandas, numpy, sklearn, seaborn, ppscore, and matplotlib.

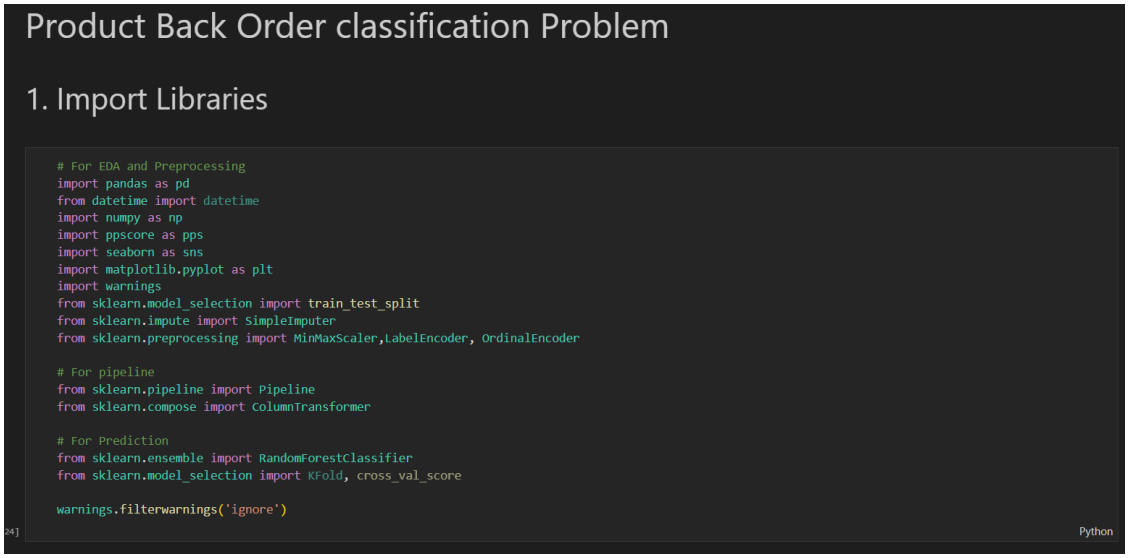


Figure 6: The imported libraries that used to perform the preliminary result.

2. Loading Dataset:

The two files of the dataset are loaded as the Pandas DataFrame using Pandas functions and then combined for exploratory data analysis (as illustrated in Figure 7).

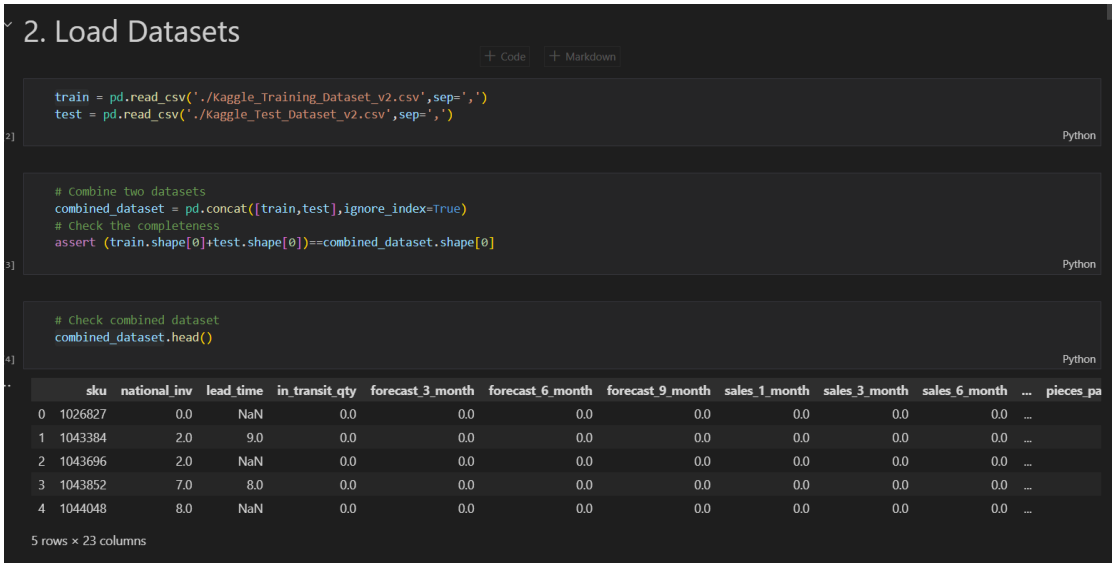


Figure 7: Loaded and Combined datasets

3. Exploratory Data Analysis (EDA)

- Dataset Considerations: The dataset is inspected in terms of its shape, data types, column names, and statistics. Numerical and categorical columns for later use are

then determined. During the statistical inspection (using the describe() function), it's observed that the features 'perf_6_month_avg' and 'perf_12_month_avg' have missing values, which appear to be filled with -99.0. Furthermore, two rows in the dataset contain missing values, and they are dropped. Finally, the target feature is encoded using Label Encoder for use in both Univariate and Multivariate analyses. The illustrations for Dataset Considerations range from Figure 8 to Figure 16.

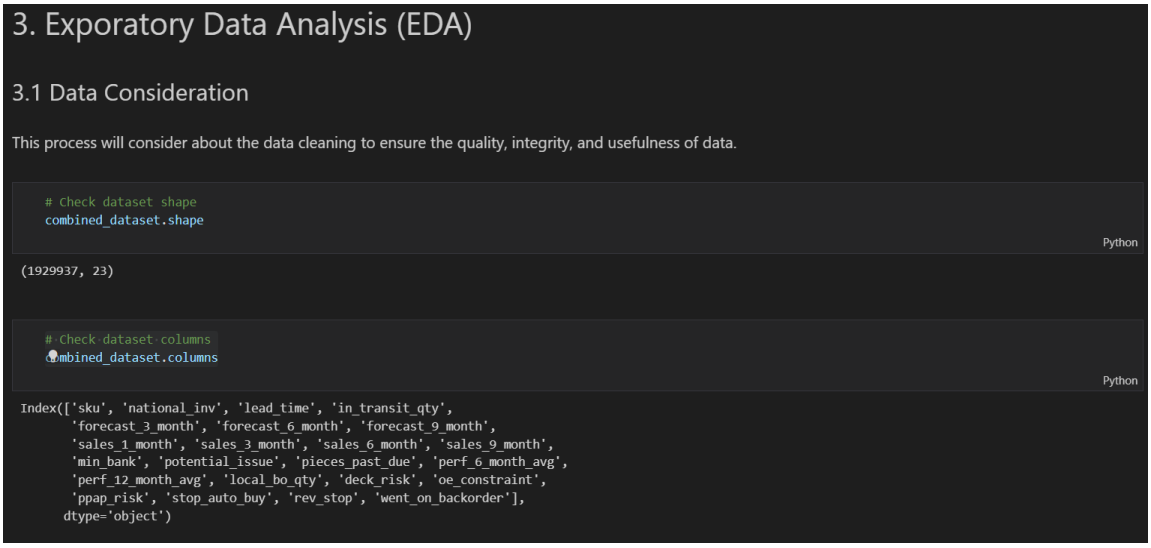


Figure 8: Shape and Columns checking

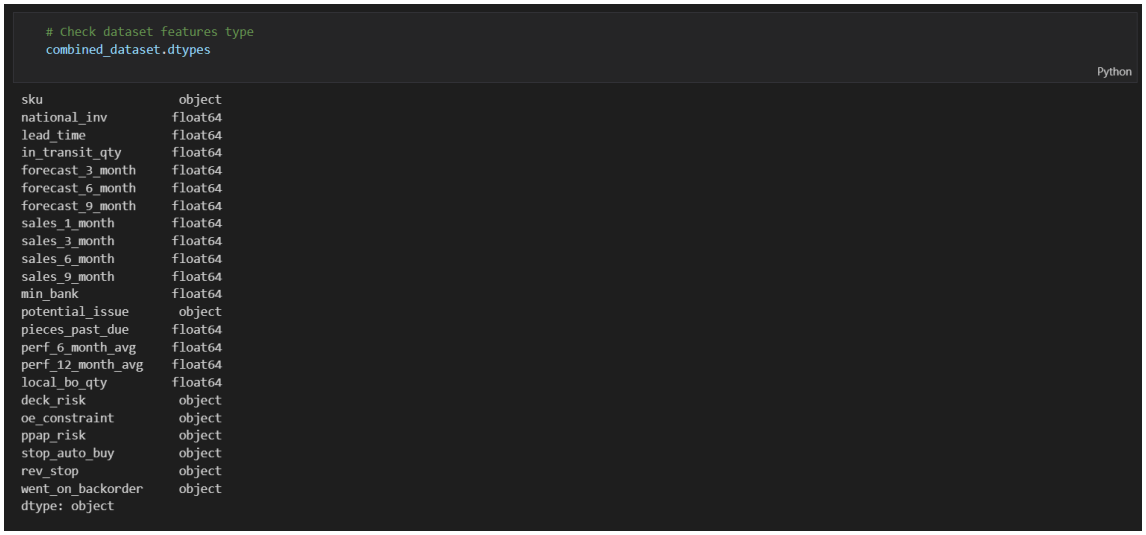


Figure 9: Check the dataset's features type

```python
# Checking numerical and categorical features
numerical_cols = []
categorical_cols = []

for column in combined_dataset.columns:
    if pd.api.types.is_numeric_dtype(combined_dataset[column]):
        numerical_cols.append(column)
    else:
        categorical_cols.append(column)

print(f'Numerical Features: {len(numerical_cols)} ,{numerical_cols}')
print(f'Categorical Features: {len(categorical_cols)} ,{categorical_cols}')
```

✓ 0.0s Python

```
Numerical Features: 15 ,['national_inv', 'lead_time', 'in_transit_qty', 'forecast_3_month', 'forecast_6_month', 'forecast_9_month', 'sales_1_month', 'sale
Categorical Features: 8 ,['sku', 'potential_issue', 'deck_risk', 'oe_constraint', 'ppap_risk', 'stop_auto_buy', 'rev_stop', 'went_on_backorder']
```

```python
# Redefined categorial features (we not used 'sku' which is product ID features)
categorical_cols.remove('sku')
```

✓ 0.0s Python

Figure 10: Separated the numerical and categorical features

```python
# Describe the combined dataset
combined_dataset.describe()
```

Python

	sales_1_month	sales_3_month	sales_6_month	sales_9_month	min_bank	pieces_past_due	perf_6_month_avg	perf_12_month_avg	local_bo_qty	went_on_backorder
	1.929935e+06	1.929935e+06	1.929935e+06	1.929935e+06	1.929935e+06	1.929935e+06	1.929935e+06	1.929935e+06	1.929935e+06	1.929937e+06
	5.536816e+01	1.746639e+02	3.415653e+02	5.235771e+02	5.277637e+01	2.016193e+00	-6.899870e+00	-6.462343e+00	6.537039e-01	7.246351e-03
	1.884377e+03	5.188856e+03	9.585030e+03	1.473327e+04	1.257968e+03	2.296112e+02	2.659988e+01	2.588343e+01	3.543230e+01	8.482875e-02
	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	-9.900000e+01	-9.900000e+01	0.000000e+00	0.000000e+00
	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	6.300000e-01	6.600000e-01	0.000000e+00	0.000000e+00
	0.000000e+00	1.000000e+00	2.000000e+00	4.000000e+00	0.000000e+00	0.000000e+00	8.200000e-01	8.100000e-01	0.000000e+00	0.000000e+00
	4.000000e+00	1.500000e+01	3.100000e+01	4.700000e+01	3.000000e+00	0.000000e+00	9.600000e-01	9.500000e-01	0.000000e+00	0.000000e+00
	7.417740e+05	1.105478e+06	2.146625e+06	3.205172e+06	3.133190e+05	1.464960e+05	1.000000e+00	1.000000e+00	1.253000e+04	2.000000e+00

Features 'perf_6_month_avg' and 'perf_12_month_avg' have the -99.0 which seem to be missing values and filled with -99.0

Figure 11: Statistics details in each feature of the datasets

```python
# Counts -99.0 in feature 'perf_6_month_avg'
combined_dataset['perf_6_month_avg'].value_counts()
```

Python

```
 0.99     163323
 1.00     150339
-99.00    148579
 0.73     128818
 0.98      97390
           ...
 0.20        921
 0.03        829
 0.04        724
 0.01        648
 0.29        572
Name: perf_6_month_avg, Length: 102, dtype: int64
```

```python
# Counts -99.0 in feature 'perf_12_month_avg'
combined_dataset['perf_12_month_avg'].value_counts()
```

Python

```
 0.99     152682
-99.00    140025
 0.78     131353
 0.98     106119
 0.97      74113
           ...
 0.23        895
 0.06        873
 0.05        743
 0.03        639
 0.02        437
Name: perf_12_month_avg, Length: 102, dtype: int64
```

Figure 12: Checking the -99.0 values in feature ‘perf_6_month_avg’ and ‘perf_12_month_avg’

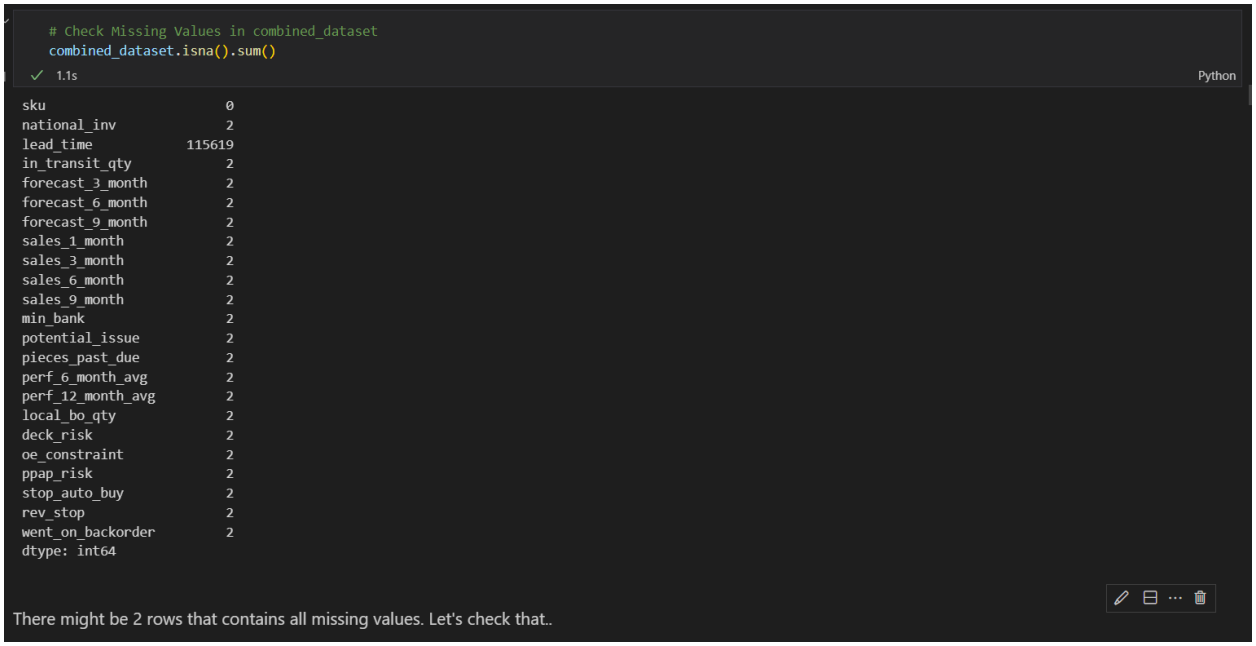


Figure 13: Checking the number of missing values in each feature.

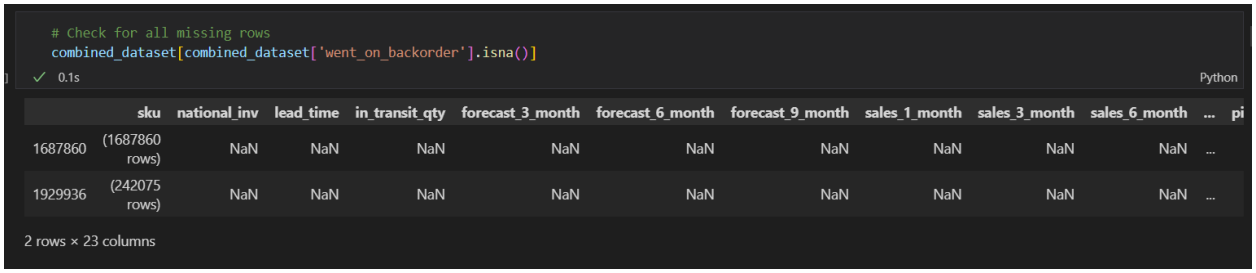


Figure 14: Checking the rows that contains missing values.

```python
# Drop 2 missing rows
combined_dataset.drop(combined_dataset[combined_dataset['sku'] == '(1687860 rows)'].index,inplace=True)
combined_dataset.drop(combined_dataset[combined_dataset['sku'] == '(242075 rows)'].index,inplace=True)
```

✓ 0.6s Python

```python
# Recheck missing rows again
combined_dataset.isna().sum()
```

Python

```
sku                   0
national_inv          0
lead_time        115617
in_transit_qty        0
forecast_3_month      0
forecast_6_month      0
forecast_9_month      0
sales_1_month         0
sales_3_month         0
sales_6_month         0
sales_9_month         0
min_bank              0
potential_issue       0
pieces_past_due       0
perf_6_month_avg      0
perf_12_month_avg     0
local_bo_qty          0
deck_risk             0
oe_constraint         0
ppap_risk             0
stop_auto_buy         0
rev_stop              0
went_on_backorder     0
dtype: int64
```

Figure 15: Drop the rows that contains missing values and recheck again.

```python
# LabelEncoder the target features
target_col = LabelEncoder().fit_transform(combined_dataset['went_on_backorder'])
combined_dataset['went_on_backorder']= target_col
```

Python

Figure 16: Label encoding the target feature for later on analysis.

- Univariate Analysis and Multivariate Analysis:

  The Kernel Density Estimate plots (KDE plots) and Count plots are used to conduct univariate analysis. For multivariate analysis, a correlation heatmap, power predictive score heatmap, and outliers function are created. The corresponding codes are provided in figures, ranging from Figure 17 to Figure 22.

3.2 Univariate Analysis

```python
# Check the skrew of data
# Create subplots for kde plots
fig, axes = plt.subplots(nrows=5, ncols=3, figsize=(10, 10))
fig.tight_layout()

# Iterate through the list and create kde plots
for i, k in enumerate(numerical_cols):
    row, col = divmod(i, 3)
    sns.kdeplot(data = combined_dataset,x=k, fill=True, ax=axes[row, col])
    axes[row, col].set_title(k)
plt.subplots_adjust(hspace=0.6)
plt.show()
```

Python

Figure 17: Plotting the numerical features as the KDE plots.

```python
# Countplot for categorical features
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(8, 12))
fig.tight_layout()

# Iterate through the list of categorical features and create count plots
for i, feature in enumerate(categorical_cols):
    row, col = divmod(i, 2)
    ax = sns.countplot(data=combined_dataset, x=feature, ax=axes[row, col])
    # Add count annotations above each bar
    for p in ax.patches:
        ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='bottom')
    axes[row, col].set_title(f'Count Plot of {feature}')
plt.subplots_adjust(hspace=0.6)
plt.show()
```

Python

Figure 18: Plotting the categorical features as the count plots.

3.3 Multivariate Analysis

```python
#Calculate the outliers
def outlier_count(col, data):

    # calculate your 25% quatile and 75% quatile
    q75, q25 = np.percentile(data[col], [75, 25])

    # calculate your inter quatile
    iqr = q75 - q25

    # min_val and max_val
    min_val = q25 - (iqr*1.5)
    max_val = q75 + (iqr*1.5)

    # count number of outliers, which are the data that are less than min_val or more than max_val calculated above
    outlier_count = len(np.where((data[col] > max_val) | (data[col] < min_val))[0])

    # calculate the percentage of the outliers
    outlier_percent = round(outlier_count/len(data[col])*100, 2)

    if(outlier_count > 0):
        print("\n"+15*'-' + col + 15*'-'+"\n")
        print('Number of outliers: {}'.format(outlier_count))
        print('Percent of data that is outlier: {}%'.format(outlier_percent))
```

Python

```python
#Loop the outliers function to find the percent of data that is outliers
for col in combined_dataset[numerical_cols]:
    outlier_count(col,combined_dataset)
```

Python

Figure 19: Defining outliers percentage function for measuring the percentages of outliers in numerical features.

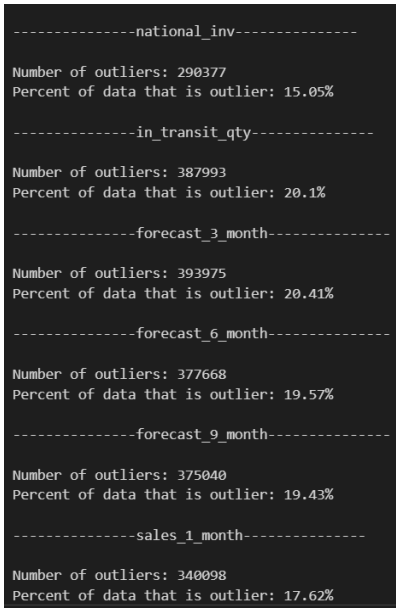


Figure 20: The result examples of the percentages of outliers in each numerical features.

```python
# Correlation Matrix
plt.figure(figsize=(10,10))
sns.heatmap(combined_dataset.corr(), fmt=".2f", annot=True, cmap="coolwarm")
```

Python

Figure 21: Plotting the heatmap of the correlation matrix

```python
# Predictive Power Score
#Create another dataframe to used with pps (just in case that adjusting the features for this)
df_pps=combined_dataset.copy()
matrix_df_pps = pps.matrix(df_pps)[['x', 'y', 'ppscore']].pivot(columns='x', index='y', values='ppscore')
plt.figure(figsize=(15,15))
sns.heatmap(matrix_df_pps, vmin=0, vmax=1, fmt=".2f", cmap="Blues", linewidths=0.5, annot=True)
```

Python

Figure 22: Plotting the heatmap of the predictive power score

4. Preprocessing

- Train and Test Datasets Splitting:

  The dataset is separated into X and y variables based on the target and non-target features. The train_test_split function is utilized to split the data into training and testing sets with a 70:30 ratio. Furthermore, the shapes of all the test and training datasets are verified. The code for this process is illustrated in Figure 23.

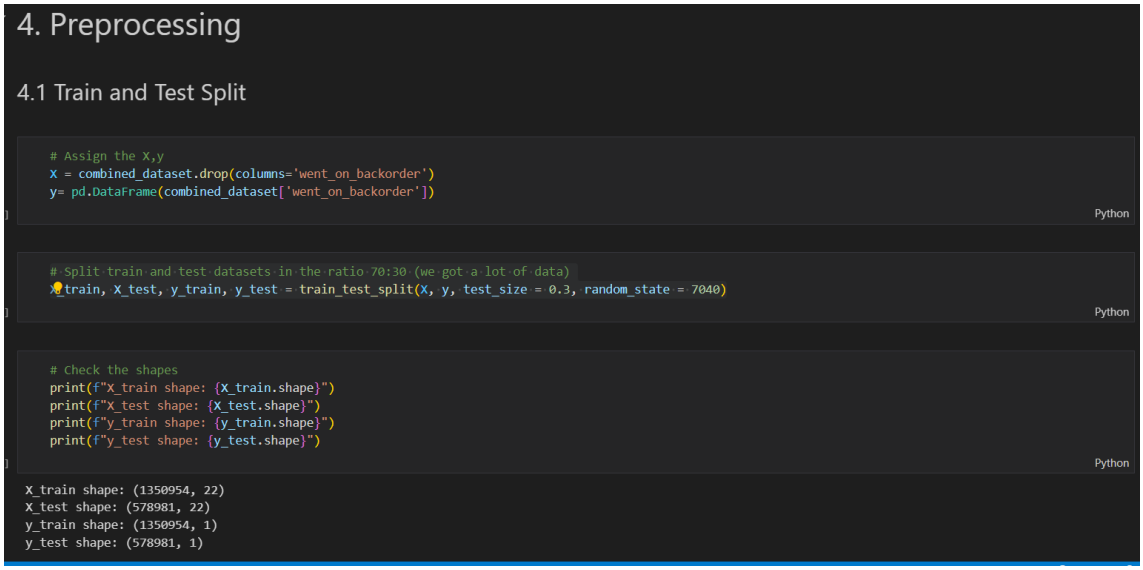


Figure 23: Splitting training and testing datasets then verifying their shape.

- Imputation:

  Imputation begins by checking all missing values, including the value '-99.00'. The SimpleImputer function from the scikit-learn library is then used for imputation. After fitting and transforming the datasets, the validation of imputation is verified. Finally, a pipeline for imputations is created and contained within the columns transformer. The code for this process is shown in Figure 24 to Figure 29.

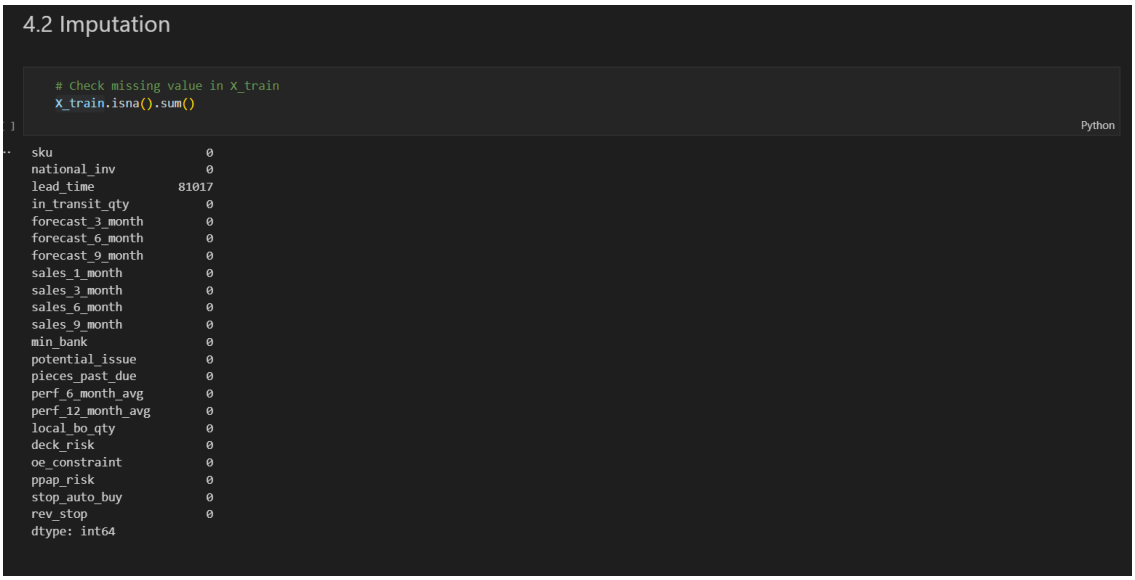


Figure 24: Checking missing values (NaN) in all features.

```python
# Check unique values of missing features
missing_features= ['lead_time','perf_6_month_avg','perf_12_month_avg']
for i in missing_features:
    print(i,X_train[i].unique(),'\n')
```

Python

```
lead_time [14.  8. 12.  2.  4. 52.  3.  9. 16. 11. nan 15.  0. 10.  5. 17.  6. 13.
 24. 20.  7. 22. 21. 30. 18. 26. 40.  1. 35. 28. 19. 25. 23.]

perf_6_month_avg [ 9.9e-01  8.0e-01  6.3e-01  4.3e-01  9.4e-01  1.0e+00  0.0e+00  9.8e-01
  9.5e-01 -9.9e+01  7.4e-01  9.6e-01  2.7e-01  8.5e-01  9.1e-01  7.8e-01
  6.6e-01  9.0e-01  5.0e-01  5.7e-01  8.6e-01  2.2e-01  9.7e-01  7.7e-01
  7.6e-01  6.9e-01  4.6e-01  9.3e-01  1.8e-01  3.1e-01  8.9e-01  8.8e-01
  8.3e-01  7.0e-01  7.9e-01  7.3e-01  4.8e-01  2.4e-01  8.2e-01  8.7e-01
  1.0e-01  8.4e-01  4.1e-01  6.8e-01  5.2e-01  3.7e-01  7.1e-01  9.2e-01
  5.1e-01  2.3e-01  6.5e-01  6.0e-01  5.6e-01  5.8e-01  5.4e-01  2.0e-02
  4.2e-01  4.4e-01  3.5e-01  2.1e-01  3.9e-01  4.0e-01  4.9e-01  6.1e-01
  6.2e-01  8.1e-01  2.8e-01  1.5e-01  3.6e-01  7.0e-02  7.5e-01  5.9e-01
  3.3e-01  6.4e-01  1.3e-01  9.0e-02  3.4e-01  6.7e-01  1.7e-01  3.8e-01
  2.6e-01  2.0e-01  4.0e-02  3.2e-01  5.3e-01  3.0e-02  5.5e-01  4.7e-01
  2.5e-01  8.0e-02  1.2e-01  1.4e-01  4.5e-01  1.6e-01  7.2e-01  1.9e-01
  1.0e-02  6.0e-02  5.0e-02  1.1e-01  3.0e-01  2.9e-01]

perf_12_month_avg [ 9.9e-01  8.2e-01  7.2e-01  1.9e-01  9.7e-01  8.9e-01  0.0e+00  9.8e-01
 -9.9e+01  8.1e-01  1.0e+00  9.3e-01  4.7e-01  7.0e-01  9.0e-01  7.4e-01
  6.6e-01  6.9e-01  9.5e-01  5.2e-01  7.6e-01  6.8e-01  8.8e-01  1.1e-01
  7.3e-01  7.8e-01  4.5e-01  9.4e-01  2.0e-01  4.6e-01  8.6e-01  8.0e-01
  7.9e-01  1.0e-02  8.3e-01  4.8e-01  9.6e-01  1.4e-01  8.4e-01  8.7e-01
  9.2e-01  8.5e-01  9.1e-01  1.3e-01  6.4e-01  5.1e-01  3.6e-01  7.5e-01
  6.1e-01  3.4e-01  7.1e-01  7.7e-01  3.5e-01  5.0e-01  5.8e-01  6.7e-01
  4.9e-01  1.0e-01  2.3e-01  5.4e-01  6.0e-02  3.3e-01  1.8e-01  5.9e-01
  4.1e-01  4.2e-01  6.2e-01  4.0e-01  5.7e-01  6.0e-01  2.8e-01  2.7e-01
  1.5e-01  6.3e-01  4.0e-02  1.2e-01  2.1e-01  3.0e-02  2.9e-01  1.7e-01
  2.5e-01  2.2e-01  9.0e-02  5.6e-01  5.3e-01  4.4e-01  5.0e-02  3.2e-01
  1.6e-01  3.7e-01  3.0e-01  5.5e-01  7.0e-02  8.0e-02  2.4e-01  3.9e-01
  3.8e-01  2.0e-02  6.5e-01  4.3e-01  2.6e-01  3.1e-01]
```

Figure 25: Checking the unique values to find the abnormal values.

Imputation:
'lead_time' = np.nan
'perf_6_month_avg' = -99.0
'perf_12_month_avg' = -99.0

```python
# There are any "-99.0" left in other features?
for feature in X_train.columns:
    unique_values = X_train[feature].unique()
    if -99.0 in unique_values:
        print(f"Feature {feature} contains -99.0 ")
```

Python

```
Feature national_inv contains -99.0
Feature perf_6_month_avg contains -99.0
Feature perf_12_month_avg contains -99.0
```

Imputation(updated):
'lead_time' = np.nan
'perf_6_month_avg' = -99.0
'perf_12_month_avg' = -99.0
'national_inv' = -99.0

Figure 26: Rechecking the '-99.00' values in other features.

```python
# updated missing features
missing_features.append('national_inv')
missing_features
```

Python

```
['lead_time', 'perf_6_month_avg', 'perf_12_month_avg', 'national_inv']
```

```python
# Create the imputers for nan and -99
imp_null = SimpleImputer(missing_values=np.nan, strategy= 'most_frequent')
imp_neg99 = SimpleImputer(missing_values= -99.0, strategy= 'most_frequent')
```

Python

```python
# fit_transform imputers for X_train and transform imputers for X_test
X_train['lead_time']= imp_null.fit_transform(X_train[['lead_time']])
X_test['lead_time']= imp_null.transform(X_test[['lead_time']])

X_train['national_inv'] = imp_neg99.fit_transform(X_train[['national_inv']])
X_test['national_inv']= imp_neg99.transform(X_test[['national_inv']])

X_train['perf_6_month_avg'] = imp_neg99.fit_transform(X_train[['perf_6_month_avg']])
X_test['perf_6_month_avg']= imp_neg99.transform(X_test[['perf_6_month_avg']])

X_train['perf_12_month_avg'] = imp_neg99.fit_transform(X_train[['perf_12_month_avg']])
X_test['perf_12_month_avg']= imp_neg99.transform(X_test[['perf_12_month_avg']])
```

Python

Figure 27: Create the SimpleImputer then fit or transform into datasets.

```python
# Check that imputer works
print(f"X_train lead_time: {X_train['lead_time'].isna().sum()}")
print(f"X_train national_inv: {len(X_train[X_train['national_inv'] == -99])}")
print(f"X_train perf_6_month_avg: {len(X_train[X_train['perf_6_month_avg'] == -99])}")
print(f"X_train perf_12_month_avg: {len(X_train[X_train['perf_12_month_avg'] == -99])}\n")

print(f"X_test lead_time: {X_train['lead_time'].isna().sum()}")
print(f"X_test national_inv: {len(X_train[X_train['national_inv'] == -99])}")
print(f"X_test perf_6_month_avg: {len(X_train[X_train['perf_6_month_avg'] == -99])}")
print(f"X_test perf_12_month_avg: {len(X_train[X_train['perf_12_month_avg'] == -99])}")
```

Python

```
X_train lead_time: 0
X_train national_inv: 0
X_train perf_6_month_avg: 0
X_train perf_12_month_avg: 0

X_test lead_time: 0
X_test national_inv: 0
X_test perf_6_month_avg: 0
X_test perf_12_month_avg: 0
```

Figure 28: Validate the imputation

```python
# Create Pipeline for imputation
impute = Pipeline(
    steps=[
        ('impute_null',imp_null),
        ('impute_(-99.0)',imp_neg99)
    ]
)
```

Python

```python
# Create ColumnTransformer for imputation
imputer = ColumnTransformer(
    transformers=[
        ("impute", impute, missing_features)
    ], remainder='passthrough', verbose_feature_names_out=False
).set_output(transform='pandas')
```

Python

Figure 29: Create the Pipeline for imputation.

- Standardization and Scaling:

  The MinMax scaler is the chosen scaling method. After its creation, it is applied to fit and transform only the numerical features. The scaling is then validated, and a pipeline for standardization and scaling is established. The code for this process is presented in Figure 30 to Figure Y.

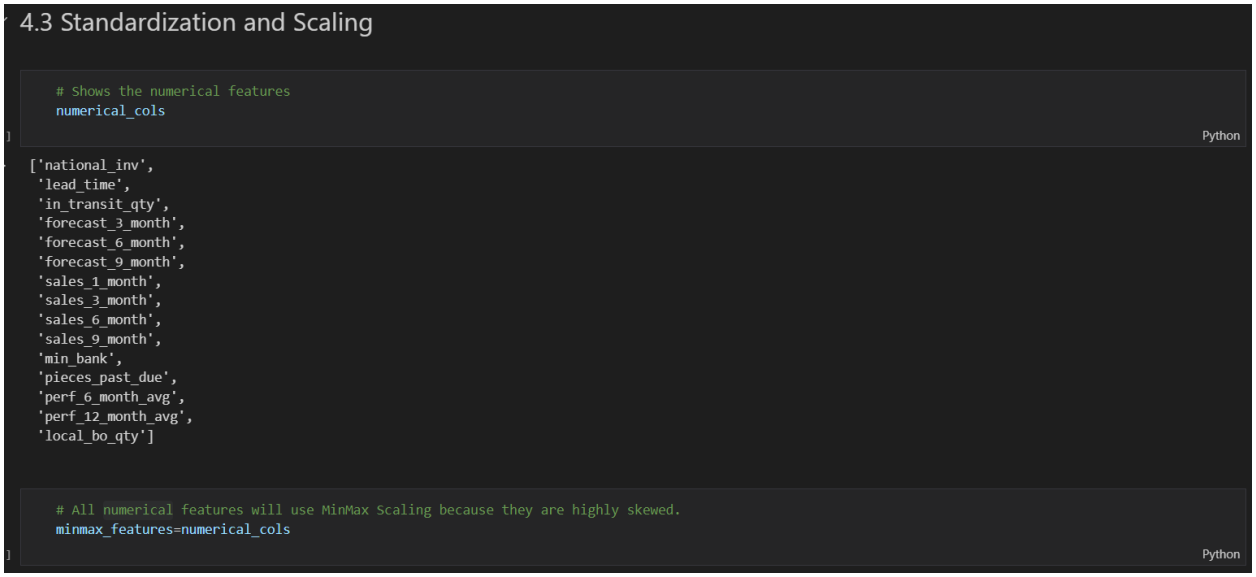
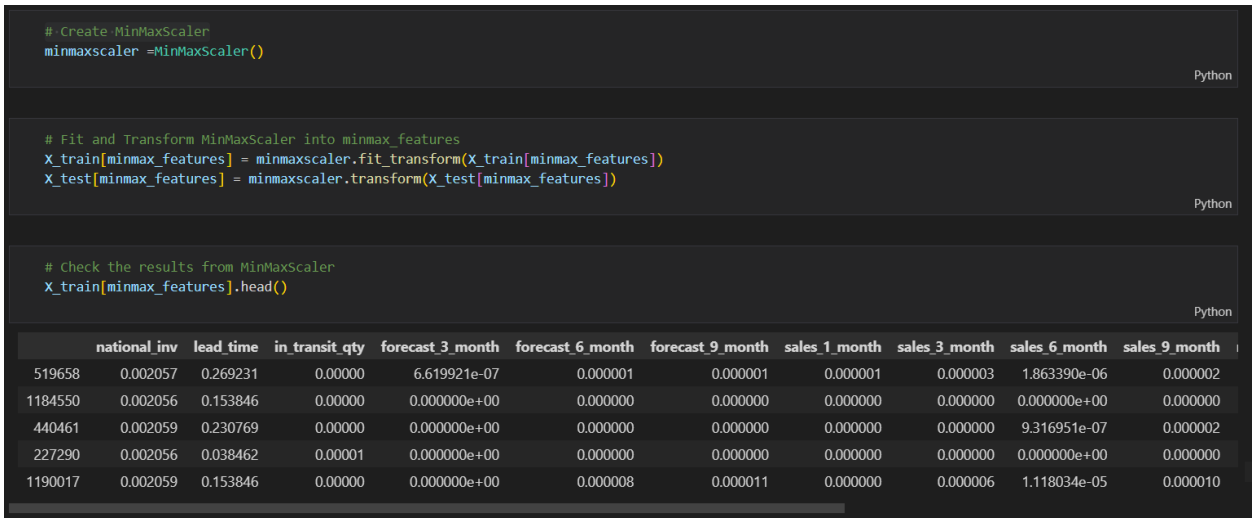


Figure 30: Checks the numerical values and stores them in the MinMax features variable.



	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_month	forecast_9_month	sales_1_month	sales_3_month	sales_6_month	sales_9_month
519658	0.002057	0.269231	0.00000	6.619921e-07	0.000001	0.000001	0.000001	0.000003	1.863390e-06	0.000002
1184550	0.002056	0.153846	0.00000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000
440461	0.002059	0.230769	0.00000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	9.316951e-07	0.000002
227290	0.002056	0.038462	0.00001	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000
1190017	0.002059	0.153846	0.00000	0.000000e+00	0.000008	0.000011	0.000000	0.000006	1.118034e-05	0.000010

Figure 31: Create, fit, and transform the MinMax scaling then validate the result.

```python
# Create ColumnTransformer for scaling
scaler = ColumnTransformer(
    transformers=[
        ('scaler_minmax',minmaxscaler,minmax_features)
    ]
    ,remainder='passthrough',verbose_feature_names_out=False
).set_output(transform='pandas')
```

Python

Figure 32: Create the Pipeline for MinMax scaling

- Encoding:

  Label encoding is employed for the categorical features to perform encoding. The process begins by selecting the features for encoding and creating the Label Encoder. Each categorical feature is iterated through and encoded. The Label Encoder is then validated. A pipeline for encoding is established using an ordinal encoder, which follows the same principles as a label encoder. The code for the encoding process is depicted in Figure 33 to Figure .

4.4 Encoding

```python
# Create the encode_cols for encoding
encode_cols = categorical_cols.copy()
encode_cols.remove('went_on_backorder')
print(encode_cols)
```

Python

```
['potential_issue', 'deck_risk', 'oe_constraint', 'ppap_risk', 'stop_auto_buy', 'rev_stop']
```

```python
# Create label encoder
labelencoder = LabelEncoder()
```

Python

```python
# Iterate through each categorical feature and encode it
for feature in encode_cols:
    X_train[feature] = labelencoder.fit_transform(X_train[feature])
    X_test[feature] = labelencoder.transform(X_test[feature])
```

Python

Figure 33: Perform the label encoding

```python
# Check the label encoding result
X_train.head()
```

Python

h	sales_6_month	...	potential_issue	pieces_past_due	perf_6_month_avg	perf_12_month_avg	local_bo_qty	deck_risk	oe_constraint	ppap_risk	stop_auto_buy	rev_stop
3	1.863390e-06	...	0	0.0	0.99	0.99	0.0	0	0	0	1	0
0	0.000000e+00	...	0	0.0	0.80	0.82	0.0	0	0	0	1	0
0	9.316951e-07	...	0	0.0	0.63	0.72	0.0	0	0	0	1	0
0	0.000000e+00	...	0	0.0	0.43	0.19	0.0	1	0	1	1	0
6	1.118034e-05	...	0	0.0	0.94	0.97	0.0	0	0	0	1	0

Figures 34: Validate encoding results

```python
# Create ColumnTransformer for encoding
# All category features are using LableEncoder
encoder = ColumnTransformer(
    transformers= [
        ('encoder_label',OrdinalEncoder(),encode_cols)
    ]
    ,remainder='passthrough'
)
```

Python

Figure 35: Create the Pipeline for encoding using Ordinal Encoder

- Pipepline and Preprocessing Result:

  In this section, all the pipelines for the preprocessing steps are combined with the RandomForest Classifier algorithm to make a sample prediction. Furthermore, the results of the preprocessing are validated. The code for this section is illustrated in Figure 36 and Figure 37.

```python
# Create Pipeline for all steps
pipeline = Pipeline([
    ('imputation',imputer),
    ('scaling',scaler),
    ('encoding',encoder),
    # Try with RandomForest Classifier algorithm
    ('prediction',RandomForestClassifier())
])
```

Python

Figure 36: Create the Pipeline for preprocessing and prediction.

Prepocessing Result

```python
X_train.head()
```

Python

	sku	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_month	forecast_9_month	sales_1_month	sales_3_month	sales_6_month	...	po
519658	1867616	0.002057	0.269231	0.00000	6.619921e-07	0.000001	0.000001	0.000001	0.000003	1.863390e-06	...	
1184550	1553554	0.002056	0.153846	0.00000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000e+00	...	
440461	1788443	0.002059	0.230769	0.00000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	9.316951e-07	...	
227290	1338851	0.002056	0.038462	0.00001	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000e+00	...	
1190017	1559412	0.002059	0.153846	0.00000	0.000000e+00	0.000008	0.000011	0.000000	0.000006	1.118034e-05	...	

5 rows × 22 columns

```python
X_test.head()
```

Python

h	sales_6_month	...	potential_issue	pieces_past_due	perf_6_month_avg	perf_12_month_avg	local_bo_qty	deck_risk	oe_constraint	ppap_risk	stop_auto_buy	rev_stop
0	0.000000e+00	...	0	0.0	1.00	0.82	0.0	0	0	0	1	0
4	6.955104e-04	...	0	0.0	0.87	0.90	0.0	0	0	0	1	0
0	0.000000e+00	...	0	0.0	0.68	0.66	0.0	0	0	1	1	0
7	4.658476e-07	...	0	0.0	0.59	0.61	0.0	0	0	0	1	0
0	4.658476e-07	...	0	0.0	0.79	0.79	0.0	0	0	0	1	0

Figure 37: Validation of X_train and X_test datasets

5. Prediction (Testing):

An example of prediction is conducted to test the usability of the pipeline, as illustrated in Figure 38. The RandomForest Classifier algorithm is chosen for testing due to its compatibility with classification problems. The pipelines appear to work well, except for the prediction performance, as shown in Figure 39. The model's accuracy is approximately 99%, indicating overfitting. This issue may stem from the imbalanced target features, as evidenced by the low F1 score. Possible solutions include resampling techniques, careful feature selection, and selecting an appropriate model architecture. Nonetheless, the preliminary results are still satisfactory.

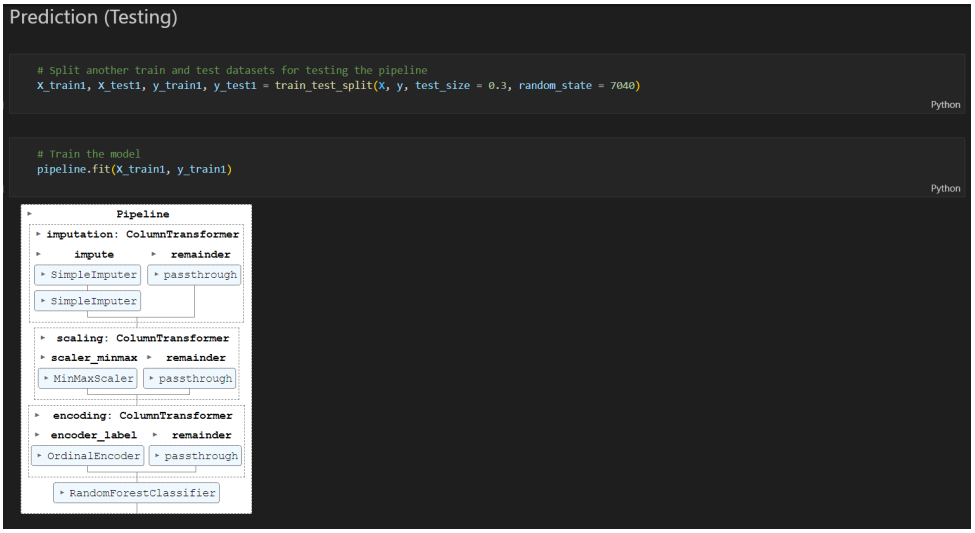


Figure 38: The pipeline and prediction testing

Figure 39: The accuracy and f1 scores of the prediction model

7. REFERENCES

[1] Santis, Rodrigo & Pestana de Aguiar, Eduardo & Goliatt, Leonardo. (2017). Predicting Material Backorders in Inventory Management using Machine Learning. https://doi.org/10.1109/LA-CCI.2017.8285684

[2] Islam, S., Amin, S.H. Prediction of probable backorder scenarios in the supply chain using Distributed Random Forest and Gradient Boosting Machine learning techniques. J Big Data 7, 65 (2020). https://doi.org/10.1186/s40537-020-00345-2

[3] Gao, H., Ren, Q., & Lv, C. (2022). Supply Chain Management and Backorder Products Prediction Utilizing Neural Network and Naive Bayes Machine Learning Techniques in Big Data Area: A Real-life Case Study. https://doi.org/10.21203/rs.3.rs-2020401/v1

[4] S. Zinjad, "Backorder Dataset," Kaggle, [Online]. Available: https://www.kaggle.com/datasets/ztrimus/backorder-dataset?select=Kaggle_Training_Dataset_v2.csv.