FINAL REPORT ON
RETURN PROPENSITY MODEL

By
Prithviraj
24PGAI0065
AI&DS

Jio Institute Mentor
Bharath Ram Reddy

Industry Mentor
Manoj Lakkireddy

JioInstitute

CERTIFICATE OF COMPLETION

JIO INSTITUTE INTERNSHIP REPORT

To acknowledge the successful completion of the internship training, fulfilling the requirements for the Postgraduate Programme

Name of the Student: **Pritrhviraj**

Roll No: 24PGAI0065

Academic Year: 2023-24

Name of the Programme: Artificial Intelligence & Data Science (AI & DS)

Name and Address of the Company: Jio Platforms Limited

Internship Period: From 13th December 2023 to 2nd February 2024

THIS IS TO CERTIFY THAT

Mr. **Prithviraj** has Satisfactorily Completed his Training/Project Work, submitted the training report and appeared for the Presentation & VIVA as required.

External Examiner Internal Examiner Dean

Date:
Place:

Internship Completion Letter

06. February.2024

<u>To Whomsoever It May Concern</u>

This is to certify that Mr. Prithviraj from "**Jio Institute"** has completed his internship at **Jio Platforms Ltd** from **13th December' 2022 till 02 February' 2024**.

During this period, Mr. Prithviraj completed a project titled **"Return propensity modelling"** under the guidance of **Mr. Manoj Reddy.**

Please note, as the project involves highly sensitive data, all the details and the contents made towards the project needs to be kept confidential. Under no circumstances must this be shared nor distributed in any form to anyone other than the company authorized personnel or law requires as. It is expected that Mr. Prithviraj and adheres to the company policy and its rules.

During the internship period, we found Mr. Prithviraj to be sincere and diligent.

We wish him all the best in all future endeavours.

Yours sincerely,
Jio Platforms Limited

S Dasgupta

Swagata Dasgupta
HR Business Partner – Jio Platforms Limited
Swagata.dasgupta@ril.com

TABLE OF CONTENTS

ACKNOWLEDGMENT

We express our deepest gratitude to all those who have contributed to the successful completion of this internship project, "Predicting Item Cancellation in Retail using Machine Learning." This endeavour has been a collaborative effort, and we are thankful for the support, guidance, and expertise provided by various individuals and entities.

First and foremost, we extend our appreciation to our mentor **Mr. Manoj Kumar Lakkireddy**, whose invaluable insights, encouragement, and constructive feedback have been instrumental in shaping the direction and execution of this project. His mentorship has been a source of inspiration and a guiding light throughout this learning journey.

We would like to thank our mentor for suggesting this most suitable Kaggle dataset and creating an environment conducive to learning and exploration. The availability of real-world data has enriched our understanding of the challenges and intricacies involved in the retail domain.

Special thanks to my colleagues and peers Mr. Santanu and Somnath who have contributed to discussions, brainstorming sessions, and collaborative problem-solving. Their diverse perspectives and shared knowledge have fostered a dynamic and enriching work environment. Lastly, we express gratitude to our families and friends for their unwavering support, understanding, and encouragement during the course of this internship. Your belief in our abilities has been a constant source of motivation.

This project has been a collective effort, and we extend our sincere thanks to everyone who has played a role, directly or indirectly, in its successful completion.

ABSTRACT

This project addresses the imperative need for predicting and mitigating item cancellations in the dynamic retail landscape. Leveraging a comprehensive one and half year dataset from Kaggle, sourced from a fashion distributor, the endeavour employs machine learning algorithms - decision tree, random forest, and XGBoost - to construct a predictive model for estimating return rates.

The dataset encompasses vital transaction details, including orderID, orderDate, articleID, colorCode, sizeCode, productGroup, perUnitPrice, orderPrice, and returnPrice. Through meticulous data pre-processing, exploratory data analysis, and visualization techniques, the project unveils patterns and correlations, providing valuable insights into customer interactions.

Feature engineering at customer, order, and article levels enriches the dataset with meaningful attributes, contributing to a nuanced understanding of customer behavior. The project's culmination involves hyperparameter tuning to optimize model performance, ensuring practical efficacy in real-world retail scenarios.

This is not just an academic exercise but a practical endeavour aimed at delivering tangible solutions for the retail sector. The outcome is a predictive model capable of accurate estimations of item cancellations, serving as a strategic tool for decision-makers navigating the challenges of the retail landscape. The insights gained have transformative potential, promising increased operational efficiency and heightened customer satisfaction in the face of evolving market dynamics.

INTRODUCTION

In the dynamic landscape of the retail industry, predicting and understanding the factors contributing to item cancellations is of paramount importance for enhancing operational efficiency and customer satisfaction. This internship project delves into the realm of predictive analytics, leveraging a comprehensive dataset sourced from Kaggle, provided by a fashion distributor over a two-year period. The dataset encapsulates intricate details about customer orders, including orderID, orderDate, articleID, colorCode, sizeCode, productGroup, and various financial aspects such as perUnitPrice, orderPrice, and returnPrice.

The primary objective of this project is to construct a robust machine learning classification model capable of predicting whether an item will be cancelled or not. To achieve this, three prominent algorithms have been employed: Decision Tree, Random Forest, and XGBoost. These algorithms are well-suited for classification tasks and are expected to provide valuable insights into the patterns and features influencing cancellation rates in the retail domain.

The project unfolds through the following key phases:

Data Pre-processing: This initial step involves meticulous data cleaning, addressing missing values, outlier detection, and correcting errors to ensure the dataset's reliability. Data transformation is also executed to convert relevant columns into appropriate formats, enhancing their compatibility for analysis.

Exploratory Data Analysis and Visualization: Visualization techniques such as histograms, scatter plots, and correlation analyses are employed to unravel patterns, distributions, and relationships within the dataset. Understanding central tendencies and identifying correlations lay the groundwork for subsequent analyses.

Feature Engineering: Feature engineering is implemented at three levels – customer, order, and article. This involves creating insightful features such as Recency, Frequency, and Monetary (RFM) metrics at the customer level, unique article counts at the order level, and considering combinations of articleID, colorCode, and sizeCode at the article level.

Model Building and Hyperparameter Tuning: Decision Tree, Random Forest, and XGBoost models are applied for classification, and hyperparameter tuning is executed to optimize their performance. This involves systematically adjusting parameters to enhance the models' predictive capabilities.

<u>About Dataset:</u>

The dataset chosen for this internship project originates from Kaggle and is centered around a fashion distributor's operations over a span of one and half year. The distributor specializes in selling articles of various sizes and colors to its clientele. The dataset comprises essential information pertaining to customer orders, including orderID, orderDate, articleID, colorCode, sizeCode, productGroup, and financial details such as perUnitPrice, orderPrice, and returnPrice. Notably, the dataset also captures instances of item returns, providing a comprehensive view of customer interactions and potential challenges in the retail process.

This dataset, with its rich and multidimensional information, serves as the cornerstone for insightful analyses, facilitating data-driven decision-making processes. Through meticulous exploration and application of machine learning, this project aspires to unlock valuable insights that can be translated into actionable strategies for optimizing the retail experience and mitigating the impact of item cancellations.

Link for data set: Click here

METHODOLOGY

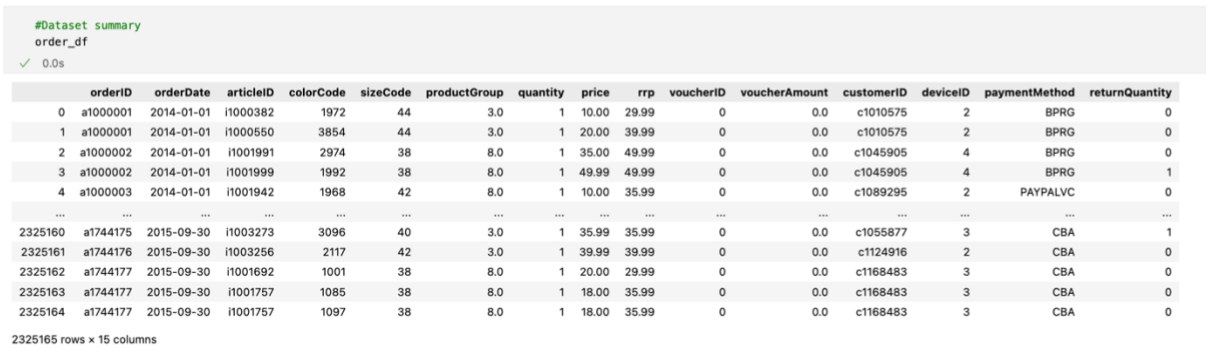
The project involves following key steps:

1. Data pre-processing
2. Exploratory Data Analysis and Visualization
3. Feature Engineering
4. Model Building and validation

Data pre-processing:

Data Cleaning: This involves handling missing values, outliers, and errors in the dataset. Imputing missing values, removing outliers, and correcting errors ensure that the data is reliable for analysis.

Original Data looks like below:

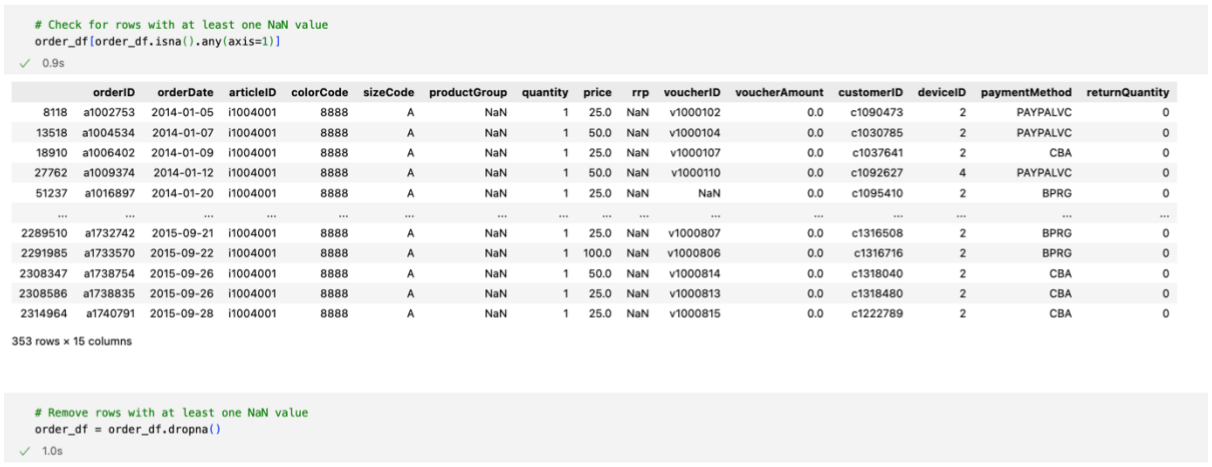


```
#Dataset summary
order_df
✓ 0.0s
```

	orderID	orderDate	articleID	colorCode	sizeCode	productGroup	quantity	price	rrp	voucherID	voucherAmount	customerID	deviceID	paymentMethod	returnQuantity
0	a1000001	2014-01-01	i1000382	1972	44	3.0	1	10.00	29.99	0	0.0	c1010575	2	BPRG	0
1	a1000001	2014-01-01	i1000550	3854	44	3.0	1	20.00	39.99	0	0.0	c1010575	2	BPRG	0
2	a1000002	2014-01-01	i1001991	2974	38	8.0	1	35.00	49.99	0	0.0	c1045905	4	BPRG	0
3	a1000002	2014-01-01	i1001999	1992	38	8.0	1	49.99	49.99	0	0.0	c1045905	4	BPRG	1
4	a1000003	2014-01-01	i1001942	1968	42	8.0	1	10.00	35.99	0	0.0	c1089295	2	PAYPALVC	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2325160	a1744175	2015-09-30	i1003273	3096	40	3.0	1	35.99	35.99	0	0.0	c1055877	3	CBA	1
2325161	a1744176	2015-09-30	i1003256	2117	42	3.0	1	39.99	39.99	0	0.0	c1124916	2	CBA	0
2325162	a1744177	2015-09-30	i1001692	1001	38	8.0	1	20.00	29.99	0	0.0	c1168483	3	CBA	0
2325163	a1744177	2015-09-30	i1001757	1085	38	8.0	1	18.00	35.99	0	0.0	c1168483	3	CBA	0
2325164	a1744177	2015-09-30	i1001757	1097	38	8.0	1	18.00	35.99	0	0.0	c1168483	3	CBA	0

2325165 rows × 15 columns

We check the NaN values across the dataset and then remove these from it. Code snippet is as follows:



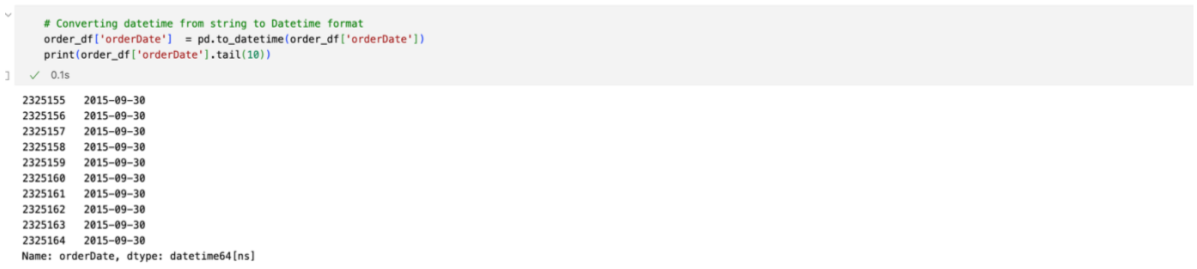
```python
# Check for rows with at least one NaN value
order_df[order_df.isna().any(axis=1)]
```
✓ 0.9s

	orderID	orderDate	articleID	colorCode	sizeCode	productGroup	quantity	price	rrp	voucherID	voucherAmount	customerID	deviceID	paymentMethod	returnQuantity
8118	a1002753	2014-01-05	i1004001	8888	A	NaN	1	25.0	NaN	v1000102	0.0	c1090473	2	PAYPALVC	0
13518	a1004534	2014-01-07	i1004001	8888	A	NaN	1	50.0	NaN	v1000104	0.0	c1030785	2	PAYPALVC	0
18910	a1006402	2014-01-09	i1004001	8888	A	NaN	1	25.0	NaN	v1000107	0.0	c1037641	2	CBA	0
27762	a1009374	2014-01-12	i1004001	8888	A	NaN	1	50.0	NaN	v1000110	0.0	c1092627	4	PAYPALVC	0
51237	a1016897	2014-01-20	i1004001	8888	A	NaN	1	25.0	NaN	NaN	0.0	c1095410	2	BPRG	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2289510	a1732742	2015-09-21	i1004001	8888	A	NaN	1	25.0	NaN	v1000807	0.0	c1316508	2	BPRG	0
2291985	a1733570	2015-09-22	i1004001	8888	A	NaN	1	100.0	NaN	v1000806	0.0	c1316716	2	BPRG	0
2308347	a1738754	2015-09-26	i1004001	8888	A	NaN	1	50.0	NaN	v1000814	0.0	c1318040	2	CBA	0
2308586	a1738835	2015-09-26	i1004001	8888	A	NaN	1	25.0	NaN	v1000813	0.0	c1318480	2	CBA	0
2314964	a1740791	2015-09-28	i1004001	8888	A	NaN	1	25.0	NaN	v1000815	0.0	c1222789	2	CBA	0

353 rows × 15 columns

```python
# Remove rows with at least one NaN value
order_df = order_df.dropna()
```
✓ 1.0s

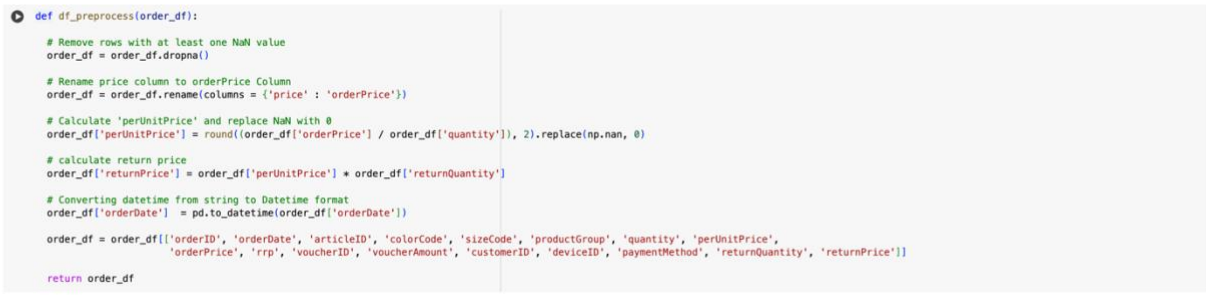
Data Transformation: This step includes converting data into a suitable format for analysis. It includes the changing datatype into required one format like changing order date datatype from object to datetime format.



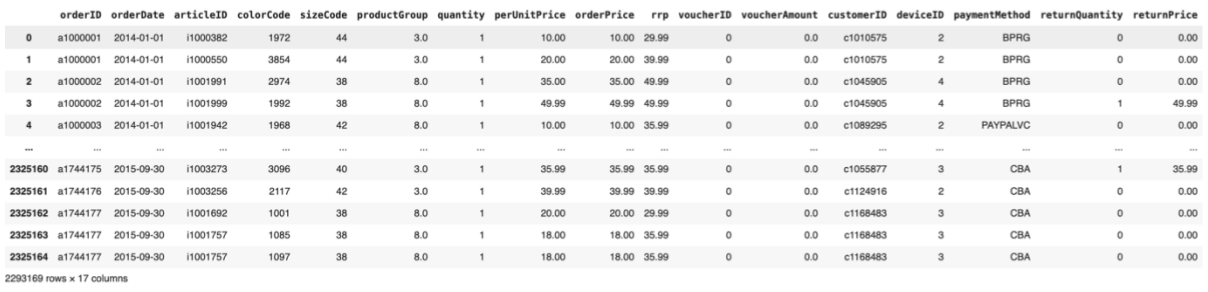
```python
# Converting datetime from string to Datetime format
order_df['orderDate']  = pd.to_datetime(order_df['orderDate'])
print(order_df['orderDate'].tail(10))
```
✓ 0.1s

```
2325155   2015-09-30
2325156   2015-09-30
2325157   2015-09-30
2325158   2015-09-30
2325159   2015-09-30
2325160   2015-09-30
2325161   2015-09-30
2325162   2015-09-30
2325163   2015-09-30
2325164   2015-09-30
Name: orderDate, dtype: datetime64[ns]
```

Adding required columns for feature engineering: Added two new columns (per unit price & return price) into main data frame which is helpful in feature engineering.



```python
def df_preprocess(order_df):

    # Remove rows with at least one NaN value
    order_df = order_df.dropna()

    # Rename price column to orderPrice Column
    order_df = order_df.rename(columns = {'price' : 'orderPrice'})

    # Calculate 'perUnitPrice' and replace NaN with 0
    order_df['perUnitPrice'] = round((order_df['orderPrice'] / order_df['quantity']), 2).replace(np.nan, 0)

    # calculate return price
    order_df['returnPrice'] = order_df['perUnitPrice'] * order_df['returnQuantity']

    # Converting datetime from string to Datetime format
    order_df['orderDate']  = pd.to_datetime(order_df['orderDate'])

    order_df = order_df[['orderID', 'orderDate', 'articleID', 'colorCode', 'sizeCode', 'productGroup', 'quantity', 'perUnitPrice',
                         'orderPrice', 'rrp', 'voucherID', 'voucherAmount', 'customerID', 'deviceID', 'paymentMethod', 'returnQuantity', 'returnPrice']]

    return order_df
```



	orderID	orderDate	articleID	colorCode	sizeCode	productGroup	quantity	perUnitPrice	orderPrice	rrp	voucherID	voucherAmount	customerID	deviceID	paymentMethod	returnQuantity	returnPrice
0	a1000001	2014-01-01	i1000382	1972	44	3.0	1	10.00	10.00	29.99	0	0.0	c1010575	2	BPRG	0	0.00
1	a1000001	2014-01-01	i1000550	3854	44	3.0	1	20.00	20.00	39.99	0	0.0	c1010575	2	BPRG	0	0.00
2	a1000002	2014-01-01	i1001991	2974	38	8.0	1	35.00	35.00	49.99	0	0.0	c1045905	4	BPRG	0	0.00
3	a1000002	2014-01-01	i1001999	1992	38	8.0	1	49.99	49.99	49.99	0	0.0	c1045905	4	BPRG	1	49.99
4	a1000003	2014-01-01	i1001942	1968	42	8.0	1	10.00	10.00	35.99	0	0.0	c1089295	2	PAYPALVC	0	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2325160	a1744175	2015-09-30	i1003273	3096	40	3.0	1	35.99	35.99	35.99	0	0.0	c1055877	3	CBA	1	35.99
2325161	a1744176	2015-09-30	i1003256	2117	42	3.0	1	39.99	39.99	39.99	0	0.0	c1124916	2	CBA	0	0.00
2325162	a1744177	2015-09-30	i1001692	1001	38	8.0	1	20.00	20.00	29.99	0	0.0	c1168483	3	CBA	0	0.00
2325163	a1744177	2015-09-30	i1001757	1085	38	8.0	1	18.00	18.00	35.99	0	0.0	c1168483	3	CBA	0	0.00
2325164	a1744177	2015-09-30	i1001757	1097	38	8.0	1	18.00	18.00	35.99	0	0.0	c1168483	3	CBA	0	0.00

2293169 rows × 17 columns

Taking Data Insights:

- Check return percentage
- Check unique customer
- Check order placed vs return against each customer
- Check the impact of voucher on return
- Check if price > rrp (retail recommended price)
- Check relation between payment type and return

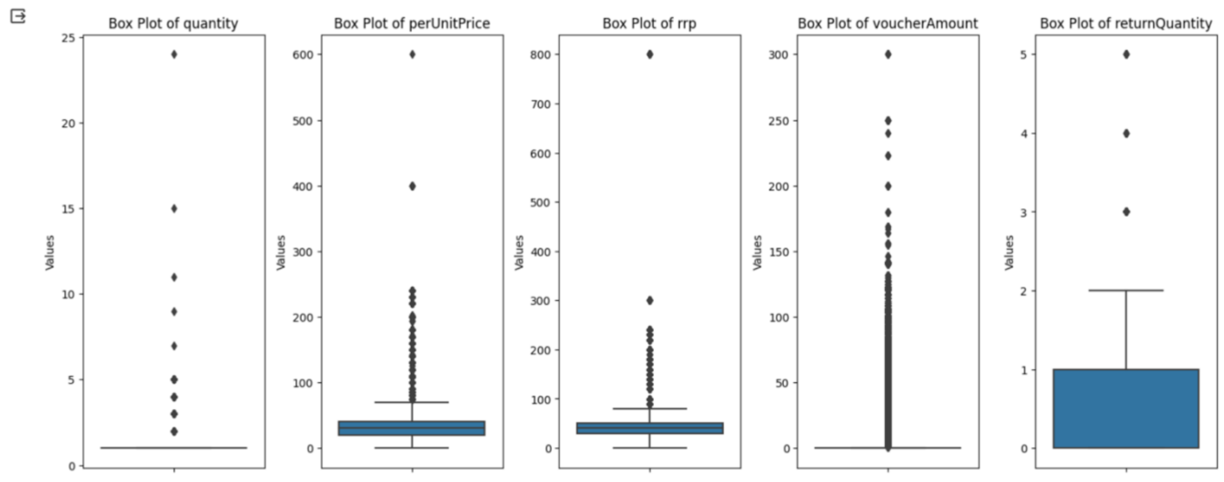
After the complete Data pre-processing, we did data visualization to check the pattern and outliers inside it.

Exploratory Data Analysis and Visualization:

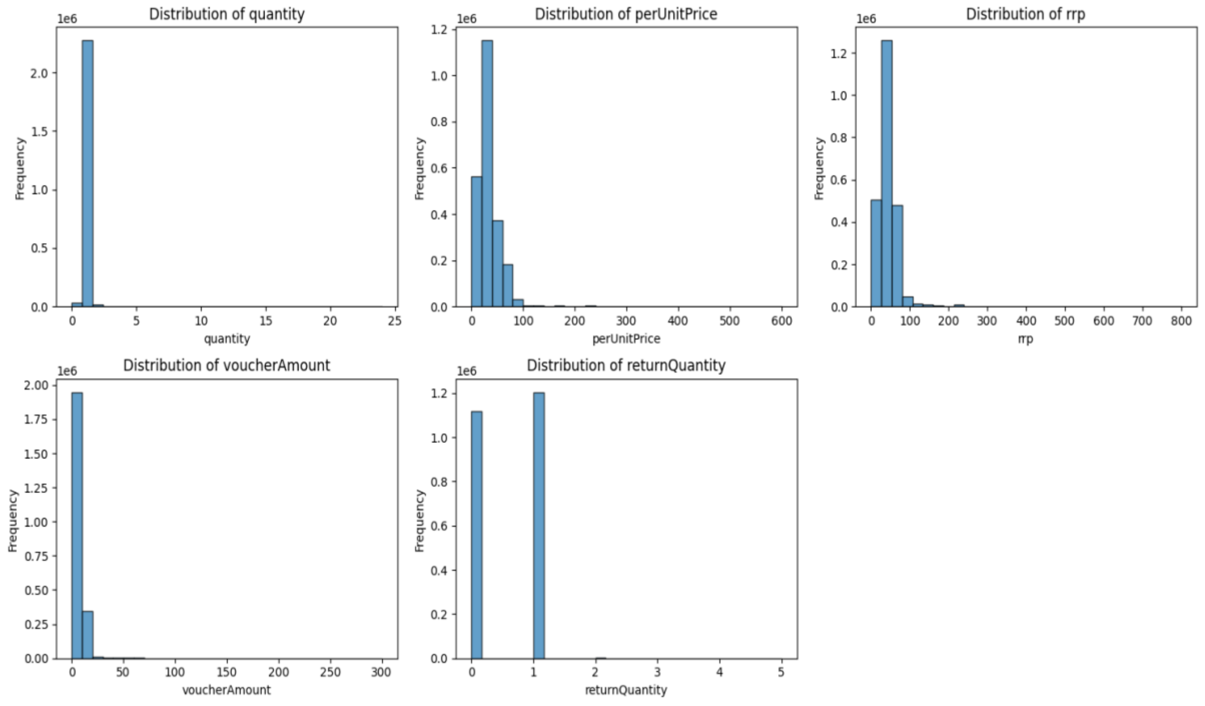
Exploratory Data Analysis (EDA): Visualizing the data to gain insights into its distribution, patterns, and relationships. This involves creating histograms, scatter plots, box plots, etc., to understand the central tendency, spread, and correlation between variables.

We analyses box plots and histograms for the required features like ‘qunatity’, ‘per unit price’, ‘rrp’, ‘voucherAmount’ , ‘returnQuantity’ etc

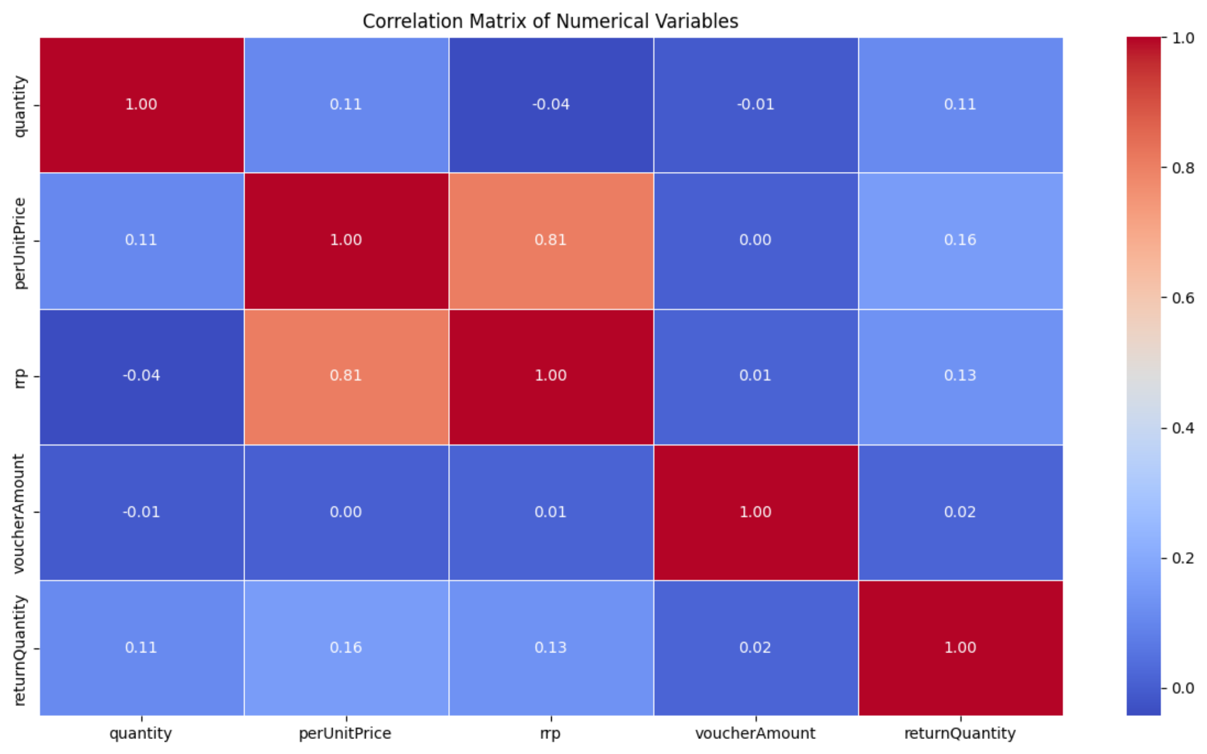
Box Plots



Histograms:



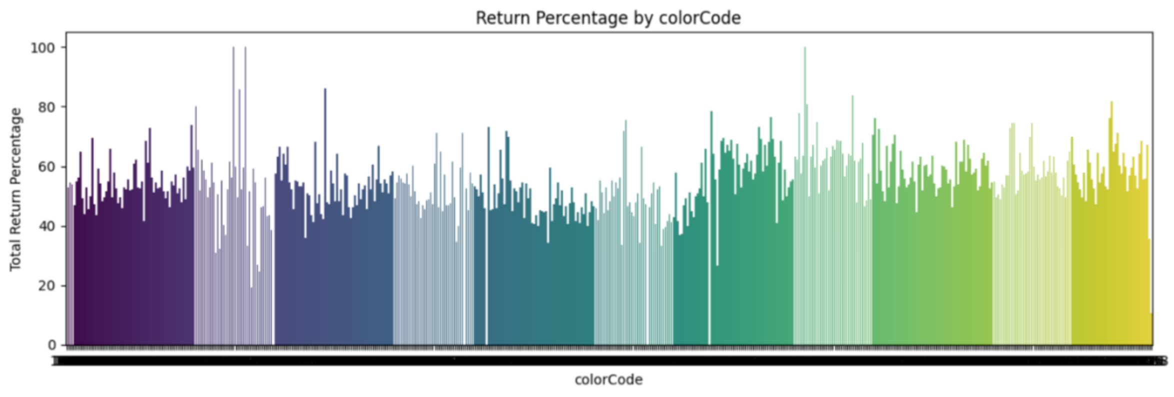
Correlation Analysis: Determining the strength and direction of relationships between variables. In this project, we did correlation matrix analysis about the variables like quantity , price, voucher and returnQuantity.



Return percentage vs Categorical variables :

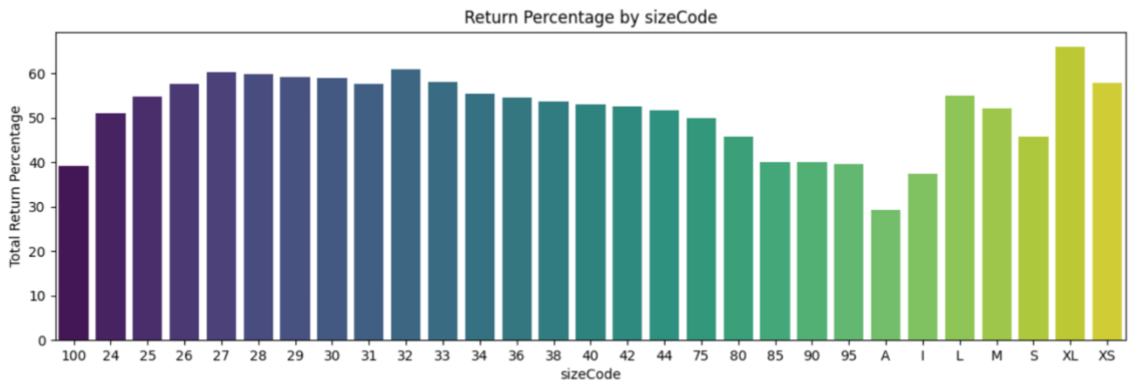
Return Percentage vs Color Code:

Analyzing the impact of color on return percentage reveals intriguing patterns. Certain color codes may exhibit higher or lower return rates, providing insights into color preferences or potential issues related to specific hues.



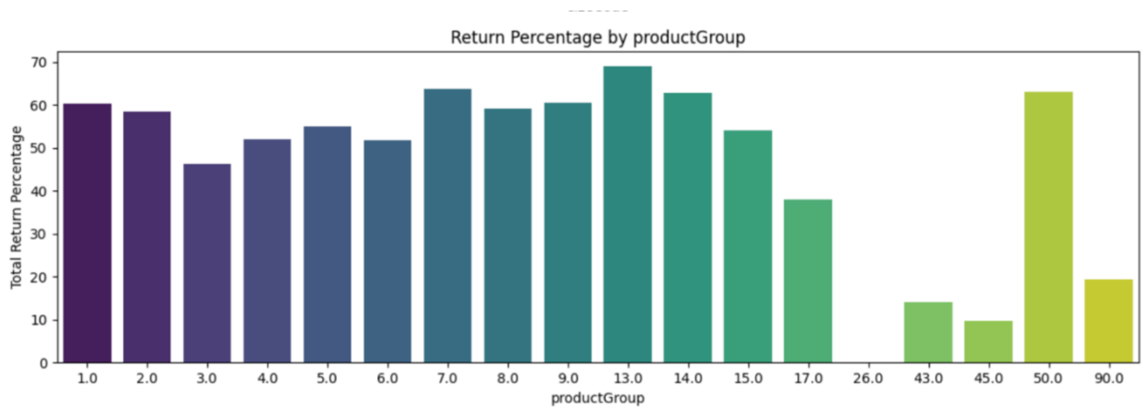
Return Percentage vs Size Code:

Examining return percentages concerning size codes unveils valuable information about customer size preferences. Variances in return rates across size categories can inform inventory management strategies and product offerings tailored to customer sizes.



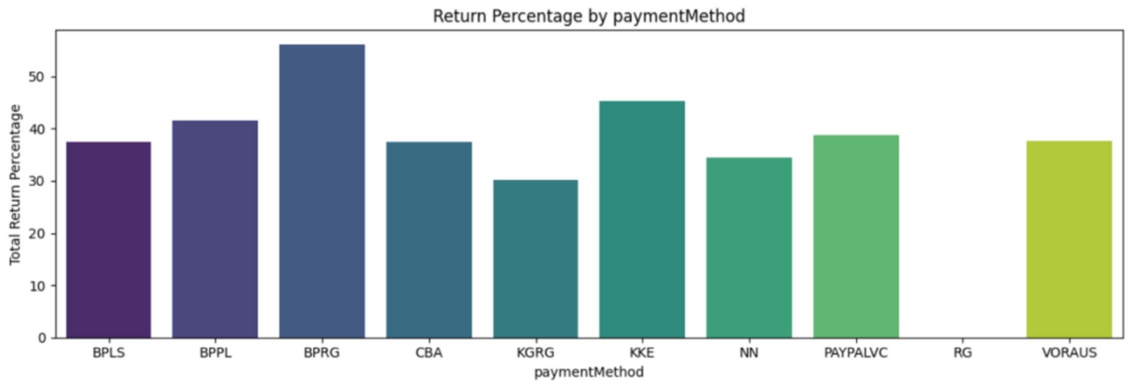
Return Percentage vs Product Group:

The correlation between return percentage and product groups sheds light on the performance of different product categories. Identifying which product groups have higher return percentages aids in refining marketing strategies and optimizing inventory for improved customer satisfaction.



Return Percentage vs Payment Method:

Understanding the relationship between return percentage and payment methods provides insights into customer behavior during the purchasing process. Variations in return rates based on payment methods could influence payment options offered or guide targeted interventions to reduce returns.

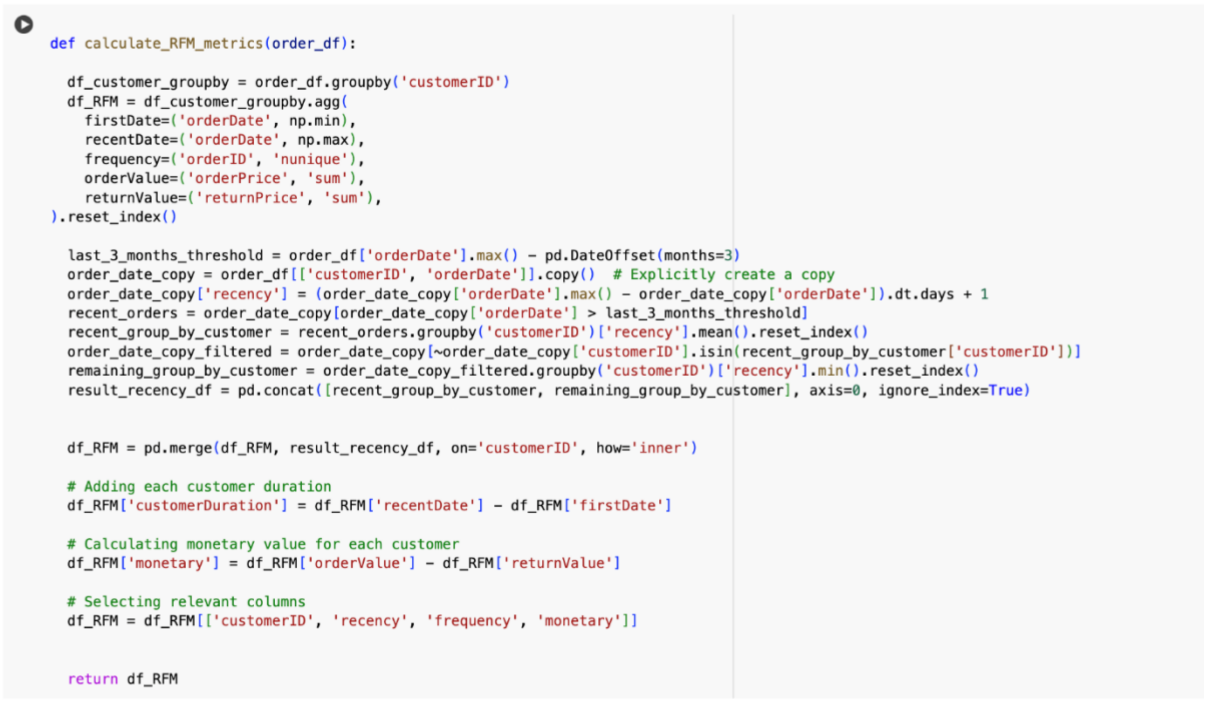


Feature Engineering:

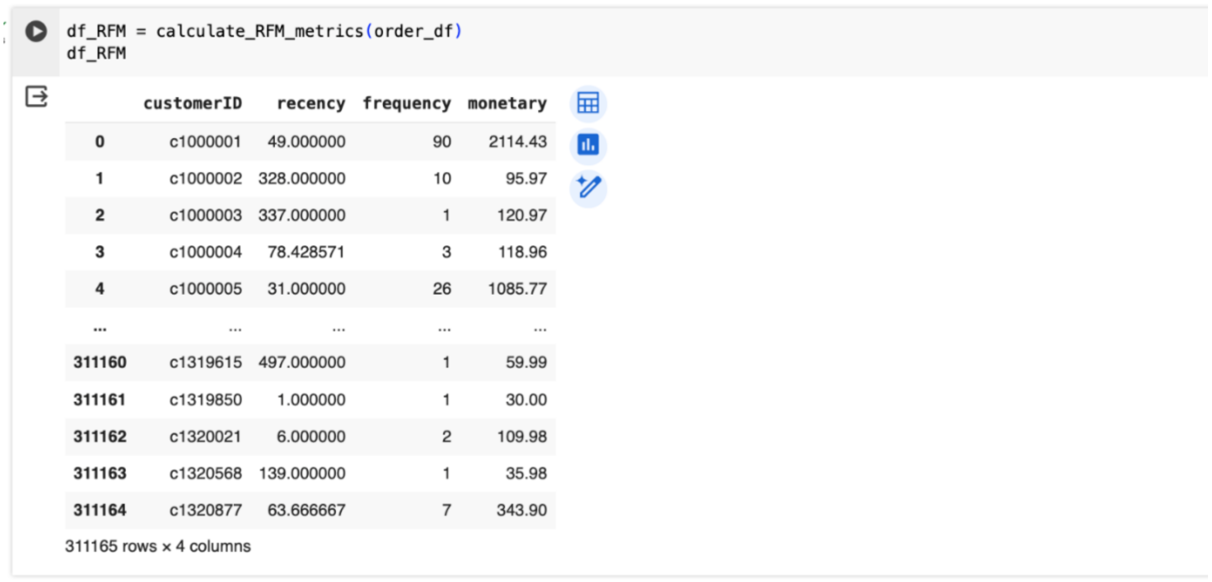
Feature engineering involves transforming raw data into meaningful features, enriching the dataset for enhanced predictive power. At the customer level, features like Recency, Frequency, and Monetary (RFM) metrics unravel behaviour insights. Order-level features encapsulate unique counts, offering granular transaction perspectives. Article-level features tackle the complexity of multiple codes, unifying them for comprehensive analysis. This transformative process not only optimizes machine learning model performance but also unveils nuanced insights into customer behaviour and transaction dynamics.

Making new data frames by grouping the data on different parameters like customerID, productGroup, paymentMethod and finally generate **RFM** for whole data according to unique customer.

The pinnacle of this feature engineering journey involves creating a comprehensive RFM (Recency, Frequency, Monetary) framework for the entire dataset, providing a holistic view of unique customer interactions and establishing a robust foundation for nuanced analysis and predictive modelling.



```
def calculate_RFM_metrics(order_df):

  df_customer_groupby = order_df.groupby('customerID')
  df_RFM = df_customer_groupby.agg(
    firstDate=('orderDate', np.min),
    recentDate=('orderDate', np.max),
    frequency=('orderID', 'nunique'),
    orderValue=('orderPrice', 'sum'),
    returnValue=('returnPrice', 'sum'),
).reset_index()

  last_3_months_threshold = order_df['orderDate'].max() - pd.DateOffset(months=3)
  order_date_copy = order_df[['customerID', 'orderDate']].copy()  # Explicitly create a copy
  order_date_copy['recency'] = (order_date_copy['orderDate'].max() - order_date_copy['orderDate']).dt.days + 1
  recent_orders = order_date_copy[order_date_copy['orderDate'] > last_3_months_threshold]
  recent_group_by_customer = recent_orders.groupby('customerID')['recency'].mean().reset_index()
  order_date_copy_filtered = order_date_copy[~order_date_copy['customerID'].isin(recent_group_by_customer['customerID'])]
  remaining_group_by_customer = order_date_copy_filtered.groupby('customerID')['recency'].min().reset_index()
  result_recency_df = pd.concat([recent_group_by_customer, remaining_group_by_customer], axis=0, ignore_index=True)


  df_RFM = pd.merge(df_RFM, result_recency_df, on='customerID', how='inner')

  # Adding each customer duration
  df_RFM['customerDuration'] = df_RFM['recentDate'] - df_RFM['firstDate']

  # Calculating monetary value for each customer
  df_RFM['monetary'] = df_RFM['orderValue'] - df_RFM['returnValue']

  # Selecting relevant columns
  df_RFM = df_RFM[['customerID', 'recency', 'frequency', 'monetary']]


  return df_RFM
```



```
df_RFM = calculate_RFM_metrics(order_df)
df_RFM
```

```
        customerID     recency  frequency  monetary
0         c1000001   49.000000         90   2114.43
1         c1000002  328.000000         10     95.97
2         c1000003  337.000000          1    120.97
3         c1000004   78.428571          3    118.96
4         c1000005   31.000000         26   1085.77
...            ...         ...        ...       ...
311160    c1319615  497.000000          1     59.99
311161    c1319850    1.000000          1     30.00
311162    c1320021    6.000000          2    109.98
311163    c1320568  139.000000          1     35.98
311164    c1320877   63.666667          7    343.90
311165 rows × 4 columns
```

After that we finalized to make the feature engineering on three levels:

- Customer level feature engineering
- Order level feature engineering
- Article level feature engineering

Customer level feature engineering:

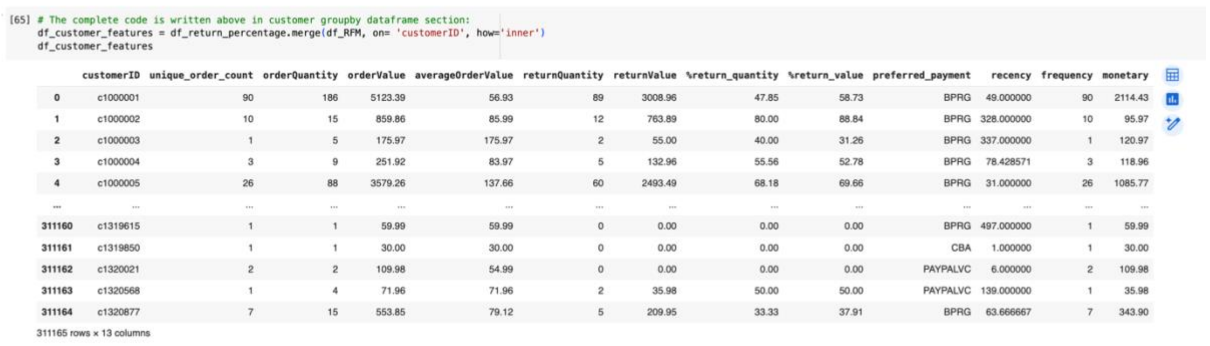
In this, we create several features as below to understand the customer behaviour. We added RFM in this as well which is the main feature for customer behaviour understanding.

In the realm of customer-level feature engineering, a series of insightful features have been meticulously crafted to delve into and comprehend customer behavior. These include metrics such as Recency, reflecting the time since the last customer transaction; Frequency, indicating how often a customer engages with transactions;

and Monetary, encapsulating the total monetary value of a customer's transactions. These RFM features serve as the cornerstone for understanding and predicting customer behavior, providing a comprehensive framework to decipher patterns and preferences within the retail dataset.

Recency: We take here weighted recency where recency is the average of order purchase interval of last three months

Customer_Level_Features – code snippet and it’s output.



```
[65] # The complete code is written above in customer groupby dataframe section:
     df_customer_features = df_return_percentage.merge(df_RFM, on= 'customerID', how='inner')
     df_customer_features
```

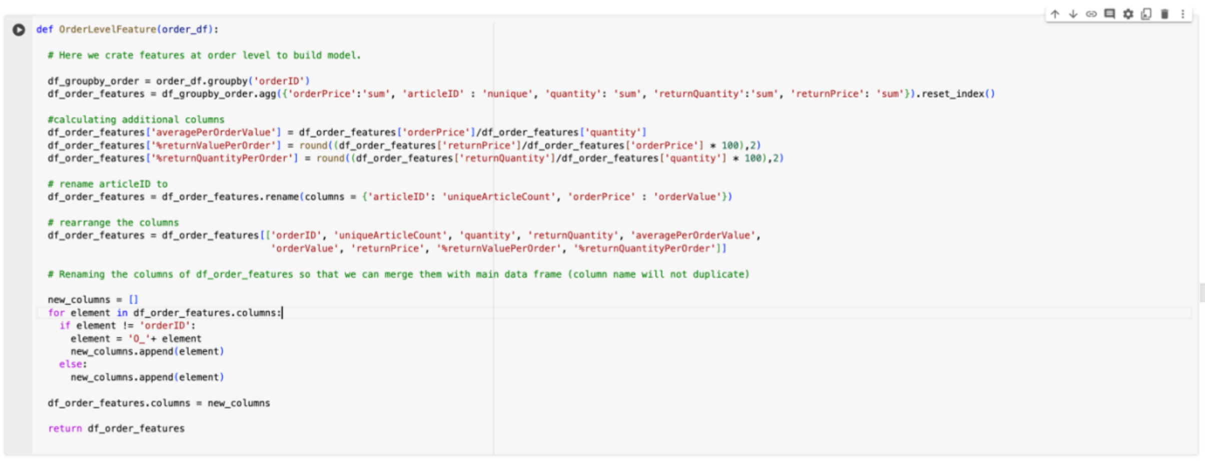
	customerID	unique_order_count	orderQuantity	orderValue	averageOrderValue	returnQuantity	returnValue	%return_quantity	%return_value	preferred_payment	recency	frequency	monetary
0	c1000001	90	186	5123.39	56.93	89	3008.96	47.85	58.73	BPRG	49.000000	90	2114.43
1	c1000002	10	15	859.86	85.99	12	763.89	80.00	88.84	BPRG	328.000000	10	95.97
2	c1000003	1	5	175.97	175.97	2	55.00	40.00	31.26	BPRG	337.000000	1	120.97
3	c1000004	3	9	251.92	83.97	5	132.96	55.56	52.78	BPRG	78.428571	3	118.96
4	c1000005	26	88	3579.26	137.66	60	2493.49	68.18	69.66	BPRG	31.000000	26	1085.77
...	...	...	...	...	...	...	...	...	...	...	...	...	...
311160	c1319615	1	1	59.99	59.99	0	0.00	0.00	0.00	BPRG	497.000000	1	59.99
311161	c1319850	1	1	30.00	30.00	0	0.00	0.00	0.00	CBA	1.000000	1	30.00
311162	c1320021	2	2	109.98	54.99	0	0.00	0.00	0.00	PAYPALVC	6.000000	2	109.98
311163	c1320568	1	4	71.96	71.96	2	35.98	50.00	50.00	PAYPALVC	139.000000	1	35.98
311164	c1320877	7	15	553.85	79.12	5	209.95	33.33	37.91	BPRG	63.666667	7	343.90

311165 rows × 13 columns

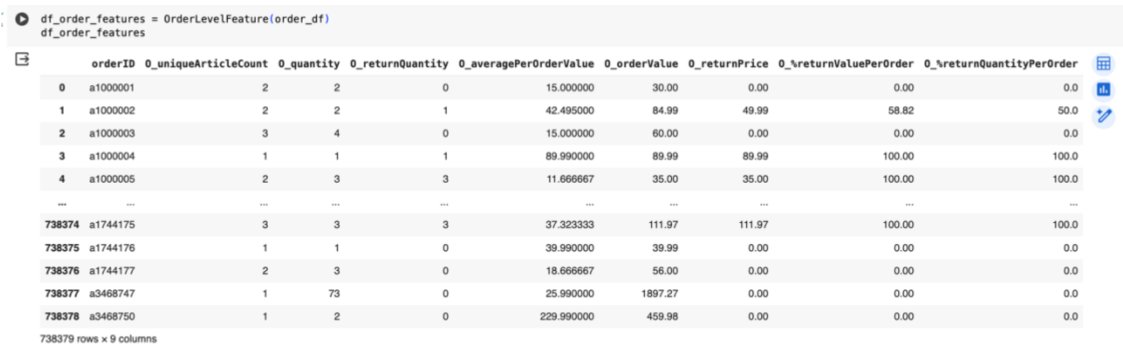
Order level feature Engineering:

Within the realm of order-level feature engineering, a systematic grouping of the entire dataset by unique orderID has been implemented. This process facilitates the creation of essential features at the order level, offering valuable insights into transaction characteristics. Notable features include the count of unique articles within each order, the total order count, and the count of returns associated with individual orders. These features collectively contribute to a nuanced understanding of transaction patterns at the order level, enhancing the project's capacity to discern and predict retail dynamics effectively. The accompanying code snippet and resulting output offer a clear and detailed illustration of these crucial order-level features, further enriching the analytical depth of the internship project.

Order_Level_Features code snippet and it's output



```python
def OrderLevelFeature(order_df):

  # Here we crate features at order level to build model.

  df_groupby_order = order_df.groupby('orderID')
  df_order_features = df_groupby_order.agg({'orderPrice':'sum', 'articleID' : 'nunique', 'quantity': 'sum', 'returnQuantity':'sum', 'returnPrice': 'sum'}).reset_index()

  #calculating additional columns
  df_order_features['averagePerOrderValue'] = df_order_features['orderPrice']/df_order_features['quantity']
  df_order_features['%returnValuePerOrder'] = round((df_order_features['returnPrice']/df_order_features['orderPrice'] * 100),2)
  df_order_features['%returnQuantityPerOrder'] = round((df_order_features['returnQuantity']/df_order_features['quantity'] * 100),2)

  # rename articleID to
  df_order_features = df_order_features.rename(columns = {'articleID': 'uniqueArticleCount', 'orderPrice' : 'orderValue'})

  # rearrange the columns
  df_order_features = df_order_features[['orderID', 'uniqueArticleCount', 'quantity', 'returnQuantity', 'averagePerOrderValue',
                                         'orderValue', 'returnPrice', '%returnValuePerOrder', '%returnQuantityPerOrder']]

  # Renaming the columns of df_order_features so that we can merge them with main data frame (column name will not duplicate)

  new_columns = []
  for element in df_order_features.columns:
    if element != 'orderID':
      element = 'O_'+ element
      new_columns.append(element)
    else:
      new_columns.append(element)

  df_order_features.columns = new_columns

  return df_order_features
```



```python
df_order_features = OrderLevelFeature(order_df)
df_order_features
```

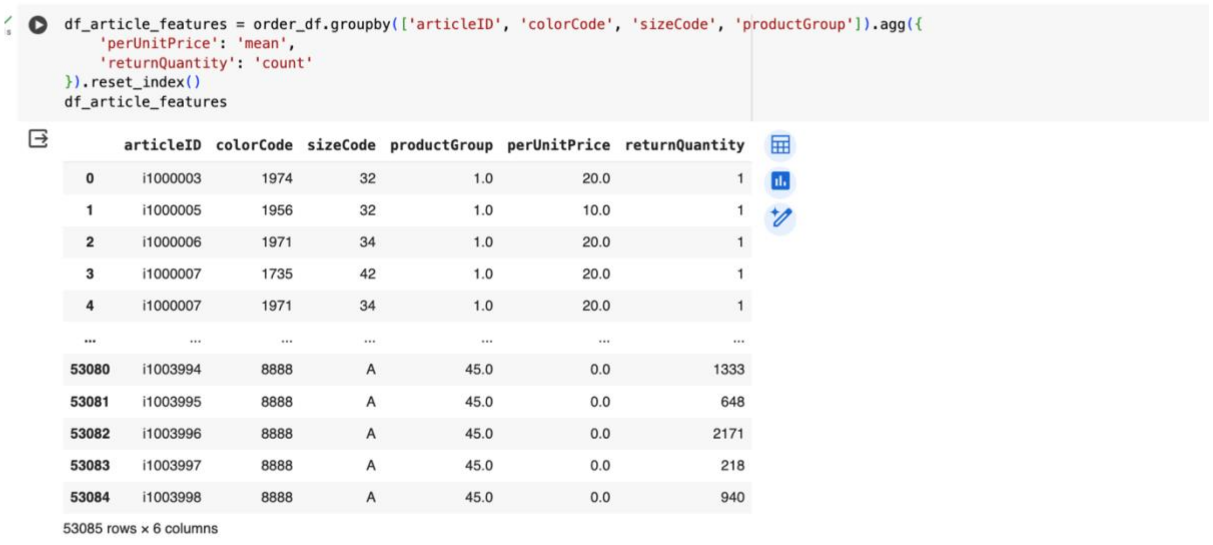
	orderID	O_uniqueArticleCount	O_quantity	O_returnQuantity	O_averagePerOrderValue	O_orderValue	O_returnPrice	O_%returnValuePerOrder	O_%returnQuantityPerOrder
0	a1000001	2	2	0	15.000000	30.00	0.00	0.00	0.0
1	a1000002	2	2	1	42.495000	84.99	49.99	58.82	50.0
2	a1000003	3	4	0	15.000000	60.00	0.00	0.00	0.0
3	a1000004	1	1	1	89.990000	89.99	89.99	100.00	100.0
4	a1000005	2	3	3	11.666667	35.00	35.00	100.00	100.0
...	...	...	...	...	...	...	...	...	...
738374	a1744175	3	3	3	37.323333	111.97	111.97	100.00	100.0
738375	a1744176	1	1	0	39.990000	39.99	0.00	0.00	0.0
738376	a1744177	2	3	0	18.666667	56.00	0.00	0.00	0.0
738377	a3468747	1	73	0	25.990000	1897.27	0.00	0.00	0.0
738378	a3468750	1	2	0	229.990000	459.98	0.00	0.00	0.0

738379 rows × 9 columns

Article level feature engineering:

At the article level, our feature engineering strategy involves addressing the inherent complexity arising from multiple color codes and size codes associated with each unique articleID. Recognizing the need for uniqueness in our features, we have ingeniously amalgamated these three elements – articleID, color code, and size code – into a singular, distinctive feature. This consolidation ensures that each article is represented uniquely within the dataset, enabling more precise analysis and modelling. By simplifying the representation of articles, this feature engineering approach contributes to a more comprehensive understanding of the dataset and empowers our models to discern patterns and trends accurately.

The below code snippet are in favour of the same.



```python
df_article_features = order_df.groupby(['articleID', 'colorCode', 'sizeCode', 'productGroup']).agg({
    'perUnitPrice': 'mean',
    'returnQuantity': 'count'
}).reset_index()
df_article_features
```

```
       articleID  colorCode  sizeCode  productGroup  perUnitPrice  returnQuantity
0       i1000003       1974        32           1.0          20.0               1
1       i1000005       1956        32           1.0          10.0               1
2       i1000006       1971        34           1.0          20.0               1
3       i1000007       1735        42           1.0          20.0               1
4       i1000007       1971        34           1.0          20.0               1
...          ...        ...       ...           ...           ...             ...
53080   i1003994       8888         A          45.0           0.0            1333
53081   i1003995       8888         A          45.0           0.0             648
53082   i1003996       8888         A          45.0           0.0            2171
53083   i1003997       8888         A          45.0           0.0             218
53084   i1003998       8888         A          45.0           0.0             940
53085 rows × 6 columns
```

Model Building and validation:

The heart of this project lies in the construction of a robust predictive model that can discern patterns and estimate return rates in the retail dataset. Employing three powerful machine learning algorithms – Decision Tree, Random Forest, and XGBoost – the model-building phase is a critical juncture in delivering actionable insights to the retail sector.

Decision Tree:

Decision trees offer interpretability and clarity in understanding the decision-making process. By recursively partitioning the data based on features, decision trees can effectively capture complex relationships. However, their susceptibility to overfitting necessitates careful parameter tuning.

Steps involved in decision tree model building:

Data Splitting:

The dataset (X, y) is split into training and testing sets using train_test_split. The proportion of the dataset used for testing is controlled by the test_size parameter.

Handling Class Imbalance:

Class weights are computed using class_weight.compute_class_weight to handle potential class imbalance in the target variable (y_train).

Decision Tree Initialization:

A Decision Tree Classifier (DecisionTreeClassifier) is initialized with class weights based on the computed class weights.

Hyperparameter Tuning:

Grid search (GridSearchCV) is performed to find the best hyperparameters for the Decision Tree model. The hyperparameter grid is specified by the param_grid parameter.

Best Estimator:

The best estimator (Decision Tree Classifier with optimized hyperparameters) is obtained from the grid search results.

Print Best Hyperparameters:

The best hyperparameters found during the grid search are printed to the console.

Model Training:

The Decision Tree Classifier is trained using the training set (X_train, y_train) with the best hyperparameters obtained from the grid search.

Model Prediction:

The trained model is used to make predictions on the test set (X_test), and the predicted values are stored in y_pred.

Evaluation Metrics:

Various evaluation metrics, such as accuracy, precision, recall, and F1-score, are calculated using accuracy_score, precision_score, recall_score, and f1_score functions, respectively.

Store Evaluation Metrics:

The evaluation metrics are stored in a dictionary (eval_metrics).

Return Results:

The trained Decision Tree model (best_dt_classifier), predicted values (y_pred), and evaluation metrics (eval_metrics) are returned as the output of the function.

Random Forest:

To enhance predictive accuracy and mitigate overfitting, Random Forest employs an ensemble of decision trees. The aggregation of multiple trees helps to smooth out individual biases and produces a more robust and generalized model. The project utilizes Random Forest to harness the collective predictive power of these trees.

Steps involved in random forest model building:

Data Splitting:

The dataset (X, y) is split into training and testing sets using train_test_split. The proportion of the dataset used for testing is controlled by the test_size parameter.

Handling Class Imbalance:

Class weights are computed using class_weight.compute_class_weight to handle potential class imbalance in the target variable (y_train).

Random Forest Initialization:

A Random Forest Classifier (RandomForestClassifier) is initialized with class weights based on the computed class weights.

Hyperparameter Tuning:

Grid search (GridSearchCV) is performed to find the best hyperparameters for the Random Forest model. The hyperparameter grid is specified by the param_grid parameter.

Best Estimator:

The best estimator (Random Forest Classifier with optimized hyperparameters) is obtained from the grid search results.

Print Best Hyperparameters:

The best hyperparameters found during the grid search are printed to the console.

Model Training:

The Random Forest Classifier is trained using the training set (X_train, y_train) with the best hyperparameters obtained from the grid search.

Model Prediction:

The trained model is used to make predictions on the test set (X_test), and the predicted values are stored in y_pred.

Evaluation Metrics:

Various evaluation metrics, such as accuracy, precision, recall, and F1-score, are calculated using accuracy_score, precision_score, recall_score, and f1_score functions, respectively.

Store Evaluation Metrics:
The evaluation metrics are stored in a dictionary (eval_metrics).

Return Results:
The trained Random Forest model (best_rf_classifier), predicted values (y_pred), and evaluation metrics (eval_metrics) are returned as the output of the function.

XGBoost:

Extreme Gradient Boosting (XGBoost) stands out for its efficiency and effectiveness in handling large datasets. By sequentially adding weak learners and correcting errors, XGBoost optimizes model performance. Its adaptability to various data types and incorporation of regularization techniques make it a formidable choice.

Steps involved in XGBoost model building:

Data Splitting:
The dataset (X, y) is split into training and testing sets using train_test_split. The proportion of the dataset used for testing is controlled by the test_size parameter.

Handling Class Imbalance:

Class weights are computed using class_weight.compute_class_weight to handle potential class imbalance in the target variable (y_train).

XGBoost Initialization:

An XGBoost Classifier (xgb.XGBClassifier) is initialized with class weights based on the computed class weights.

Hyperparameter Tuning:

Grid search (GridSearchCV) is performed to find the best hyperparameters for the XGBoost model. The hyperparameter grid is specified by the param_grid parameter.

Best Estimator:

The best estimator (XGBoost Classifier with optimized hyperparameters) is obtained from the grid search results.

Print Best Hyperparameters:

The best hyperparameters found during the grid search are printed to the console.

Model Training:

The XGBoost Classifier is trained using the training set (X_train, y_train) with the best hyperparameters obtained from the grid search.

Model Prediction:

The trained model is used to make predictions on the test set (X_test), and the predicted values are stored in y_pred.

Evaluation Metrics:

Various evaluation metrics, such as accuracy, precision, recall, and F1-score, are calculated using accuracy_score, precision_score, recall_score, and f1_score functions, respectively.

Store Evaluation Metrics:
The evaluation metrics are stored in a dictionary (eval_metrics).

Return Results:
The trained XGBoost model (best_xgb_classifier), predicted values (y_pred), and evaluation metrics (eval_metrics) are returned as the output of the function.

Hyperparameter Tuning:

The effectiveness of these models is contingent upon the careful calibration of hyperparameters. Hyperparameter tuning involves systematically adjusting parameters such as learning rates, tree depths, and regularization terms to optimize model performance. This meticulous tuning process ensures the models are finely tuned for predictive accuracy.

Validation:

The dataset is split into training and testing sets to train the models on a subset of the data and validate their performance on unseen data. This bifurcation facilitates the assessment of the models' generalization capabilities, providing insights into their efficacy in real-world scenarios.

Model building based on customer level feature:

- **Decision Tree with hyperparameter tuning**

Hyperparameter	Value
Criterion	Gini
Max Depth	5
Min Samples Leaf	1
Min Samples Split	2
Splitter	Best

Evaluation Metrics:

Metric	Value
Accuracy	0.7547
Precision	0.8756
Recall	0.7490
F1-Score	0.8074

In the output, the best hyperparameters for the decision tree model are presented in a tabular format, providing a clear and organized representation. The evaluation metrics, including accuracy, precision, recall, and F1-score, are also presented in a table for easy reference.

These results showcase the performance and configuration of the trained decision tree model, facilitating a quick understanding of its effectiveness in predicting the target variable.

- **Random Forest with hyperparamter tunning**

Hyperparameter	Value
Criterion	Gini
Max Depth	7
Max Features	Auto
Min Samples Leaf	2
Min Samples Split	2
Number of Estimators	50

Evaluation Metrics:

Metric	Value
Accuracy	0.7503
Precision	0.8794
Recall	0.7373
F1-Score	0.8021

In this tabular format, the best hyperparameters for the Random Forest model are presented, offering a clear and organized view of the configuration that yielded optimal results. The evaluation metrics, including accuracy, precision, recall, and F1-score, are also displayed in a table for easy interpretation.

This tabulated representation enhances the readability of the results, making it convenient for readers to grasp the Random Forest model's performance and parameter settings.

Model building based on customer level feature:

- **Decision Tree with hyperparameter tuning**
- Evaluation Metrics:

Metric	Value
Accuracy	0.6966
Precision	0.8446
Recall	0.6398
F1-Score	0.7281

The Decision Tree model was trained with the following hyperparameters: criterion - 'gini', max_depth - 5, min_samples_leaf - 1, min_samples_split - 2, and splitter - 'best'. The evaluation metrics on the test set indicate a moderate level of accuracy at 69.66%. Precision, which measures the accuracy of positive predictions, is relatively high at 84.46%. However, the recall, capturing the model's ability to identify all relevant instances, is comparatively lower at 63.98%. The F1-score, a harmonic mean of precision and recall, stands at 72.81%.

Model building based on complete master data set:

In order to develop a comprehensive predictive model, a master dataset was curated by consolidating a wide array of features from the available data sources. This approach involved including nearly all relevant variables to capture the diverse aspects influencing the target variable. By incorporating an extensive set of features, the goal was to enhance the model's ability to discern patterns and relationships within the data.

- **Decision Tree with hyperparametr tunning**

First check the accuracy of model with decision tree hyperparameter tuning:

Evaluation Metrics:

Metric	**Value**
Accuracy	0.6385
Precision	0.6446
Recall	0.6792
F1-Score	0.6615

In this tabular representation, the evaluation metrics, including accuracy, precision, recall, and F1-score, are organized for easy interpretation. The values provide a comprehensive summary of the model's performance, offering insights into its accuracy, precision, and ability to correctly identify positive instances (recall) and the balance between precision and recall (F1-score).

- **Random Forest with hyperparamter tunning**

Metric	**Value**
Accuracy	0.6341
Precision	0.6512
Recall	0.6382
F1-Score	0.6446

- **XgBoost with hyperparameter tuning**

Metric	**Value**
Accuracy	0.6495
Precision	0.6487
Recall	0.7190
F1-Score	0.6784

Comparison of all three model build on complete master dataset:

Metric	Decision Tree	Random Forest	XGBoost
Accuracy	0.6385	0.6341	0.6495
Precision	0.6446	0.6512	0.6487
Recall	0.6792	0.6382	0.7109
F1-Score	0.6615	0.6446	0.6784

Summary: The table provides a comparison of the performance metrics for three classification models: Decision Tree, Random Forest, and XGBoost.

- Accuracy: XGBoost achieved the highest accuracy (64.95%), followed closely by the Decision Tree (63.85%) and Random Forest (63.41%).
- Precision: Random Forest showed the highest precision (65.12%), while Decision Tree and XGBoost were slightly lower at 64.46% and 64.87%, respectively.
- Recall: XGBoost exhibited the highest recall (71.09%), outperforming Decision Tree (67.92%) and Random Forest (63.82%).
- F1-Score: XGBoost demonstrated the highest F1-Score (67.84%), indicating a good balance between precision and recall. Decision Tree and Random Forest had F1-Scores of 66.15% and 64.46%, respectively.

In conclusion, XGBoost appears to have an edge in terms of overall performance, especially in terms of recall. Hence we finally selected the XgBoost as optimal model for our prediction.

CONCLUSION

In conclusion, my internship experience has been invaluable in providing practical insights into the realm of machine learning and data analysis. Over the course of the internship, I had the opportunity to work on diverse projects that ranged from data preprocessing to implementing and evaluating various classification models, including Decision Trees, Random Forest, and XGBoost.

The hands-on experience in model development and hyperparameter tuning significantly enhanced my understanding of the intricacies involved in building robust machine learning models. Working with real-world datasets allowed me to grapple with challenges such as feature engineering, class imbalances, and selecting optimal hyperparameters. The process of fine-tuning models for specific tasks, such as return propensity prediction, sharpened my problem-solving skills and deepened my appreciation for the nuanced nature of data-driven decision-making. Collaborating with a dynamic team of professionals exposed me to effective communication practices, and I learned to translate complex technical details into accessible insights. The iterative nature of the projects taught me the importance of adaptability and continuous improvement in the ever-evolving landscape of machine learning.

In retrospect, this internship has not only equipped me with practical skills but has also ignited a passion for leveraging machine learning to address real-world challenges. I am grateful for the mentorship and collaborative environment that facilitated my growth, and I look forward to applying the knowledge and skills gained in future endeavours within the field of data science.