IS424 Data Mining and Business Analytics

G1T3 Final Report

Less for More: Retail Returns Classification with Logistic Regression, Support Vector Machine, RandomForest, XGBoost and Deep Neural Network

Name	Student ID
Chang Wai Phang Peter Gabriel	01369357
Cheyenne Loh Xinyi	01410070
Deviyani Patro	01439552
Min Thu Kha	01372965
Ryan Yak	01377187
Wong Kelly	01406153
Xavier Boon Santimano	01446440

Table of Contents

Executive Summary

This report commences with an Introduction & Motivation section, which frames the inquiry within the expansive growth of e-commerce, which is expected to constitute a 41% share of global retail by 2027 (Boston Consulting Group, 2023). As online sales soar, a corresponding issue has arisen: the increasing rate of retail returns, 20.8% in 2022, marking a significant leap from 18.1% in the 2021 *(*Repko, 2022). This trend is especially pronounced in the apparel sector.

The subsequent Problem Statement section posits the core challenge: addressing the cost implications of these rising return rates. This part of the report specifies the use of binary classification models as a predictive tool to anticipate e-commerce returns, a novel approach aimed at pre-emptively identifying potential returns before they occur.

In the Literature Overview, we consolidated existing knowledge and previous studies, setting the stage for our empirical investigation presented in the Dataset and Methodology sections. These sections detail the data compilation processes and the rigorous Data Pre-processing techniques utilized to prepare the dataset for analysis.

A core component of the report is the Models section, which assesses the effectiveness of various algorithms in predicting e-commerce returns. Techniques such as Logistic Regression, Support Vector Machine, Random Forest, XGBoost, and Deep Neural Network are compared to ascertain the most effective model based on Area-Under-Curve score.

Results & Discussions synthesize the data findings, provide critical evaluation of the limitations encountered, and articulate the implications of the results. The study's findings not only contribute to academic discourse but also offer practical recommendations for online retailers.

Finally, the Conclusion & Future Work encapsulates the insights gleaned from the research and proposes strategic interventions and areas for future exploration. The report offers a strategic roadmap for stakeholders grappling with the operational and financial impacts of product returns in e-commerce.

1. Introduction & Motivation

E-commerce's share of global retail is expected to increase from 18% in 2017 to 41% by 2027, underscoring its significant growth potential (Boston Consulting Group, 2023). However, alongside this growth, online sales face notable challenges, including a steep rise in the rate of retail returns, which stood at 20.8% in 2022 compared to 18.1% the previous year (Repko, 2022). With such high rates of return, the retail sector incurs costs of approximately $642 billion annually to deal with return requests (nShift, 2023).

Given these dynamics, it is imperative to analyse and understand online customer return patterns. Doing so can help mitigate returns and balance customer satisfaction with effective return management strategies, ultimately enhancing online sales performance.

Our project aims to leverage multiple supervised machine learning techniques to predict return rates. We aim to provide actionable insights that empower ecommerce retailers to optimize inventory, enhance customer satisfaction, and mitigate the economic impact of returns on discounted sales.

2. Problem Statement

Our project aims to develop a binary classification model that more accurately predicts customer returns, surpassing current market offerings. By identifying and analysing key variables that influence return rates, we will enhance the ability of e-commerce retailers to manage returns effectively.

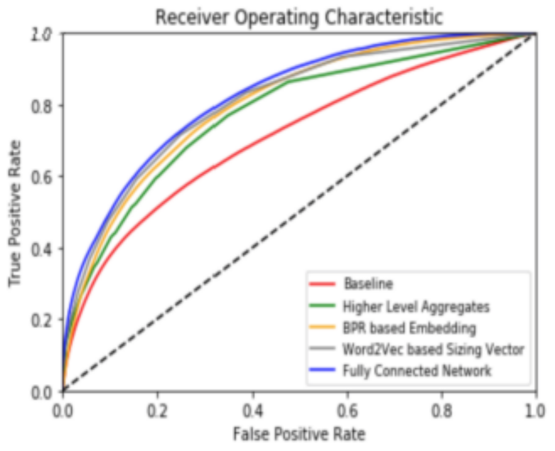
This solution is expected to aid companies to optimize inventory management, improve marketing strategies, and ultimately enhance profitability and customer experience by reducing the economic and operational impacts of returns.

3. Literature Overview

3.1 Predicting product return volume using machine learning methods (Cui et al., 2020)

Main Aim	Construct an effective data-driven model for forecasting future return volume, with good out-of-sample prediction accuracy.
Methodology	• **Modelling Techniques Used:** Linear Regression, Random Forest, Gradient Boosting • **Performance Metrics:** Mean Squared Error (MSE) and R^2 • **Variables:** o Predictor variables based on dataset and research o 2-way and 3-way interaction terms o Feature selection techniques (LASSO, LARS-OLS hybrid, SCAD and Elastic Net) employed
Notable Results	Linear Regression with LASSO obtained the smallest test MSE among all considered models. Additionally, both its training and test MSE were close, indicating it is a robust model.

3.2 Early Bird Catches the Worm: Predicting Returns Even Before Purchase in Fashion E-commerce (Kedia et al., 2019)

Main Aim	• To predict a customer's likelihood of return in advance during browsing or at shopping cart page so that e-tailers can take preventive measures • The return probability is forecasted at two levels: cart-level and product-level • Product-level model (pre-determined as gradient boosted classifier) is built on the cart-level model
Methodology	• **Relevant Modelling Techniques Used:** Deep Neural Network (DNN) Model, Gradient Boosted Classifier • **Performance Metrics:** Precision, Recall, AUC, ROC • **Variables:** o Product embeddings using Bayesian Personalized Ranking (BPR) captured users' taste and products' latent hidden features o Embedding using skip-gram model captured users' body shape and size (Word2Vec) o Extensive feature engineering was done – features are broadly categorized into product level features, cart level features, user level features
Notable Results	Amongst all the models considered for cart-level classifier, fully connected DNN model proved to be the best model with highest AUC of 83.2%, highest precision of 74% and highest area under the ROC.  *Figure 3.1 Receiver Operation Characteristic (ROC) curves, which shows fully connected DNN model to be the best (Kedia et al., 2019)*

3.3 Encoding high dimension categorical attributes (Udilâ, et al., 2023)

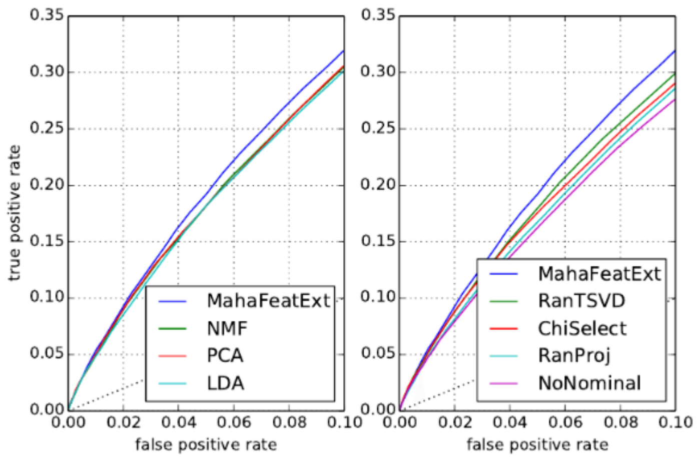
Main Aim	Assessing Encoding Techniques for Categorical Data: Comparing One-Hot, Ordinal, Target, CatBoost, and Count Encoders Across Linear Models, Decision Trees, and SVMs.
Methodology	**Relevant Technique:** Target Encoding • Replaces each category with the mean (or some other aggregation) of the target variable for that category • Provides a more informative representation of the data than one-hot encoding or ordinal encoding • Only use train set for encoding to prevent overfitting
Notable Results	Not applicable

3.4 Predicting Product Returns in E-Commerce: The Contribution of Mahalanobis Feature Extraction (Urbanke et al., 2015)

Main Aim	• To develop a machine learning algorithm (Mahanobis Feature Extraction) which scales well to sparse matrices, which is the case with e-commerce returns data • To compare this algorithm against other commonly used dimensionality reduction algorithms
Methodology	• **Relevant Modelling Techniques Used:** Decision Trees, Random Forest, Adaptive Boost, Gradient Boost, Linear Discriminant Analysis, Logistic Regression, Support Vector Machine • **Performance Metrics:** Precision, Recall, AUC, ROC • **Variables:** o Predictor variables broadly categorized into product level features, cart level features, user level features o Dimensionality reduction methods, such as: Mahanobis Feature Extraction, Principal Component Analysis and truncated singular value decomposition

Notable Results

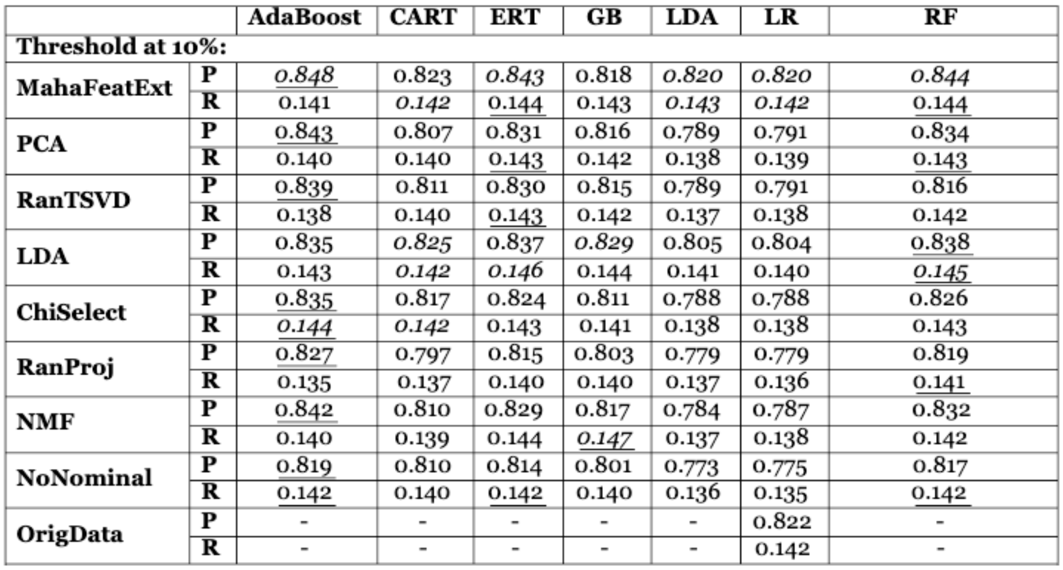
Amongst all the combinations considered for classifying returns, the Mahanobis Feature Extracted method resulted in the best ROC scores across the board.



Mahalanobis feature extraction = **MahaFeatExt**, principal component analysis = **PCA**, linear discriminant analysis = **LDA**, randomized truncated singular value decomposition = **RanTSVD**, feature selector based on univariate chi-squared statistic = **ChiSelect**, random projection = **RanProj**, non-negative matrix factorization = **NMF**, no nominal indicators = **NoNominal**

Figure 2. Out-of-sample ROC-curves for different feature extractors, in combination with best classifier

Figure 3.2 Receiver Operation Characteristic (ROC) curves, which shows performance of best classifier for different dimensionality reduction methods (Urbanke et al., 2015)



		AdaBoost	CART	ERT	GB	LDA	LR	RF
Threshold at 10%:								
MahaFeatExt	**P**	*0.848*	0.823	*0.843*	0.818	*0.820*	*0.820*	*0.844*
	R	0.141	*0.142*	0.144	0.143	*0.143*	*0.142*	0.144
PCA	**P**	0.843	0.807	0.831	0.816	0.789	0.791	0.834
	R	0.140	0.140	0.143	0.142	0.138	0.139	0.143
RanTSVD	**P**	0.839	0.811	0.830	0.815	0.789	0.791	0.816
	R	0.138	0.140	0.143	0.142	0.137	0.138	0.142
LDA	**P**	0.835	*0.825*	0.837	*0.829*	0.805	0.804	0.838
	R	0.143	*0.142*	*0.146*	0.144	0.141	0.140	*0.145*
ChiSelect	**P**	0.835	0.817	0.824	0.811	0.788	0.788	0.826
	R	*0.144*	*0.142*	0.143	0.141	0.138	0.138	0.143
RanProj	**P**	0.827	0.797	0.815	0.803	0.779	0.779	0.819
	R	0.135	0.137	0.140	0.140	0.137	0.136	0.141
NMF	**P**	0.842	0.810	0.829	0.817	0.784	0.787	0.832
	R	0.140	0.139	0.144	*0.147*	0.137	0.138	0.142
NoNominal	**P**	0.819	0.810	0.814	0.801	0.773	0.775	0.817
	R	0.142	0.140	0.142	0.140	0.136	0.135	0.142
OrigData	**P**	-	-	-	-	-	0.822	-
	R	-	-	-	-	-	0.142	-

Figure 3.3 Precision and recall for different dimensionality reduction methods and classification models, with 10% classification threshold (Urbanke et al., 2015)

3.5 Implications

These are the key lessons to be drawn from the existing literature, which are highly relevant to the development of our classification model and evaluation of it.

1. The clarity of relationship between independent variables and returns will determine the best model to use, for instance linear regression in 3.1 and DNN in 3.2.
2. A DNN model could better capture the complex relationship of our dataset compared to more traditional models
3. To minimize likelihood of returns, some companies may opt for an exceptionally low classification threshold like 10%, which in-turn reduces the recall to abysmal values.
    - In particular, Urbanke et al.'s research paper seems to prioritise maximising precision over recall, we believe that it is more important to maximise the recall (TP/(TP+FN)) as this ensures that we capture more actual returns.
    - We aim to create a more balanced model that captures a good proportion of true returns without falsely flagging customers (this is important for companies that wish to apply penalties, incentives, or other types of methods to customers who they flag as likely to return an item)
4. We want to apply manual feature extraction and feature selection methods, as opposed to relying on dimensionality reduction algorithms, to create a model that is more understandable to the business and less black box in nature

4. Dataset

The dataset (Alisa_K, 2024) that we plan to use comes from a Kaggle competition and its description is as follows in Table 4.1. It comprises of authentic customer orders from a German online fashion retailer, containing data regarding both orders and customers. The dataset is especially helpful when using classification techniques, as it is large and is densely labelled. It consists of 100,000 rows and 14 attributes explained in the table below.

	Attributes	**Description**
1	order_item_id	Unique identifier of each order.
2	order_date	Date when the order was placed.
3	delivery_date	Date when the order was delivered to the customer.
4	item_id	Unique identifier for each item.
5	item_size	Size of the item ordered.
6	item_color	Color of the item ordered.
7	brand_id	Identifier for the brand of the item.
8	item_price	Price of the item.
9	user_id	Unique identifier for each customer.
10	user_title	Title of the customer (e.g., Mr., Mrs., Ms.).
11	user_dob	Date of birth of the customer.
12	user_state	State where the customer resides.
13	user_reg_date	Registration date of the customer.
14	return	A binary value on whether the order was returned, with 0 representing returned order and 1 being non-returned order.

Table 4.1 Table of each attribute's description for our dataset

5. Methodology

5.1 Data Pre-processing

The table below outlines the pre-processing steps taken in our dataset. It includes exploratory data analysis, data cleaning, transformation and standardization, feature engineering and feature selection.

No.	**Description**	**Corresponding Figure in Appendix**
1	Analyse the dimensions of our data. Our data consists of 100,000 rows and 14 attributes. We observed that only 2 attributes have missing data - delivery date and user date of birth. We also observed that some of the time data (eg. order, delivery dates) are in object data type, instead of date data type (datetime64).	*Figure 5.1*
2	Analyse the descriptive stats of the numerical attributes. Since order item id, item id, brand id and user id are nominal categorical attributes, there is no need to analyse their descriptive stats.	*Figure 5.2*
3	Investigating the absence of delivery dates in our data, we examine whether products lacking delivery dates were returned. We discovered that none of the products without delivery dates were returned. To represent this observation and prediction for entries with missing delivery information in the test data, we introduce a new attribute "no_delivery," assigning a value of 1 to products with missing delivery dates and 0 to those with complete delivery dates.	*Figure 5.3*
4	In analyzing the item price, we found items with price = 0. There are 2 sizes with price=0, “unsized” and size =40. • Price = 0, size = “unsized”. 395 rows of data. We assume that these were gifts, so we will be leaving them untouched. • Price = 0, size = 40. 1 row of data. Dropped the single row as we assume it is an error, and the frequency of it is insignificant. This leaves our data with 99,999 rows.	*Figure 5.4*

5	Convert all date variables into date datatype. The attributes converted were order date, delivery date, user date of birth and user registration date.	*Figure 5.5*
6	Our item sizes are labelled in 3 different ways: 1. Integer (eg. 34,56,43) 2. Integer + (eg. 8+,39+) 3. String (eg. “m”, “s”, “xs”) We standardize the size to just the string size. 1. We take sizes with integer“+” to mean that it is considered the next biggest size. (eg. 9+ to 10). 2. For integer and integer + sizes, we mapped them to descriptive string size categories, based on predefined limits associated with pants sizes(Alterations Express, 2023) and child sizes (*Baby & Kids Clothes*, n.d.).	*Figure 5.6,* *Figure 5.7*
7	Created gender attribute ("is_female” and “is_male”) from “user_title”. This variable was homogenous towards females (more than 90% of the data) but we thought that maybe splitting them into 3 different categories (the invisible category represents families/companies) could provide some information regarding returns.	*Figure 5.8*
8	Created age attribute (“age”) to get age of user when they order a product by calculating difference between date of birth and order date.	*Figure 5.9*
9	Created age group attribute ("age_group”) by binning the ages by groups. The age groups are: 'Missing', '21 and Under', '22-30', '31-45', '46-55', '56-65', '65 and Above'. We experimented to see whether age or age group is more statistically effective and found age group to be more statistically effective.	*Figure 5.10*
10	Created delivery days attribute “delivery_days“ that contains information on delivery time (in terms of number of days) by calculating difference between delivery and order date. We found that there are instances of negative waiting time, which is not possible since a product cannot be delivered before a customer order. We	*Figure 5.11*

	replace those negative waiting time with median of waiting time where waiting time > 0.	
11	Created attribute “no_items” which is the number of products purchased by a customer for each order, by grouping user id and order date.	*Figure 5.12*
12	Created attribute “avg_freq_purchases” that stores average monthly purchases per user per order.	*Figure 5.13*
13	Created attribute “is_special” to check if the order is within 10 days before or after a special occasion, and 2 days before or after Black Friday and Cyber Monday (since the sale periods are usually shorter for these sale events).	*Figure 5.14, Figure 5.15*
14	Created attribute “is_bday” to check if orders or deliveries were made within 30 days before or after user birthday.	*Figure 5.16*
15	Split the data into train-test split, with 80% training data (“X_encode”, “y_train”) and 20% test data (“X_test”, “y_test”). To prevent data leakage and overfitting we will create and map training features to the entire dataset. We used the train test split with same seed to create variables without data leakage.	*Figure 5.17*
16	Check for skewness for all numeric variables, which are price (“item_price”), no of items (“no_items”), delivery days (“delivery_days”), age, and average frequency of purchases (“avg_freq_purchases”). We found that they are right skewed, so we log-transformed all of them. For delivery days, added 0.01 to account for cases where delivery days = 0. For price, added 10 to account for cases where price= 0.	*Figure 5.18, Figure 5.19, Figure 5.20*

Attributes	**Original Skewness**	**Skewness after transformation**
Item_price	1.980993	0.067159
no_items	2.821200	- 0.059183
delivery_days	4.022536	0.433625
age	-2.575843	- 0.197428
avg_freq_purchases	4.500025	0.049956

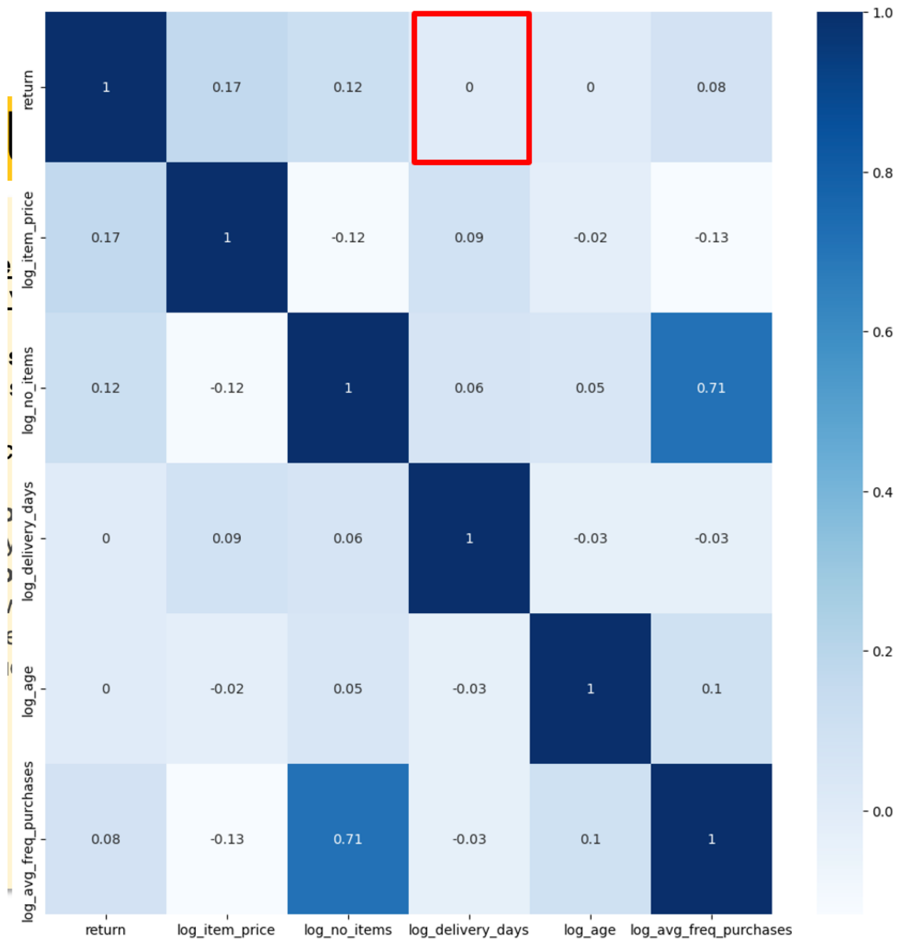
	Table 5.26 Attributes skewness before and after transformation We then standardized log-transformed price, no_items, age, delivery_days and avg_freq_purchases.	
17	Investigates correlation between categorical attributes using Chi-squared tests (Melanie, 2024). Setting p value of 0.01, it was found that order_item_id, is_special and least_returned_size are not correlated with returns.	*Figure 5.21*
18	We dropped log_delivery_days because the correlation was approximately 0. We kept log_age as although it may not be linearly correlated with returns, certain ranges of age imply a higher/lower rate of return.  |	return	log_item_price	log_no_items	log_delivery_days	log_age	log_avg_freq_purchases | return: 1, 0.17, 0.12, 0, 0, 0.08 log_item_price: 0.17, 1, -0.12, 0.09, -0.02, -0.13 log_no_items: 0.12, -0.12, 1, 0.06, 0.05, 0.71 log_delivery_days: 0, 0.09, 0.06, 1, -0.03, -0.03 log_age: 0, -0.02, 0.05, -0.03, 1, 0.1 log_avg_freq_purchases: 0.08, -0.13, 0.71, -0.03, 0.1, 1	*Figure 5.22,* *Figure 5.23*

	Figure 5.27 Correlation heatmap, with return and log_delivery_days correlation highlighted We also dropped attributes that were engineered on and insignificant attributes. The dropped attributes are item_id, item_color, user_title, order_item_id, order_date, delivery_date, user_dob, user_reg_date, age_group, is_special, least_returned_size.	
19	One hot encoded 'user_state', 'item_size'.	*Figure 5.24*
20	To prevent data leakage from the training set (‘X_encode’) to the test set, we mapped average return rates for users and brands from the training data to their respective counterparts in the test set. Any missing values for new users were conservatively filled with 0.5.	*Figure 5.25*

These are the final variables after data pre-processing:

- return (target)
- no_delivery
- is_female, is_male
- is_bday
- most_returned_item, least_returned_item
- most_returned_color, least_returned_color
- log_item_price
- log_no_items
- log_age
- log_avg_freq_purchases
- user_state_Baden-Wuerttemberg, user_state_Bavaria, user_state_Berlin, user_state_Brandenburg, user_state_Bremen, user_state_Hamburg, user_state_Hesse, user_state_Lower Saxony, user_state_Mecklenburg-Western Pomerania, user_state_North Rhine-Westphalia, user_state_Rhineland-Palatinate, user_state_Saarland, user_state_Saxony, user_state_Saxony-Anhalt, user_state_Schleswig-Holstein, user_state_Thuringia
- item_size_child, item_size_l, item_size_m, item_size_s, item_size_unsized, item_size_xl, item_size_xs, item_size_xxl, item_size_xxxl
- user_return_rate
- brand_return_rate

5.2 Modelling Process

5.2.1 Models Considered

Our literature review, intuition and additional research led to the testing of the following models: Logistic Regression, Support Vector Machine, Random Forest, XGBoost, and Deep Neural Network.

We briefly explain the rationale for choosing each model for our problem in the table below.

Model	Rationale
Logistic Regression	Logistic regression is a linear model that can be used for binary classification. It estimates the probability that a given instance belongs to a particular class by fitting the input features to a logistic/sigmoid function that maps real value number to a value between 0 and 1. It is easy to implement and train, performs well when the dataset is linearly separable and over-fitting can be reduced with the used of regularisation techniques. (*Advantages and disadvantages of logistic regression*, 2023)
Support Vector Machines (SVM)	The strength of Support vector machines (SVM) leveraged for predicting returns were that SVM being a set of supervised learning methods used for classification, is effective in high dimensional spaces, (*sci-kit learn*, 2024). Our data set with a high dimension space of 39 attributes made SVM a suitable classification algorithm choice. However, while SVM does not inherently provide AUC metrics, we still chose to evaluate its performance using accuracy as a benchmark against other models so that we can gain insights into SVM's classification capability relative to other models.
Random Forest	Random Forest is a powerful ensemble learning technique that combines multiple decision trees to improve prediction accuracy and control over-fitting (*sci-kit learn*, 2024). Each tree is trained on a random subset of the data, ensuring diversity and robustness in the model's predictions. This method is particularly effective for classification tasks within complex and high-

	dimensional datasets, making it well-suited for challenges like predicting item returns where discerning patterns within noisy data is crucial.
XGBoost	XGBoost is renowned for its efficiency and performance, particularly in structured data scenarios. It employs gradient boosting to build predictive models, which is beneficial for high-dimensional datasets. Its strength lies in handling sparse data and applying regularization to reduce overfitting (*Simplillearn*, 2023), making it ideal for complex problems like item return predictions, where discerning subtle patterns is key. This capability to manage closely clustered data points effectively suggests XGBoost could be particularly useful for our project.
Deep Neural Network (DNN)	Deep Neural Networks (DNN) are a class of machine learning algorithms inspired by the structure and function of the human brain. They consist of multiple layers of interconnected neurons, allowing them to learn complex patterns and relationships in the data (*What is deep learning*?, n.d). DNN is chosen for its unparalleled proficiency in discerning intricate patterns within data. With multiple layers of neurons, the network can efficiently process vast datasets and automatically extract significant features (*Understanding deep learning: Exploring its advantages and disadvantages*, 2023). Moreover, its highly non-linear nature allows for the creation of complex relationships between input features and the target variable (Bouniot et al., 2023).

5.2.2 Model Evaluation Criteria

We utilized the **test ROC-AUC score** as the primary metric to assess the performance of our models' predictive ability on **unseen** data, where the best-performing models are identified by having the highest test ROC-AUC score. Unlike other metrics that require the specification of a threshold (for example, accuracy and precision), the ROC-AUC score provides a comprehensive evaluation of the model's ability to differentiate between positive and negative instances across all classification thresholds.

The score ranges from 0 to 1, where 0.5 indicates random guessing and 1 signifies perfect performance. By focusing on maximizing the test ROC-AUC score, we ensure that the selected models exhibit robust performance in classifying unseen data, thus enhancing their practical utility and reliability in real-world scenarios.

5.2.3 General Procedure in our Analysis

We adopted the following process to find our best model: we crafted baseline models, conducted cross-validation for hyperparameter tuning based on ROC-AUC scores, and refined models to optimal parameters. In refining our models, we adopted a phased approach to hyperparameter tuning for efficiency and to focus on model aspects sequentially. Initially, we targeted parameters influencing model complexity, subsequently adjusting others like learning rate and randomness, ensuring a structured and prioritized optimization process.

To ensure our model accurately reflected improvements over the baseline, we addressed data leakages by manually dividing the dataset into 5 folds, stored in a matrix, recalculating 'user_return_rate' and 'brand_return_rate' with each fold's training data. We kept the categorization of other variables (such as the most/least returned- variables) unchanged to align with the test set. Due to these custom steps, we also developed our own grid search function, which navigates through the specified parameter space, assessing models based on their ROC-AUC score, as the scikit-learn's GridSearchCV could not accommodate our process.

Finally, we consolidate the best performing models from each model category and compare their performance to determine our overall best model.

5.2.4 Model Category Specific Analysis

In this section, we explain our choice of hyperparameters to tune, the selected range of those hyperparameters and any other considerations/steps taken that we deem relevant for each model category.

Logistic Regression

The baseline model for logistic regression was constructed using the initial parameters provided by scikit-learn Logistic Regression.

After constructing the baseline model, we decided to employ three types of regularisation techniques (LASSO, Ridge and Elastic Net). The table below summarises the characteristics of each regularisation technique. The three regularisation techniques have different penalty terms and handle multicollinearity differently.

LASSO	Ridge	Elastic Net
• Penalizes absolute value of coefficients • Shrink coefficients of less important predictors to exactly zero (model simplification and variable selection) • In case of correlated predictors, LASSO selects one predictor and drives the coefficients of others to zero	• Penalizes squared value of coefficients • Shrinks coefficients of less predictors proportionally (close to zero) • Reduces impact of multicollinearity by reducing the influence of correlated predictors	• Combines LASSO penalty and Ridge penalty • Selects one variable from group of correlated variables and assign it non-zero coefficient while shrinking coefficients of other correlated variables towards zero

Table 5.2 Comparison between three regularisation techniques for logistic regression

We first fixed the penalty term for each of the regularisation techniques we were considering. We did so by setting the parameter penalty = ‘l1’/ ‘l2’/ ‘elasticnet’, where ‘l1’ corresponds to LASSO penalty, ‘l2’ corresponds to Ridge penalty and ‘elasticnet’ corresponds to Elastic Net penalty. We also set the parameter solver = ‘saga’ for all as it could handle all three types of regularization and we standardized the parameter random state to 42.

For each penalty, we considered the following hyperparameters for tuning:

1. C: Inverse of Regularization Strength (Smaller values specify stronger regularization)

   We considered the range (0.001, 0.01, 0.1, 1, 10, 100) to explore a wide range of regularization strengths.

2. (For Elastic Net Only) l1_ratio: Parameter weighting l1 vs l2 regularization

   We considered the range (0.1, 0.3, 0.5, 0.7, 0.9) to explore different combinations of l1 penalty term and l2 penalty term.

To select our best model, we compare the cross-validated (CV) ROC-AUC scores of the tuned models for each regularization technique to select the best tuned model. Afterwards, we obtain the test ROC-AUC score of the best tuned model by building it on the train dataset. We compare the value obtained with that of the baseline model we have generated.

Unlike other complex models like RandomForest and XGBoost, since logistic regression does not handle non-linear effects, we also considered using interaction terms to capture any pertinent complex relationships that could exist between variables in our dataset. We present the interaction terms and their justifications in the table below.

Interaction Term	Justification
log_item_price * is_bday	If the price seems too high for what a user receives, the user will be more likely to return the item. However, this also depends on how urgently the user needs the item. In this case, is_bday can be a proxy for how urgently the user needs the item.
log_age * log_avg_freq_purchases	If the user is older, the user may be more risk-averse and sceptical of return policies and therefore, be less likely to return the item. However, this also depends on how active an ecommerce user he/she is, which is proxied by log_avg_freq_purchases.
log_no_items * log_avg_freq_purchases	If a user buys many items in one go, he/she is more likely to return an item. However, this also depends on how frequently the user buys online.
log_item_price * brand_return_rate	If a user finds the item too expensive, he/she is more likely to return the item. However, this also depends on the brand prestige (proxied by brand_return_rate).
most_returned_colour * brand_return_rate least_returned_colour * brand_return_rate	Popularity of colour is proxied by most_returned_colour/least_returned_colour. Most_returned_colour implies least popular colour. Least_returned_colour implies most popular.

	If a user does not like the colour, he/she is more likely to return the item. However, this depends on the brand prestige as well. If a user likes the colour a lot, he/she is less likely to return the item. However, this depends on the brand prestige as well.

We include the above-mentioned interaction terms in our dataset and redo the test train split with the dataset including the interaction terms. We repeat the same analysis as we did for Logistic Regression without interaction terms.
After the inclusion of interaction terms, we end up with 1 baseline model and 1 best tuned model for both without interaction terms and after interaction terms. We compare the test ROC-AUC scores of all 4 models to ultimately select the best one for logistic regression.

SVM

The baseline model built followed all initial parameters provided by the scikit-learn Linear Support Vector Classification (SVM.LinearSVC). This model is like SVM.SVC with parameter kernel='linear' but implemented in terms of liblinear rather than libsvm; liblinear is faster and can scale better to large numbers of samples. (Scikit-learn, n.d.). The random state was set to 42 to control the pseudo random number generation for shuffling the data for probability estimates[1]. (Scikit-learn, n.d.)
The most important parameters identified were

- C: The regularization parameter. (Trade-off between high size of the margin separating classes and misclassification of points).
- max_iter: The number of iterations to be run, balancing accuracy and model computation speed.
- Dual: Choice to use Optimize a higher-dimensional space (dual space), which may make the problem easier to solve, especially when dealing with non-linearly separable data. (*Dual support vector machine*, 2023).

As for the other parameters, they are all kept to the default:

  - penalty: l2
  - loss: squared_hinge

[1] Predicting the probability data points belong to a particular class: 0 or 1

    - dual: True
    - tol: 1e-4
    - multi_class: ovr
    - fit_intercept: True
    - intercept_scaling: 1.0
    - class_weight: None
    - max_iter: 1000

Random Forest

The baseline model built followed all initial parameters provided by the scikit-learn.

The parameters that we consider for the RandomForest classifier model from scikit-learn are (*Sklearn.ensemble.randomforestclassifier*, n.d.):

- n_estimators: Number of trees in the RandomForest model.
- criterion: Function to measure quality of split, either Gini impurity, information gain or log_loss.
- max_depth: Maximum depth of the decision trees in the forest. This parameter helps to prevent overfitting.
- min_samples_split: Minimum number of samples required to split an internal node. Setting a higher value can help prevent overfitting.
- max_features: Maximum number of features to consider when looking for the best split, either sqrt(number of features), log2(number of features) or no maximum number.

The baseline RadomForestClassifier() model is built with the random_state parameter set to 42 to initialize the internal random number generator which will decide splitting of data in the decision trees to allow deterministic and results to be produced for each run with the same results at the same random_state. As for the other parameters, they are all kept to the default:

- n_estimators: 100
- criterion: Gini impurity
- max_depth: None
- min_samples_split: 2
- max_features: sqrt(number of features)

To reach the final model state, we tuned the parameters in 2 phases – firstly focusing on the complexity of the trees (max_depth and min_samples_split), and then parameters relating to criterion, efficiency and sampling (criterion, n_estimators, max_features).

For complexity, the tuning involved conducting a broad search to find the best starting point, and then narrowing down the hyperparameter values to find an optimal. We first did a broad search for permutations of max_depth (None, 10, 20) and min_samples_split (2, 5) before conducting a narrowed search.

For criterion, efficiency and sampling, we found the best performing value for each hyperparameter individually. We considered the following values for each parameter:

- Criterion: gini, entropy
- N_estimators: [Broad: 50, 100, 150], [Narrow: 20 – 31]
- Max_features: sqrt, log2

XGBoost

The baseline model constructed followed the default parameters provided by the XGBoost library.
The parameters considered for the XGBoost classifier model are:

- Max Depth[2]: Depth of the trees, which helps control over-fitting.
- Minimum Child Weight[2]: Minimum sum of instance weight needed in a child to take further partitioning steps, also a measure to control over-fitting.
- Subsample[2]: Ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collected half of the data instances to grow trees and this will prevent overfitting.
- Column Sample by Tree[2]: Subsample ratio of columns when constructing each tree.
- Lambda[3]: The L2 regularization term on weights.
- ETA (Learning Rate): Step size shrinkage used in an update to prevent overfitting.
- Booster: Type of model to run at each iteration (gbtree, gblinear).
- Loss Function: The loss function to be minimized during training (binary logistic for binary classification, reg:squarederror for regression tasks).

[2] Note that these parameters are only considered for a tree-based model.
[3] Note that this parameter is only considered for a linear-based model.

For the baseline XGBoostClassifier() model, the following default parameters were set:

- Max Depth: 3
- Minimum Child Weight: 1
- Subsample: 1
- Column Sample by Tree: 1
- ETA (Learning Rate): 0.03
- Booster: Tree
- Loss Function: Binary Logistic

Parameter tuning for the final model state was approached in stages, initially focusing on the model structure by experimenting with different booster models (gblinear, gbtree) and objective functions (binary:logistic, reg:squarederror).

Next, we optimized the regularization via the lambda parameter, testing values (0, 0.1, 0.2, 0.3, 0.4, 0.5), and determined that a lambda value of 0 was the initial optima. From here, we had to explore smaller decimal steps to search for improvement to validation AUC.

We had a similar approach for ETA (learning rate), values (0.01, 0.05, 0.1, 0.2, 0.3) were trialed, and 0.01 was initially found to be the ideal learning rate – then we did a narrowed search using smaller decimal steps.

Lastly, we determined the number of boosting rounds needed for the model to converge. We set a limit of 10 early stopping rounds to avoid needless computation, with the tolerance = 0.00001.

DNN

DNN models in Keras require several key parameters for construction:

- Number of nodes in input layer: Number of input nodes that correspond to features of input data
- Number of hidden layer: Depth of neural network
- Number of hidden layer nodes: Number of nodes within each hidden layer.

- Number of nodes in output layer: Number of nodes in output layer that corresponds to number of classes for a classification problem
- Epochs: Number of times the dataset is passed forward and backward through the neural network during the training process
- Batch size: Number of samples of training data that is processed by the model in each training iteration. Large batch sizes results in faster training but may result in overfitting, while smaller batch sizes give a more accurate result, but is more computationally intensive

A baseline DNN model was first built with the following parameters:

- Number of nodes in input layer: 40 (there are 39 features and 1 extra node is added for bias)
- Number of hidden layer: 1 (Heaton, 2017)
- Number of hidden layer nodes: 27 (Heaton, 2017), with ReLU activation function
- Number of nodes in output layer: 1, with Sigmoid activation function (binary classification)
- Epochs: 11 (*Epoch : An essential notion in real-time programming*, 2023)
- Batch size: 32 (Yoshua, 2012)

For our hyperparameter tuning, we tuned the number of nodes in input layer (35,40 and 45), number of hidden layer nodes (25,27,30), and number of epochs (5,10,11,15 and 20). These parameters were selected based on their proximity to the baseline values, with slight adjustments upwards and downwards.

The number of batch size was not configured as adjusting it could significantly affect the training and convergence behaviour of the model. Since our primary focus was on exploring the impact of changes from other key hyperparameters, such as the number of nodes and epochs, we maintained a consistent batch size to ensure comparability and stability across different experiments. Additionally, keeping the batch size constant reduced the complexity of parameter combinations under consideration.

Due to the constraints in computational power and the increased computational demands of our custom grid search function, which incorporates 5-fold cross-validation, our parameter space for the DNN model was restricted. As a result, we were unable to explore a wider range of hyperparameters during the tuning process.

6. Results & Discussions

6.1 Results

6.1.1 Baseline models

Model	Hyperparameters used	Test ROC-AUC (3 decimal places)
Logistic Regression (without interaction terms)	• Inverse of strength of regularization (c): 1.0 • Penalty: ‘l2’ • Solver: ‘lbfgs’ • Maximum iterations for solvers to merge: 100	0.757
Logistic Regression (with interaction terms)	• Inverse of strength of regularization (c): 1.0 • Penalty: ‘l2’ • Solver: ‘lbfgs’ • Maximum iterations for solvers to merge: 100	0.756
SVM	• Strength of the regularization (c): 1.0 • Max iteration: 1000 • Dual formulation of linear SVM: True	0.686
Random Forest	• Maximum depth: None • Max features: sqrt(features) • Minimum samples to split an internal node: 2 • Number of trees: 100 • Criterion: Gini	0.756
XGBoost	• Maximum depth: 3 • Minimum sum of child weight: 1 • Subsample ratio per boost: 1 • Subsample ratio of columns per boost: 1 • Learning rate: 0.03 • Booster: gbtree • Objective: binary:logistic	0.741

DNN	• Input layer nodes: 40 • Epochs: 11 • Number of hidden layers: 1 • Hidden layer nodes: 27	0.752

Table 6.1 Table of baseline models and hyperparameters used, and Test ROC-AUC score

6.1.2 Hyperparameter-tuned models

Model	Optimal Hyperparameter	Test ROC-AUC (3 decimal places)
Logistic Regression without Interaction Terms (Baseline) [4]	Default parameters by sklearn.linear_model.LogisticRegression • Inverse of strength of regularization (c): 1.0 • Penalty: ‘l2’ • Solver: ‘lbfgs’ • Maximum iterations for solvers to merge: 100	0.757
SVM	• Strength of the regularization (c): 1.3 • Max iteration: 500 • Dual formulation of linear SVM: False	0.686
Random Forest	• Maximum depth: 5 • Max features: log2(features) • Minimum samples to split an internal node: 2 • Number of trees: 26 • Criterion: gini	0.762
XGBoost	• Number of booster rounds: 134 • Lambda: 0.00001 • Learning rate: 0.015 • Booster: gblinear • Objective: binary:logistic	0.768

[4] Intermediate results can be found in the Appendix (Section 9.2)

DNN	• Input layer nodes: 35 • Epochs: 5 • Number of hidden layers: 1 • Hidden layer nodes: 27	0.753

Table 6.2 Table of hyperparameter-tuned models, optimal hyperparameters and the Test ROC-AUC score

6.1.3 Metrics of best performing model – XGBoost

Metrics	Test Score (3.d.p)
ROC-AUC	0.768
Accuracy	0.700
Precision	0.645
Recall	0.736
F1 Score	0.687

Table 6.3 Table of Performance Metrics Summary for XGBoost Model (49% Classification Threshold)

After performing hyperparameter tuning, the best model found was the XGBoost Model with the binary classification metric scores achieved as seen in Table 6.3. We believe that the XGBoost model outperformed all other models due to the following reasons:

- **Capturing non-linear relationships**
  In return prediction models, there is a need to often capture complex, non-linear relationships between the features and likelihood of returns. XGBoost excels in capturing intricate, non-linear relationships due to its capability to model complex interactions between features.
- **Model Performance**
  XGBoost is known for its high performance and efficiency, making it suitable for handling large-scale datasets commonly encountered in return prediction problems.
- **Effective Regularization Techniques**
  XGBoost incorporates regularisation techniques, such as controlling the L2 penalty, which can curb overfitting and enhance the model’s generalisation ability.

Furthermore, we found that the XGBoost model with linear model performs better than a XGBoost model with a tree because:

- **Linear relationships**
  It might be possible that the relationship of the more important features and target variable are linearly related
- **Overfitting**
  Though tree models are powerful and able to model complex relationships, they also tend to suffer from overfitting, especially when dealing with high-dimensional datasets with complex interactions between features.

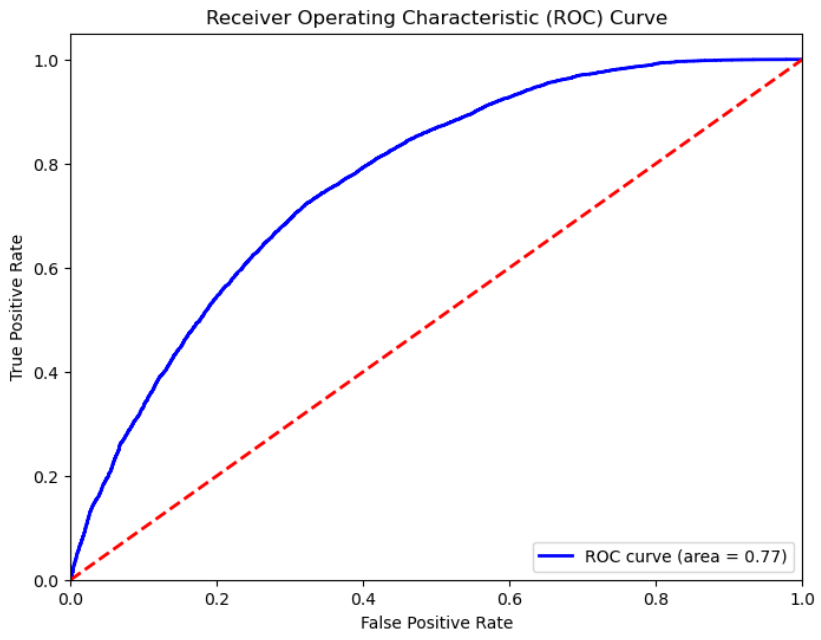


Figure 6.4 ROC Curve for XGBoost Model

The ROC curve generated from our model evaluation provides valuable insights into the trade-off between true positive rate and false positive rate across various classification thresholds, illustrating the model's performance in distinguishing between positive and negative instances. A steeper ROC curve indicates superior discriminative performance, with the area under the curve (AUC) serving as a quantitative measure of the model's overall performance. The higher the AUC, the better the model is at predicting positive and negative classes correctly. (Narkhede, 2018)

With an AUC score of 0.77 significantly higher than the random guessing threshold of 0.5 (where a model has no discrimination capacity to distinguish between positive and negative classes), our XGBoost model exhibit a strong ability to correctly classify instances.

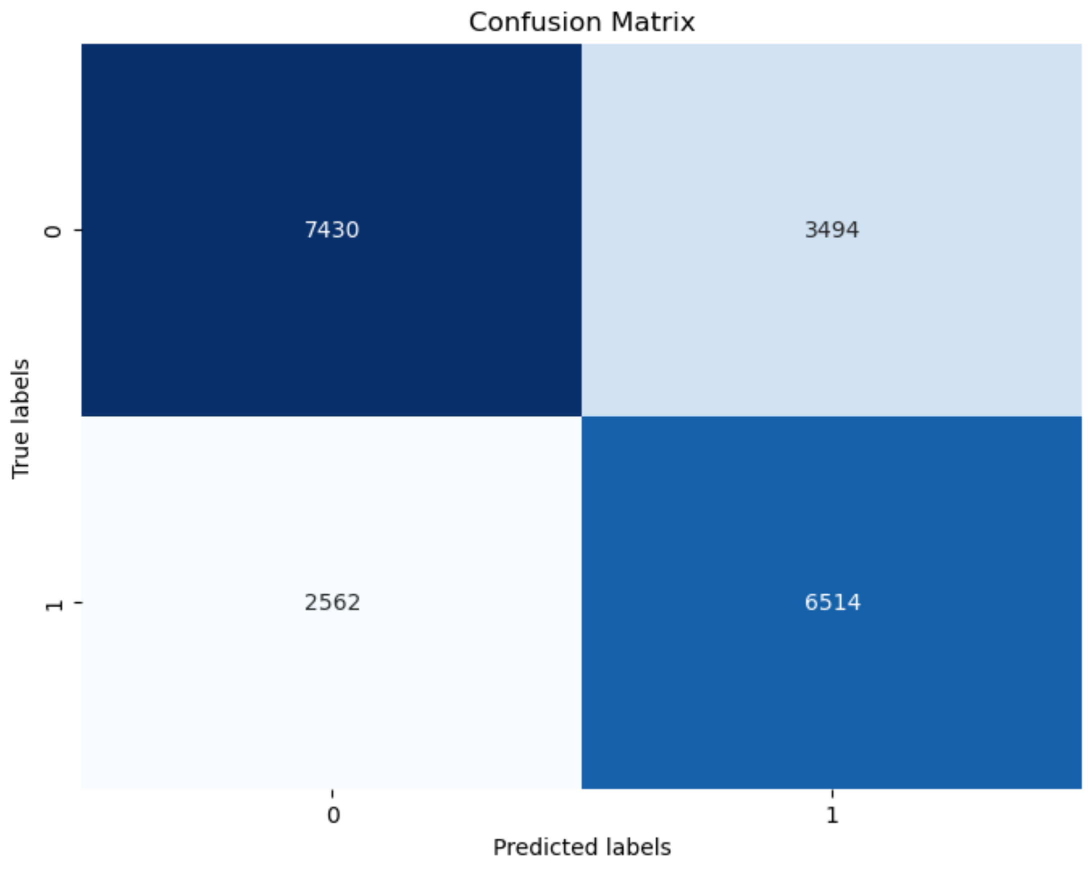


Figure 6.5 Confusion Matrix for XGBoost Model (49% Classification Threshold)

At a 49% classification threshold, our confusion matrix shows that our model demonstrates strong performance in accurately predicting both true positives and negatives. The model also exhibits a low incidence of false negatives, indicating its effectiveness in correctly identifying instances of returns.

However, it is observed that there is a moderate number of false positives, suggesting instances where the model predicts returns that do not actually happen. Despite this, we feel that a higher false positive rate (overestimation of returns) is acceptable for a returns forecasting model because it allows the retailers to proactively prepare for potential returns, thereby enhancing customer satisfaction by ensuring smoother return processes.

Going back to what was discussed in 3.6, if maximising recall is a priority, then using a 45% classification threshold is also feasible. This results in an approximate 2% loss in precision but a 7% gain in recall.

Metrics	**Test Score (3.d.p)**
ROC-AUC	0.768
Accuracy	0.697
Precision	0.621
Recall	0.799
F1 Score	0.698

Table 6.6 Table of Performance Metrics Summary for XGBoost Model (45% Classification Threshold)

6.2 Limitations

There are several limitations that impact our models' effectiveness and generalizability. Firstly, we encountered challenges related to high computational power requirements. Given the complexity of the machine learning models and tuning processes involved, such as deep neural networks and cross validation with grid search, it becomes difficult to thoroughly explore hyperparameters. This limitation restricts the extent to which the models' performances can be optimized.

Secondly, the dataset used in our project primarily consists of time-centric information. While our pre-processing has provided valuable insights into customer behaviour, it may not capture all relevant temporal factors influencing return rates. The lack of more time-centric features could limit the models' ability to accurately predict returns, particularly if important predictors are not adequately represented. Furthermore, if we had access to data over a longer period, we could then incorporate historical values to enable a deeper understanding of retailer, brand, and user habits over time. Additionally, leveraging historical data allows us to capture evolving trends and preferences, enabling dynamic adjustments in predictive modelling strategies to better align with changing consumer behaviours.

Thirdly, the dataset used in our project is sourced exclusively from a single fashion retailer in Germany. This restriction to a single source introduces potential biases and limits the model's generalizability to other geographical regions or online retailers. As a result, the model may not perform optimally when applied to datasets from different countries or sectors, thereby restricting its broader applicability. Furthermore, larger datasets may or may not impact our approach to classification (pre-processing, retailer-specific thresholds, number of dimensions, best model used, etc.).

7. Conclusion & Future Work

In conclusion, XGBoost model was the best performing model with an AUC score of 0.768, comparable to scores found in our literature review and strikes a good balance between precision and recall unlike other online fashion retailer returns classification models in the market. Our approach to manual dimensionality reduction also allowed us to get a better true sensing of the important variables for this retailer as compared to the other models where interpretation of the variables is lost during the black box process.

7.1 Feature Importance

Important features found by the XGBoost model (in order of absolute weightage) are no_delivery, least_returned_item, least_returned_color, item_size_child, item_size_unsized, user_state_Bremen, *user_return_rate, most_returned_item, log_item_price, log_no_items, most_returned_color, log_avg_freq_purchases, is_bday.*

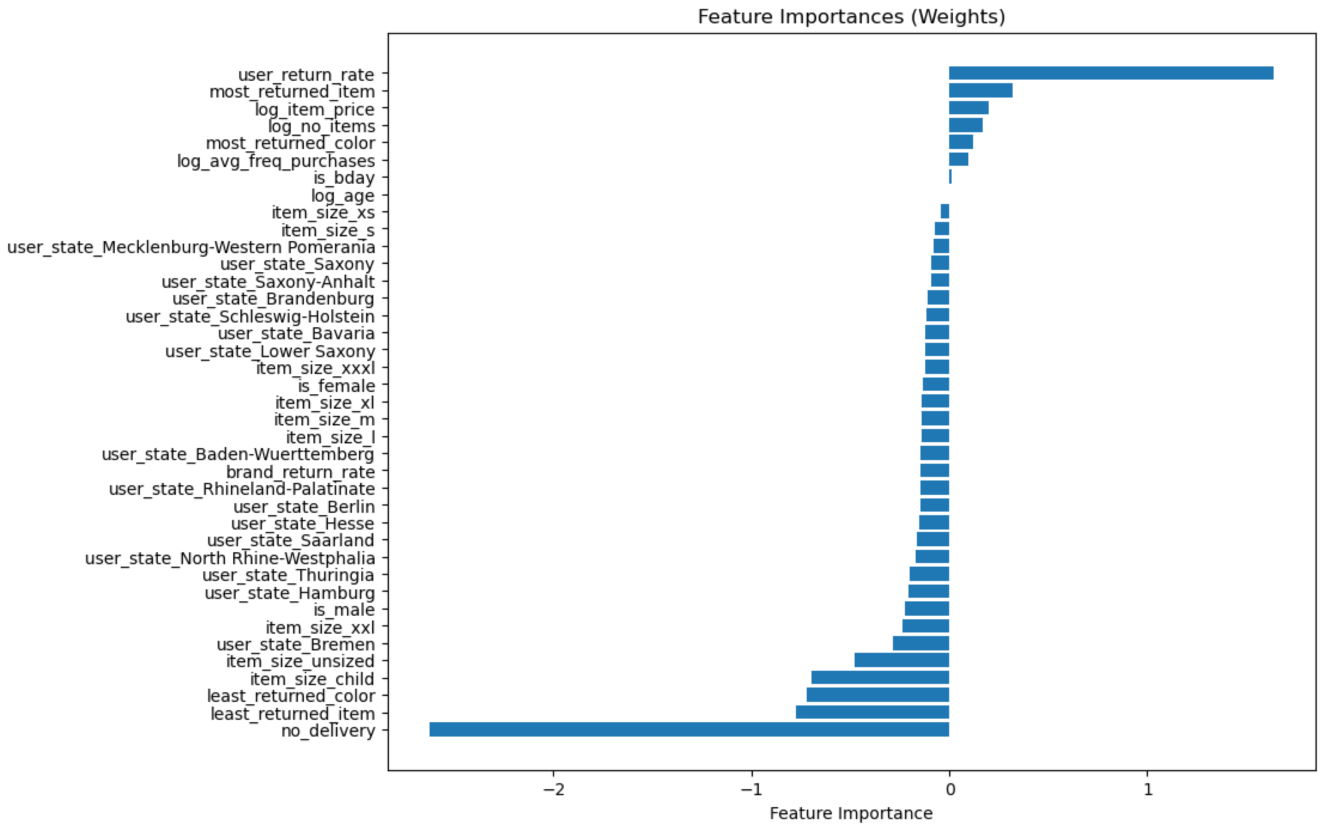


Figure 7.1 Best performing model: XGBOOST feature importance by weight

Based on these important features, we have a set of recommendations for this online fashion retailer. which may help improve (by lowering) their customer clothing return rates in Figure 8.2.

Important features with a positive weight (normalised coefficients) are focused on as they contribute to the likelihood of an item being returned, whereas negative weighted features contribute to the likelihood of an item not being returned.

Further exploration can be done by splitting customer clothing return data by the above features individually then clustering customers to get a more detailed picture of product returns habits of the various clusters (segments) of customers. The XGBoost model would then be tested over the different clusters to see if predictions deviate too much from the original baseline, giving the business an opportunity to further fine tune their approach based on the segment of shopper they are dealing with. It is also possible that these clusters could be used to further improve predictions.

Based on the identified features, we can then derive a set of recommendations for retailers to appropriately address our problem statement of reducing profit losses in terms of logistical costs and product wastage, as well as heighten the customer experience. For instance, we understand that certain product characteristics such as price are significant factors which influence retail returns. In this case, retailers could divert more effort towards testing and customer surveys to gather insights on finding a balance between price and quality that meets their customer expectations (items that are too cheap and of low quality may have a higher likelihood of being returned).

We can also analyse purchases characteristics such as whether customers have the tendency to bundle their items and evaluate the average frequency of purchases to examine commonly purchased baskets of products to better identify demand trends. This data can then be used to develop better inventory management strategies to avoid stockouts. From a product perspective, retailers can also investigate which products are most returned and make improvements from there. For instance, items could be compared based on their fit/style to find consumer patterns.

The following table highlights a comprehensive list of recommendations derived from each feature, and their applications.

Important feature(s)	Recommendation(s)
user_return_rate	Perform further analysis such as clustering by rate (binned by thresholds relevant to the business)
most_returned_item	<u>Reapply lessons learned from successful factors</u> • Analyze least-returned items to identify successful design elements in least_returned_item • Apply learnings to most_returned_item
log_item_price	<u>Customer Feedback on more expensive items</u> • Conduct testing and customer surveys. • Gather insights on areas for improvement. • Use feedback to enhance product quality and customer satisfaction.
log_no_items	<u>Bundle Discounts</u> • Provide incentives for purchasing multiple items together. • Encourage more deliberate purchases and increase transaction value.
most_returned_color	<u>Market Analysis</u> Examine the pattern of returns for that color • Customer taste and preference for color profiles • Expected vs Actual color • Quality of clothes

	• Discoloration • Staining
log_avg_freq_purchases	<u>Purchase Analysis</u> • Examine average frequent purchases and item combinations. • Identify trends in the number of items and sizes bought together. • Utilize data to inform inventory and marketing strategies.
is_bday	<u>Purchase Analysis</u> • Identify lead time before birthday, the purchases were made • Examine what the customers purchases returned were • Identify possible root causes for returning item on birthday o e.g. customer bought many different clothes for birthday celebration but decided to return outfits that they did not like

Table 7.2 Recommendations to online fashion retailer

7.2 Extended Analysis on Dimensionality Reduction

Dimensionality reduction is a process and technique to reduce the number of dimensions or features in a date set. The goal is to reduce amount of time and memory required by data mining algorithms by filtering out redundant and correlated features.
For our approach, we decided to take a manual dimensionality reduction approach to see how effective it could be compared to more traditional approaches where unsupervised learning algorithms are used.

However, if we wish to increase generalizability of our model (e.g. by combining more data sources from different retailers), it is possible that we would have to utilise dimensionality reduction techniques on the whole/partial part of the dataset to extract out more valuable attributes that will give us the most optimal results for the findings of our project.

7.2.1 Utilising Unsupervised Learning Dimensionality Reduction Techniques

To prepare for future works, we applied three dimensionality reduction algorithms to our top-performing model, XGBoost, to assess their impact on performance metrics. The results and analysis are shown below:

Dimensionality Reduction Technique	**Description**	**Result**
Principle Component Analysis (PCA)	Linear dimensionality reduction to reduce dataset dimensions by seeking uncorrelated components	Components: 7 AUC: 0.723
Independent Component Analysis (ICA)	Non-linear dimensionality reduction by finding statistically independent components	Components: 7 AUC: 0.735
Uniform Manifold Approximation and Projection (UMAP)	Non-linear dimensionality reduction by constructing low-dimensional representation of data	Components: 7 AUC: 0.512

Table 7.3 Summary of dimensionality reductions techniques and corresponding results on our dataset and model

The initial baseline model of XGBoost had an AUC of 0.768 using 39 columns. Following the implementation of the three dimensionality reduction techniques, we observed that ICA produced the best result among the rest of AUC 0.735. The result however seems to be lower than the baseline model of XGBoost but considering the number of components has been reduced to 7 from the original 39, which is less than 20% of the initial features, we have achieved a good, closely approaching baseline result.

Therefore, we conclude that ICA has the best performing result and should be chosen as the technique to use for future dimensionality reduction technique for this dataset.

If we were to expand our model with more data (such as data from multiple countries) or a dataset with variables more than what we used, then we would need to leverage on dimensionality reduction technique to handle the increased complexity and size of the dataset efficiently, mitigating issues such as overfitting, computational problems and the curse of dimensionality.

7.2.2 Using only XGBoost Most Important Features

We also tried re-training an XGBoost model using our best models' top 10 most important features. This re-trained and tuned model had an AUC of 0.755, which represents a worthwhile trade-off of an approximate 1% decrease in AUC for a 75% decrease in the number of input features.

Overall, this shows us that (at least in the case of this data set) it is more worthwhile to perform manual feature selection if we need to simplify the model due to whatever reason.

8. References

Alisa_K, Johannes. (2019). *BADS1920*. Kaggle. https://kaggle.com/competitions/bads1920

Alterations Express. (2023, June 22). *How to read pants size*. wikiHow. https://www.wikihow.com/Read-Pants-Size

Baby & Kids Clothes. Mothercare Online. (n.d.). https://www.mothercare.com.sg/size-charts/baby--kids-fashion/

Bergstra, J., & Bengio, Y. (2012). *Random search for hyper-parameter optimization*. Journal of Machine Learning Research, 13, 281-305. https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Bouniot, Q., Redko, I., Mallasto, A., Laclau, C., Arndt, K., Struckmeier, O., Heinonen, M., Kyrki, V., & Kaski, S. (2023, October 17). *Understanding deep neural networks through the lens of their non-linearity*. Cornell University. https://arxiv.org/abs/2310.11439#:~:text=Indeed%2C%20DNNs%20are%20highly%20non,are%20largely%20responsible%20for%20this.

Boston Consulting Group. (2023, October 31). *E-commerce poised to capture 41% of global retail sales by 2027—up from just 18% in 2017*. Boston Consulting Group. https://www.bcg.com/press/31october2023-ecommerce-global-retail-sales

DataScientest. (2023, June 2). *Epoch : An essential notion in real-time programming*. DataScientest. https://datascientest.com/en/epoch-an-essential-notion#:~:text=A%20larger%20number%20of%20epochs,to%20optimally%20modify%20the%20weights

GeeksforGeeks. (2023, January 10). *Advantages and disadvantages of logistic regression*. GeeksforGeeks. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/

GeeksforGeeks. (2023, January 23). *Dual support vector machine*. GeeksforGeeks. https://www.geeksforgeeks.org/dual-support-vector-machine/

Hailong Cui, Sampath Rajagopalan, Amy R. Ward (2020). P*redicting product return volume using machine learning methods*. European Journal of Operational Research. Volume 281, Issue 3,2020, Pages 612-627, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2019.05.046.

Heaton, J. (2017, June 1). *The number of hidden layers.* Heaton Research. https://www.heatonresearch.com/2017/06/01/hidden-layers.html

IBM. (n.d.). *What is deep learning?* IBM. https://www.ibm.com/topics/deep-learning

Kedia, S., Madan, M., & Borar, S. (2019). Early bird catches the worm: Predicting returns even before purchase in fashion e-commerce | semantic scholar. https://arxiv.org/pdf/1906.12128.pdf

Melanie. (2024, February 27). *Calculate correlation between two variables: How do you measure dependence?* DataScientest. https://datascientest.com/en/calculate-correlation-between-two-variables-how-do-you-measure-dependence

Narkhede, S. (2018a, June 27). *Understanding AUC - roc curve*. Medium. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

nShift. (2023, June 27). *Returns cost retailers $600 billion a year*. nShift. https://nshift.com/press/returns-cost-retailers-600-billion-a-year

Repko, M. (2022, January 25). *A more than $761 billion dilemma: Retailers' returns Jump as online sales grow*. CNBC. https://www.cnbc.com/2022/01/25/retailers-average-return-rate-jumps-to-16point6percent-as-online-sales-grow-.html

Rucha Yancha LLP. (2023, June 29). *Understanding deep learning: Exploring its advantages and disadvantages*. Rucha Yancha LLP. https://www.yantrallp.com/blog/understanding-deep-learning-exploring-its-advantages-and disadvantages/#:~:text=Ability%20to%20Learn%20Complex%20Patterns,and%20automatically%20extract%20meaningful%20features.

Scikit-Learn. (2024). *4.1. Partial dependence and individual conditional expectation plots*. scikit-learn. https://scikit-learn.org/stable/modules/partial_dependence.html#partial-dependence-plots

Scikit-Learn. (n.d.). *Sklearn.ensemble.randomforestclassifier*. scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Udilâ, et al. (2023, June 25). *Encoding Methods for Categorical Data: A Comparative Analysis for Linear Models, Decision Trees, and Support Vector Machines*. Repository TU Delft. https://repository.tudelft.nl/islandora/object/uuid:10b91b99-2685-4a45-b44e-48fbbf808ce2/datastream/OBJ/download#:~:text=Target%20encoding%2C%20also%20known%20as

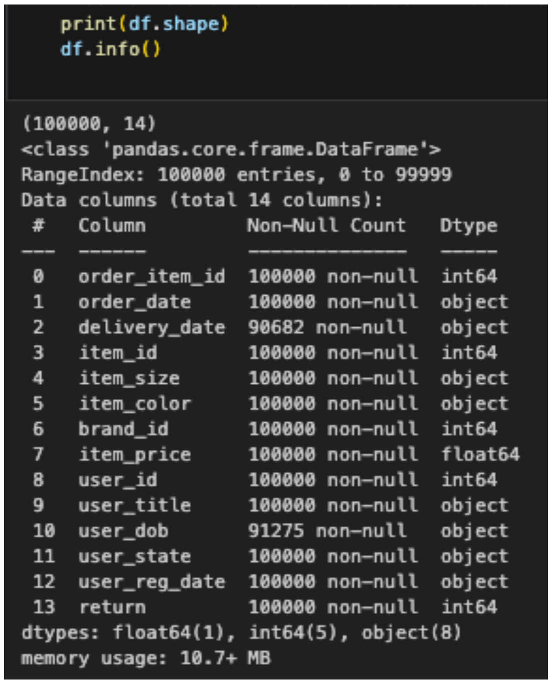
Urbanke, P., Kranz, J., & Kolbe, L. (2015, December). Predicting product returns in e-commerce: The contribution of Mahalanobis feature extraction completed research paper. https://www.researchgate.net/publication/283271154_Predicting_Product_Returns_in_E-Commerce_The_Contribution_of_Mahalanobis_Feature_Extraction_Completed_Research_Paper

XGBoost. (2024). *Python API reference — xgboost 2.1.0-dev documentation*. XGBoost. https://xgboost.readthedocs.io/en/latest/python/python_api.html#xgboost.Booster.get_score

Yoshua, B. (2012, September 16). *Practical Recommendations for Gradient-Based Training of Deep Arthitectures*. https://arxiv.org/pdf/1206.5533.pdf

9. Appendix

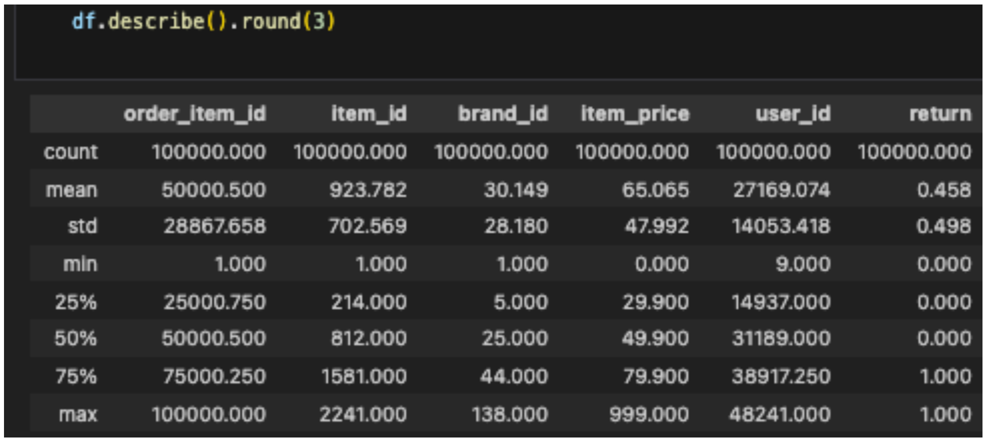
9.1 Data Pre-Processing



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

```
(100000, 14)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   order_item_id  100000 non-null  int64  
 1   order_date     100000 non-null  object 
 2   delivery_date  90682 non-null   object 
 3   item_id        100000 non-null  int64  
 4   item_size      100000 non-null  object 
 5   item_color     100000 non-null  object 
 6   brand_id       100000 non-null  int64  
 7   item_price     100000 non-null  float64
 8   user_id        100000 non-null  int64  
 9   user_title     100000 non-null  object 
 10  user_dob       91275 non-null   object 
 11  user_state     100000 non-null  object 
 12  user_reg_date  100000 non-null  object 
 13  return         100000 non-null  int64  
dtypes: float64(1), int64(5), object(8)
memory usage: 10.7+ MB
```

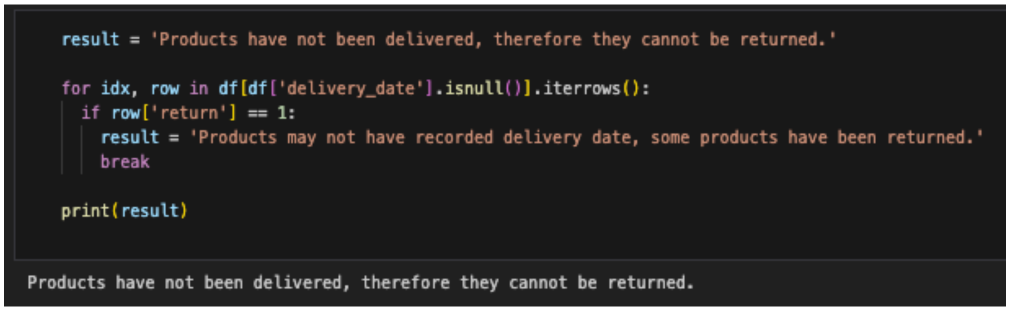
Figure 5.1 Dimensions of our data



```
df.describe().round(3)
```

	order_item_id	item_id	brand_id	item_price	user_id	return
count	100000.000	100000.000	100000.000	100000.000	100000.000	100000.000
mean	50000.500	923.782	30.149	65.065	27169.074	0.458
std	28867.658	702.569	28.180	47.992	14053.418	0.498
min	1.000	1.000	1.000	0.000	9.000	0.000
25%	25000.750	214.000	5.000	29.900	14937.000	0.000
50%	50000.500	812.000	25.000	49.900	31189.000	0.000
75%	75000.250	1581.000	44.000	79.900	38917.250	1.000
max	100000.000	2241.000	138.000	999.000	48241.000	1.000

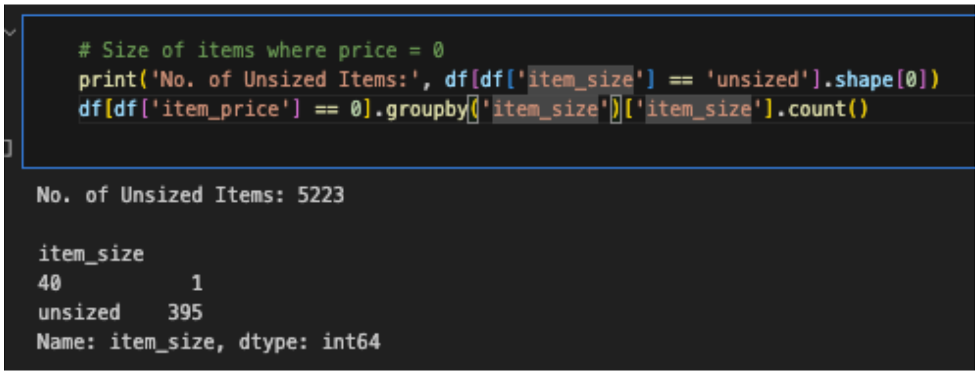
Figure 5.2 Descriptive stats of our data



```
result = 'Products have not been delivered, therefore they cannot be returned.'

for idx, row in df[df['delivery_date'].isnull()].iterrows():
  if row['return'] == 1:
    result = 'Products may not have recorded delivery date, some products have been returned.'
    break

print(result)
```

```
Products have not been delivered, therefore they cannot be returned.
```

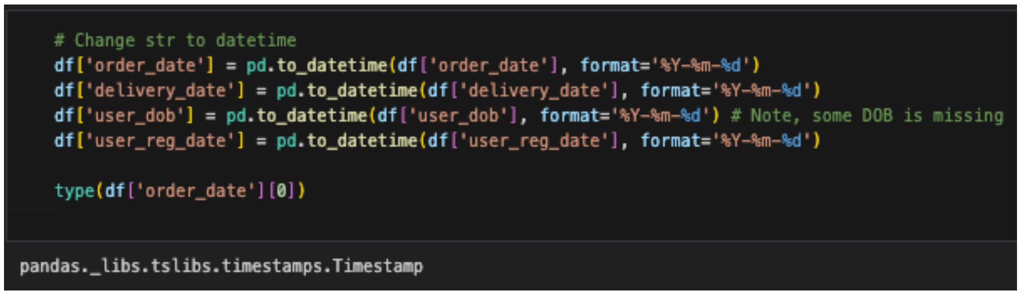
Figure 5.3 Analysing missing delivery dates



```
# Size of items where price = 0
print('No. of Unsized Items:', df[df['item_size'] == 'unsized'].shape[0])
df[df['item_price'] == 0].groupby('item_size')['item_size'].count()
```

```
No. of Unsized Items: 5223

item_size
40            1
unsized     395
Name: item_size, dtype: int64
```

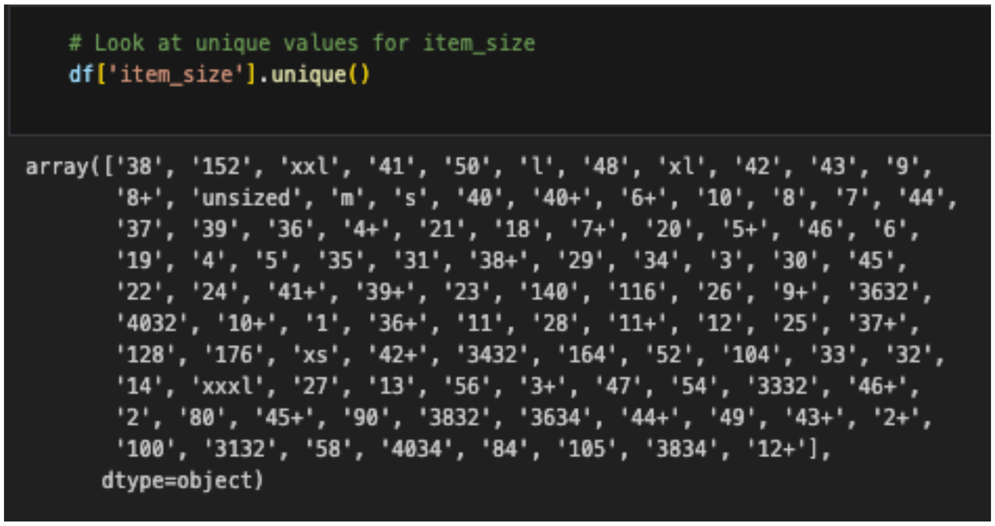
Figure 5.4 Analysing item price



```
# Change str to datetime
df['order_date'] = pd.to_datetime(df['order_date'], format='%Y-%m-%d')
df['delivery_date'] = pd.to_datetime(df['delivery_date'], format='%Y-%m-%d')
df['user_dob'] = pd.to_datetime(df['user_dob'], format='%Y-%m-%d') # Note, some DOB is missing
df['user_reg_date'] = pd.to_datetime(df['user_reg_date'], format='%Y-%m-%d')

type(df['order_date'][0])
```

```
pandas._libs.tslibs.timestamps.Timestamp
```

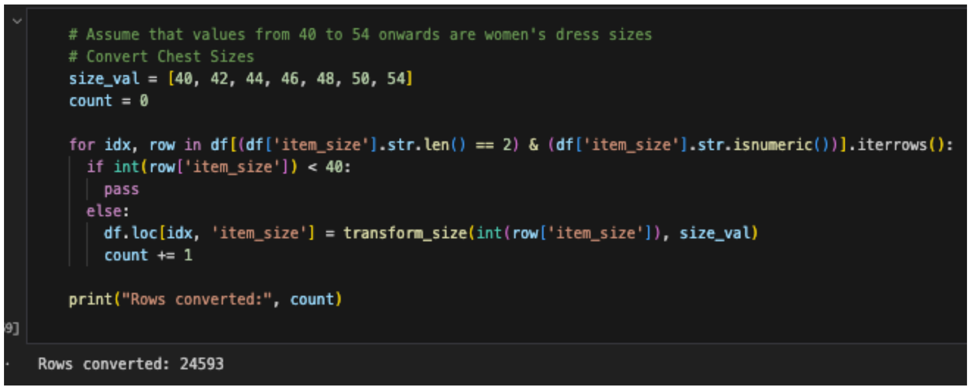
Figure 5.5 Converting into date datatype



```
# Look at unique values for item_size
df['item_size'].unique()
```

```
array(['38', '152', 'xxl', '41', '50', 'l', '48', 'xl', '42', '43', '9',
       '8+', 'unsized', 'm', 's', '40', '40+', '6+', '10', '8', '7', '44',
       '37', '39', '36', '4+', '21', '18', '7+', '20', '5+', '46', '6',
       '19', '4', '5', '35', '31', '38+', '29', '34', '3', '30', '45',
       '22', '24', '41+', '39+', '23', '140', '116', '26', '9+', '3632',
       '4032', '10+', '1', '36+', '11', '28', '11+', '12', '25', '37+',
       '128', '176', 'xs', '42+', '3432', '164', '52', '104', '33', '32',
       '14', 'xxxl', '27', '13', '56', '3+', '47', '54', '3332', '46+',
       '2', '80', '45+', '90', '3832', '3634', '44+', '49', '43+', '2+',
       '100', '3132', '58', '4034', '84', '105', '3834', '12+'],
      dtype=object)
```

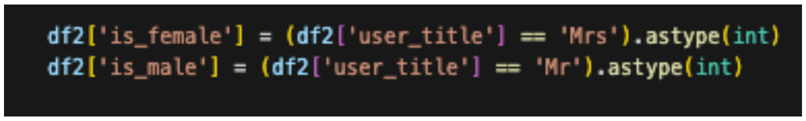
Figure 5.6 List of unique values for item size



```
# Assume that values from 40 to 54 onwards are women's dress sizes
# Convert Chest Sizes
size_val = [40, 42, 44, 46, 48, 50, 54]
count = 0

for idx, row in df[(df['item_size'].str.len() == 2) & (df['item_size'].str.isnumeric())].iterrows():
  if int(row['item_size']) < 40:
    pass
  else:
    df.loc[idx, 'item_size'] = transform_size(int(row['item_size']), size_val)
    count += 1

print("Rows converted:", count)
```

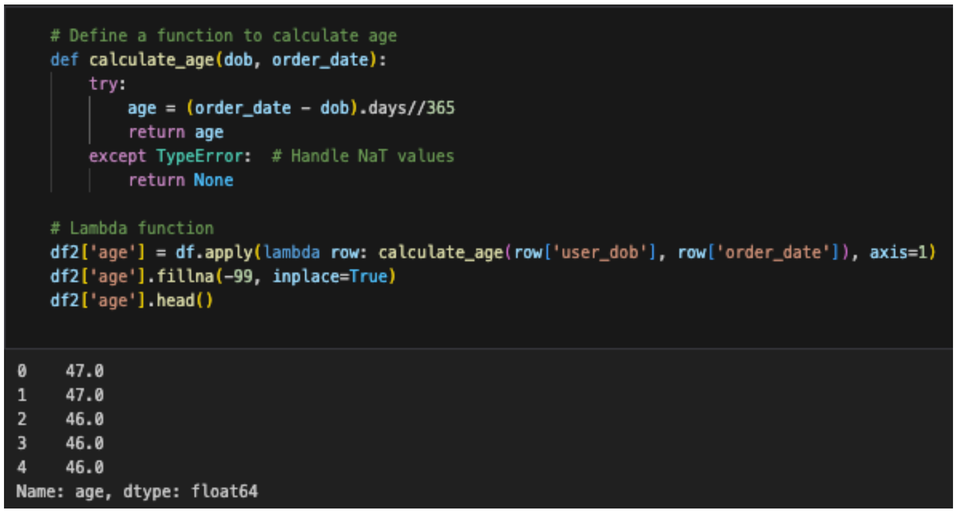
```
Rows converted: 24593
```

Figure 5.7 Example of mapping integer size to descriptive string size



```
df2['is_female'] = (df2['user_title'] == 'Mrs').astype(int)
df2['is_male'] = (df2['user_title'] == 'Mr').astype(int)
```

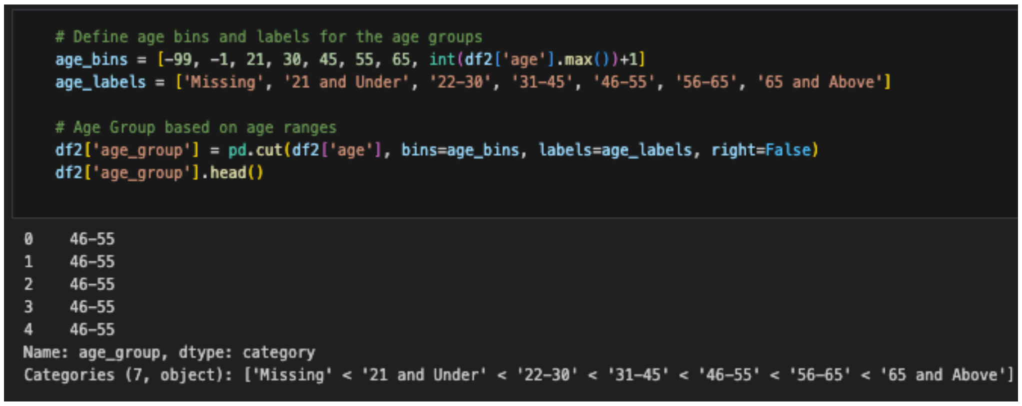
Figure 5.8 Creation of gender attributes



```python
# Define a function to calculate age
def calculate_age(dob, order_date):
    try:
        age = (order_date - dob).days//365
        return age
    except TypeError:  # Handle NaT values
        return None

# Lambda function
df2['age'] = df.apply(lambda row: calculate_age(row['user_dob'], row['order_date']), axis=1)
df2['age'].fillna(-99, inplace=True)
df2['age'].head()
```

```
0    47.0
1    47.0
2    46.0
3    46.0
4    46.0
Name: age, dtype: float64
```

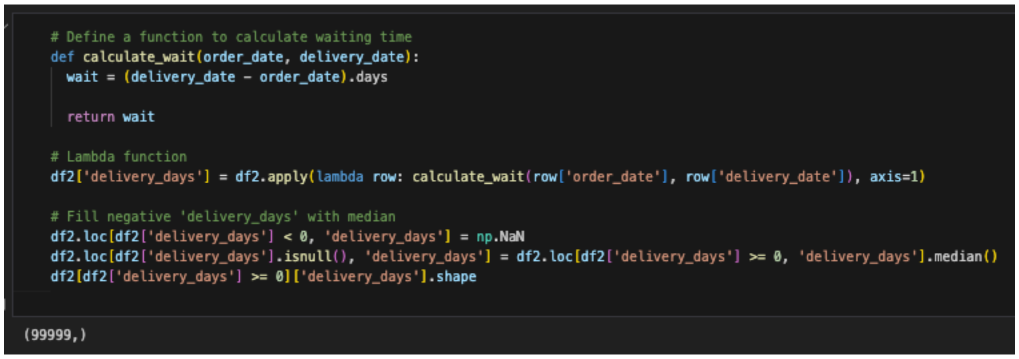
Figure 5.9 Creation of age attribute



```python
# Define age bins and labels for the age groups
age_bins = [-99, -1, 21, 30, 45, 55, 65, int(df2['age'].max())+1]
age_labels = ['Missing', '21 and Under', '22-30', '31-45', '46-55', '56-65', '65 and Above']

# Age Group based on age ranges
df2['age_group'] = pd.cut(df2['age'], bins=age_bins, labels=age_labels, right=False)
df2['age_group'].head()
```

```
0    46-55
1    46-55
2    46-55
3    46-55
4    46-55
Name: age_group, dtype: category
Categories (7, object): ['Missing' < '21 and Under' < '22-30' < '31-45' < '46-55' < '56-65' < '65 and Above']
```

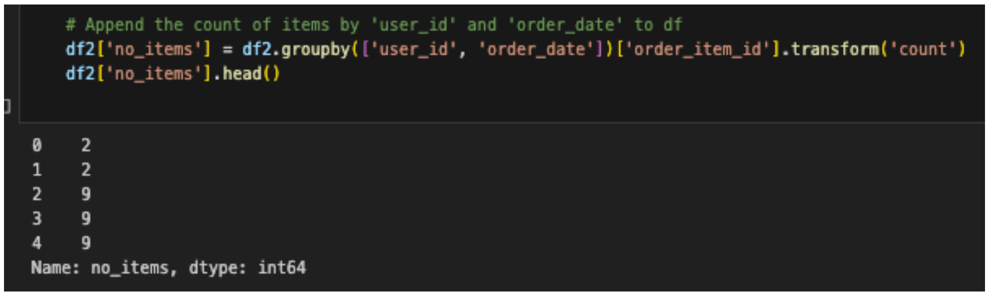
Figure 5.10 Creation of age group



```python
# Define a function to calculate waiting time
def calculate_wait(order_date, delivery_date):
    wait = (delivery_date - order_date).days

    return wait

# Lambda function
df2['delivery_days'] = df2.apply(lambda row: calculate_wait(row['order_date'], row['delivery_date']), axis=1)

# Fill negative 'delivery_days' with median
df2.loc[df2['delivery_days'] < 0, 'delivery_days'] = np.NaN
df2.loc[df2['delivery_days'].isnull(), 'delivery_days'] = df2.loc[df2['delivery_days'] >= 0, 'delivery_days'].median()
df2[df2['delivery_days'] >= 0]['delivery_days'].shape
```

```
(99999,)
```

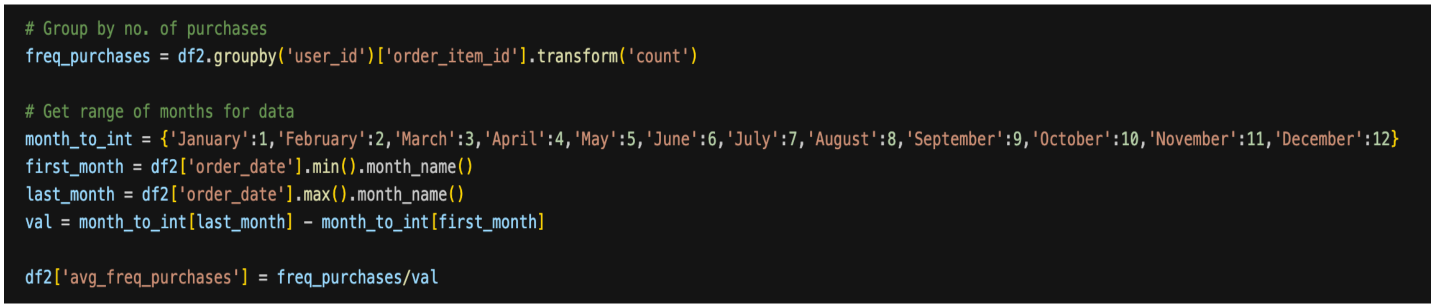
Figure 5.11 Creation of delivery days and imputation of negative waiting time with median waiting time



```
# Append the count of items by 'user_id' and 'order_date' to df
df2['no_items'] = df2.groupby(['user_id', 'order_date'])['order_item_id'].transform('count')
df2['no_items'].head()
```

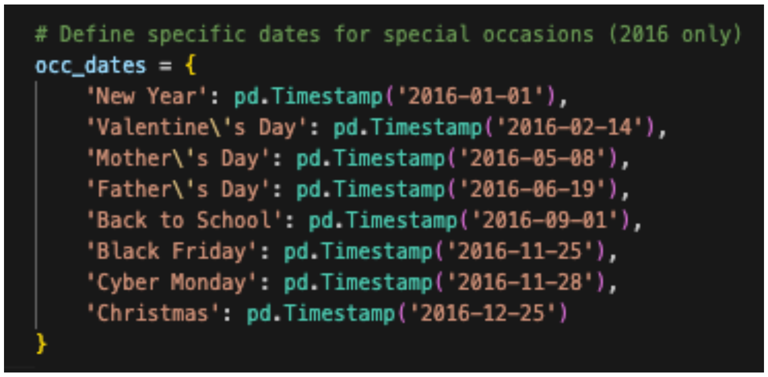
```
0    2
1    2
2    9
3    9
4    9
Name: no_items, dtype: int64
```

Figure 5.12 Creation of attribute "no_items"



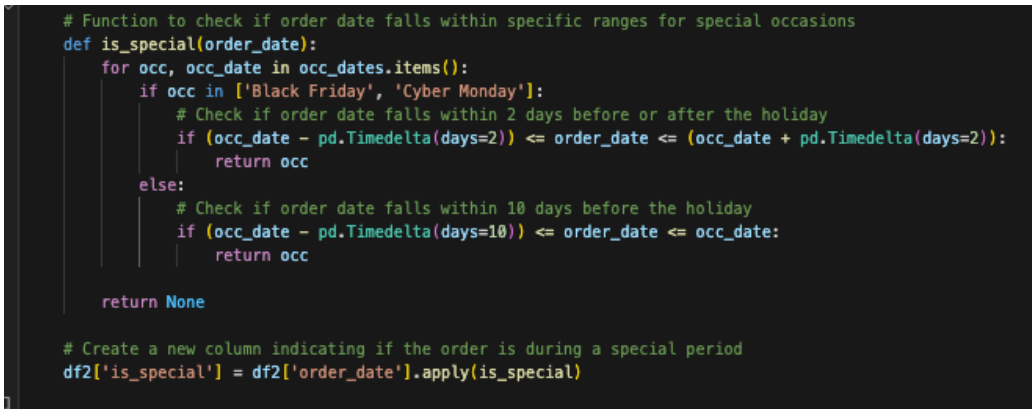
```
# Group by no. of purchases
freq_purchases = df2.groupby('user_id')['order_item_id'].transform('count')

# Get range of months for data
month_to_int = {'January':1,'February':2,'March':3,'April':4,'May':5,'June':6,'July':7,'August':8,'September':9,'October':10,'November':11,'December':12}
first_month = df2['order_date'].min().month_name()
last_month = df2['order_date'].max().month_name()
val = month_to_int[last_month] - month_to_int[first_month]

df2['avg_freq_purchases'] = freq_purchases/val
```

Figure 5.13 Creation of attribute "avg_freq_purchases"



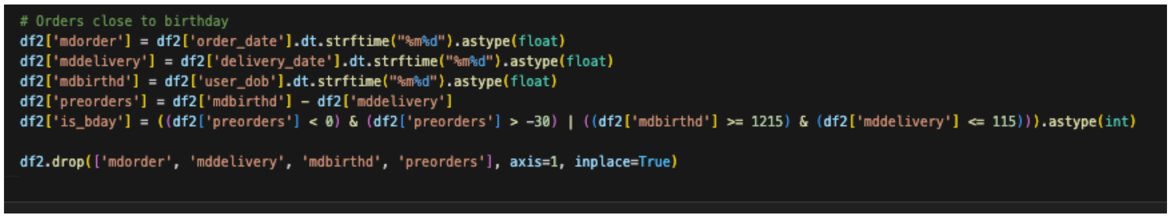
```
# Define specific dates for special occasions (2016 only)
occ_dates = {
    'New Year': pd.Timestamp('2016-01-01'),
    'Valentine\'s Day': pd.Timestamp('2016-02-14'),
    'Mother\'s Day': pd.Timestamp('2016-05-08'),
    'Father\'s Day': pd.Timestamp('2016-06-19'),
    'Back to School': pd.Timestamp('2016-09-01'),
    'Black Friday': pd.Timestamp('2016-11-25'),
    'Cyber Monday': pd.Timestamp('2016-11-28'),
    'Christmas': pd.Timestamp('2016-12-25')
}
```

5.14 Defining special occasions



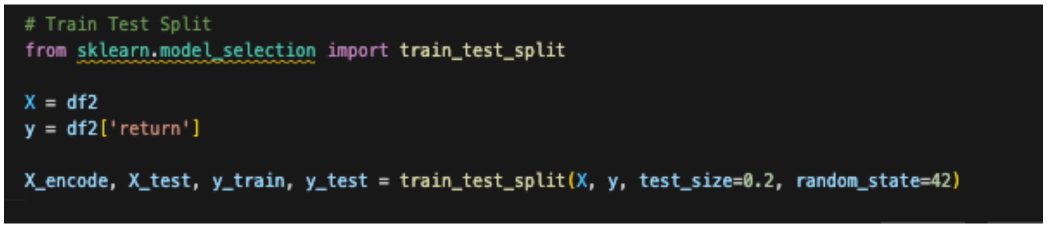
```python
# Function to check if order date falls within specific ranges for special occasions
def is_special(order_date):
    for occ, occ_date in occ_dates.items():
        if occ in ['Black Friday', 'Cyber Monday']:
            # Check if order date falls within 2 days before or after the holiday
            if (occ_date - pd.Timedelta(days=2)) <= order_date <= (occ_date + pd.Timedelta(days=2)):
                return occ
        else:
            # Check if order date falls within 10 days before the holiday
            if (occ_date - pd.Timedelta(days=10)) <= order_date <= occ_date:
                return occ

    return None

# Create a new column indicating if the order is during a special period
df2['is_special'] = df2['order_date'].apply(is_special)
```

Figure 5.15 Creation of attribute "is_special"



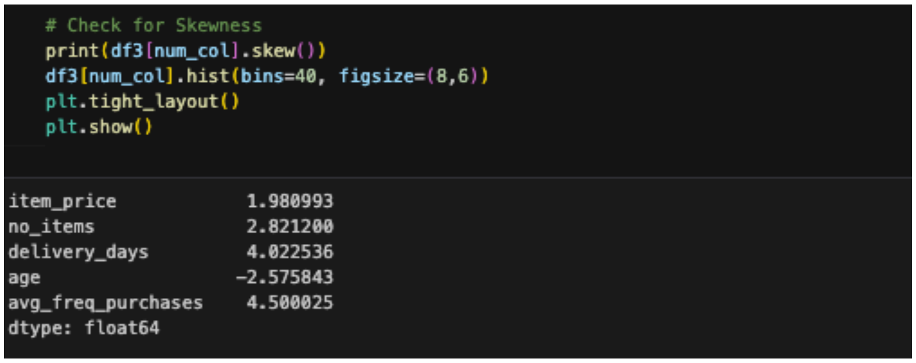
```python
# Orders close to birthday
df2['mdorder'] = df2['order_date'].dt.strftime("%m%d").astype(float)
df2['mddelivery'] = df2['delivery_date'].dt.strftime("%m%d").astype(float)
df2['mdbirthd'] = df2['user_dob'].dt.strftime("%m%d").astype(float)
df2['preorders'] = df2['mdbirthd'] - df2['mddelivery']
df2['is_bday'] = ((df2['preorders'] < 0) & (df2['preorders'] > -30) | ((df2['mdbirthd'] >= 1215) & (df2['mddelivery'] <= 115))).astype(int)

df2.drop(['mdorder', 'mddelivery', 'mdbirthd', 'preorders'], axis=1, inplace=True)
```

Figure 5.16 Creation of attribute "is_bday"



```python
# Train Test Split
from sklearn.model_selection import train_test_split

X = df2
y = df2['return']

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

Figure 5.17 Splitting of data into train-test split



```python
# Check for Skewness
print(df3[num_col].skew())
df3[num_col].hist(bins=40, figsize=(8,6))
plt.tight_layout()
plt.show()
```

```
item_price            1.980993
no_items              2.821200
delivery_days         4.022536
age                  -2.575843
avg_freq_purchases    4.500025
dtype: float64
```

Figure 5.18 Original skewness for numeric variables

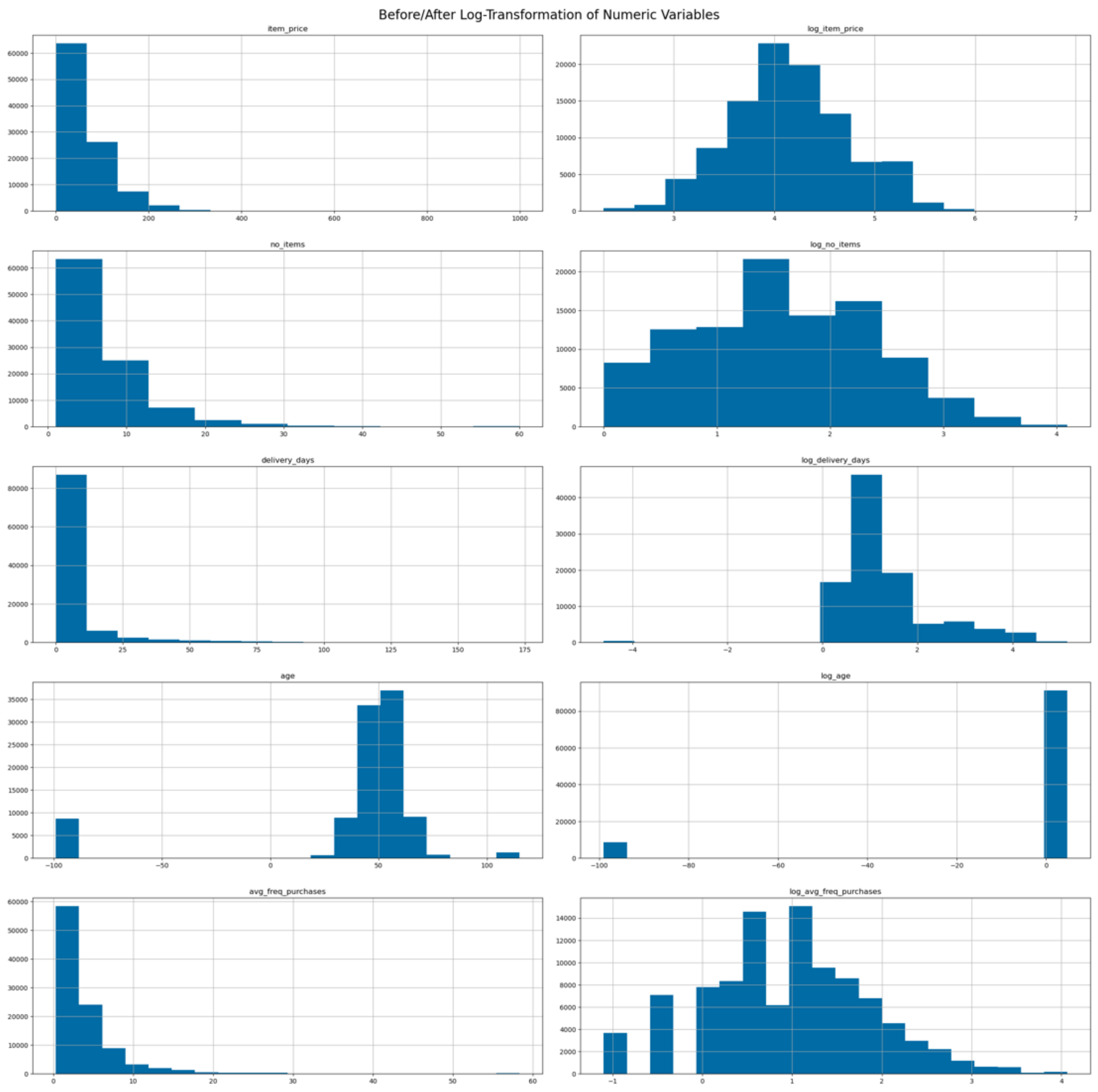
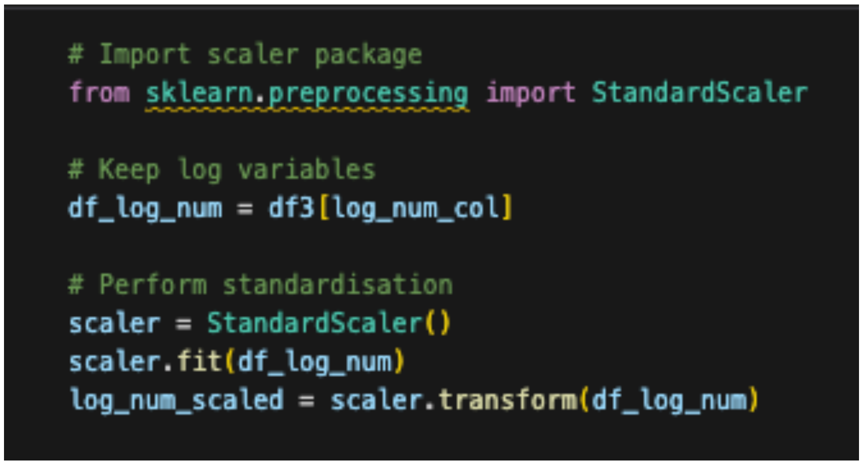
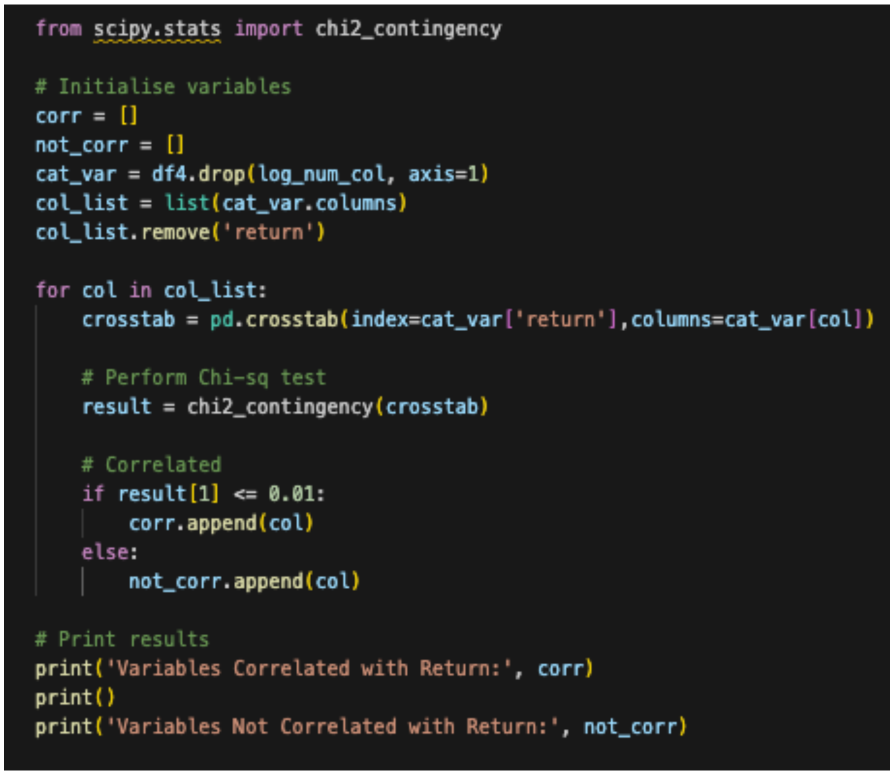


Figure 5.19 Histograms for numeric variables before and after transformation



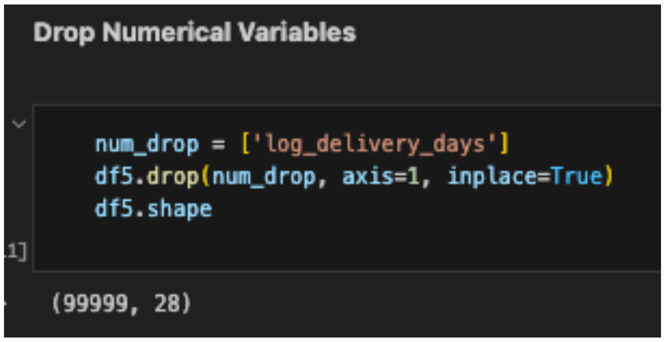
```python
# Import scaler package
from sklearn.preprocessing import StandardScaler

# Keep log variables
df_log_num = df3[log_num_col]

# Perform standardisation
scaler = StandardScaler()
scaler.fit(df_log_num)
log_num_scaled = scaler.transform(df_log_num)
```

Figure 5.20 Scaling of numeric variables



```python
from scipy.stats import chi2_contingency

# Initialise variables
corr = []
not_corr = []
cat_var = df4.drop(log_num_col, axis=1)
col_list = list(cat_var.columns)
col_list.remove('return')

for col in col_list:
    crosstab = pd.crosstab(index=cat_var['return'],columns=cat_var[col])

    # Perform Chi-sq test
    result = chi2_contingency(crosstab)

    # Correlated
    if result[1] <= 0.01:
        corr.append(col)
    else:
        not_corr.append(col)

# Print results
print('Variables Correlated with Return:', corr)
print()
print('Variables Not Correlated with Return:', not_corr)
```

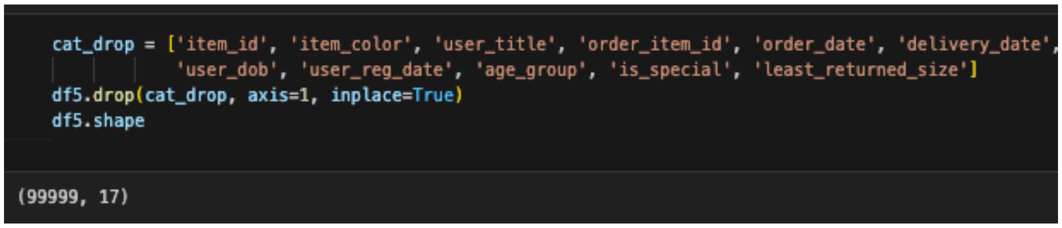
Figure 5.21 Check significance of Categorical Variables


Drop Numerical Variables

```
num_drop = ['log_delivery_days']
df5.drop(num_drop, axis=1, inplace=True)
df5.shape
```

```
(99999, 28)
```

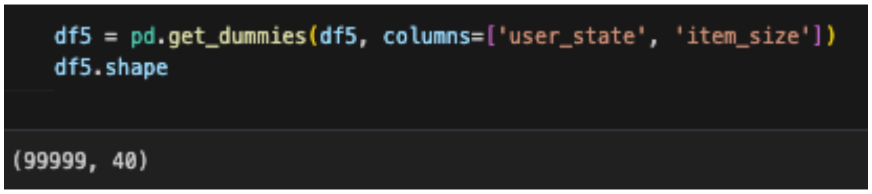
Figure 5.22 Dropping log_delivery_days



```
cat_drop = ['item_id', 'item_color', 'user_title', 'order_item_id', 'order_date', 'delivery_date',
            'user_dob', 'user_reg_date', 'age_group', 'is_special', 'least_returned_size']
df5.drop(cat_drop, axis=1, inplace=True)
df5.shape
```

```
(99999, 17)
```

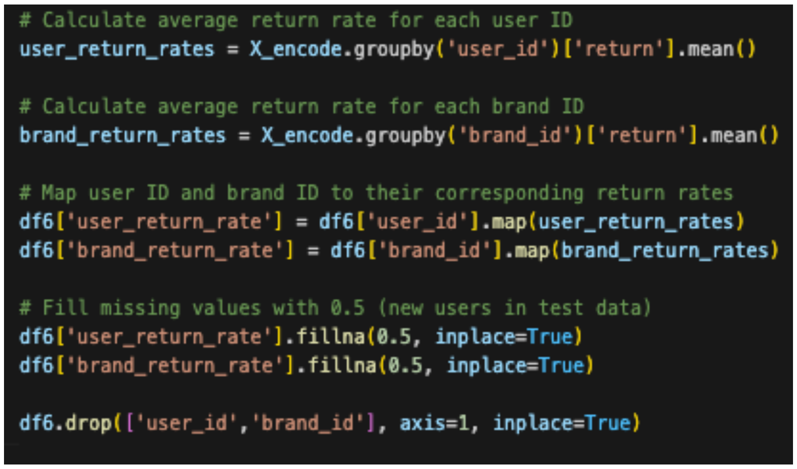
Figure 5.23 Dropping engineered and insignificant attributes



```
df5 = pd.get_dummies(df5, columns=['user_state', 'item_size'])
df5.shape
```

```
(99999, 40)
```

Figure 5.24 One hot encoding



```
# Calculate average return rate for each user ID
user_return_rates = X_encode.groupby('user_id')['return'].mean()

# Calculate average return rate for each brand ID
brand_return_rates = X_encode.groupby('brand_id')['return'].mean()

# Map user ID and brand ID to their corresponding return rates
df6['user_return_rate'] = df6['user_id'].map(user_return_rates)
df6['brand_return_rate'] = df6['brand_id'].map(brand_return_rates)

# Fill missing values with 0.5 (new users in test data)
df6['user_return_rate'].fillna(0.5, inplace=True)
df6['brand_return_rate'].fillna(0.5, inplace=True)

df6.drop(['user_id','brand_id'], axis=1, inplace=True)
```

Figure 5.25 Calculating average return rate for each user and average return rate for each brand

9.2 Logistic Regression Intermediate Results

The table below summarises the Test ROC-AUC and CV ROC-AUC scores of the models considered for Logistic Regression (without interaction terms).

Logistic Regression (Baseline)	**Logistic Regression + LASSO**	**Logistic Regression + Ridge**	**Logistic Regression + Elastic Net**
No hyperparameter tuning	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100, 100]'	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100], 'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
	Optimal value of hyperparameter(s): {'C':100}	Optimal value of hyperparameter(s): {'C':10}	Optimal value of hyperparameter(s): {'C':10} {'l1_ratio:0.1}
Test ROC-AUC score: 0.7564	**CV ROC- AUC score: 0.74557**	**CV ROC-AUC score: 0.74577**	**CV ROC-AUC score: 0.74557**

The table below summarises the Test ROC-AUC and CV ROC-AUC scores of the models considered for Logistic Regression (with interaction terms).

Logistic Regression with interaction terms (Baseline)	**Logistic Regression with interaction terms + LASSO**	**Logistic Regression with interaction terms + Ridge**	**Logistic Regression with interaction terms + Elastic Net**
No hyperparameter tuning	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100]	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100]'	Hyperparameter(s) to tune and their ranges: 'C': [0.001, 0.01, 0.1, 1, 10, 100], 'l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
	Optimal value of hyperparameter(s): {'C':100}	Optimal value of hyperparameter(s): {'C':10}	Optimal value of hyperparameter(s): {'C':10} {'l1_ratio:0.1}
Test ROC-AUC score: 0.7564	**CV ROC- AUC score: 0.7454**	**CV ROC-AUC score: 0.7454**	**CV ROC-AUC score: 0.7454**

Since CV AUC-ROC scores were similar across the penalty methods (for with and without interaction terms), we opted for elastic net due to its ability to perform variable selection and regularization simultaneously.

We further tuned the 'max_iter' for each elastic net model, with range [100,200,300]. We found that the optimal ‘max_iter’ value for the Logistic Regression with Elastic Net Model (without interaction terms) was 200 and this model obtained a CV ROC-AUC Score of 0.7456. As for the Logistic Regression with Elastic Net Model (with interaction terms), the optimal ‘max_iter’ value was 100 with a CV ROC-AUC Score of 0.7454.

The table below shows the Test ROC-AUC Scores of the baseline model and tuned Elastic Net model for Logistic Regression with and without interaction terms.

Model	**LogReg (Baseline)**	**LogReg + Elastic Net (Tuned)**	**LogReg_ InteractionTerm (Baseline)**	**LogReg_ InteractionTerm + Elastic Net (Tuned)**
Test ROC-AUC Score	**0.7566**	**0.7566**	**0.7564**	**0.7564**

The models without interaction terms performed better than the model with interaction terms. The inclusion of interaction terms could have made the model more prone to overfitting (although the difference is marginal). We selected the baseline model as opposed to the Elastic Net model for Logistic Regression (without interaction terms) as when we considered more decimal points, the former had a higher Test ROC-AUC score of 0.7566285 than the latter which had a Test ROC-AUC score of 0.7566278