University of Central Florida

School of Data, Mathematical, and Statistical Sciences (SDMSS)

STA4724 - Big Data Analytics

Shopper Behavior Analysis

Authors:

Jackson Small, Andres Machado, Thomas Tibbetts, Sarah Taha

December 5, 2025

TABLE OF CONTENTS

CHAPTER 1: Introduction

1.1 Project Statement and Motivation

Businesses capture every click, pause, and page view to try to understand a customer's preferences. This understanding can lead to better recommendations for products and services, as well as to better the overall experience for all customers. The purpose of this project is to investigate those underlying patterns as users navigate an online retail site and how they relate to a user's buying decision. Through supervised classification methods, we aim to evaluate the session features that strongly influence a customer's decision to purchase. This type of analysis is valuable for any business because understanding customer intent can guide internal discussions about marketing, site design, and overall revenue strategy.

1.2 Dataset Description

We used the *Online Shoppers Purchasing Intention Dataset*[1] from the UCI Machine Learning Repository. The dataset was anonymously donated by a retailer and was used in [1] which served as a stepping stone for our analysis. Due to the anonymous nature of this dataset, the specific retailer and collection details are undisclosed for privacy reasons. Consequently, any statistical inferences or predictive models

[1]Available at: https://archive.ics.uci.edu/dataset/468/online+shoppers+purchasing+intention+dataset

produced in this work reflect patterns within this dataset only and should not be generalized to other retailers or populations without further validation. Despite these limitations, the dataset offers a valuable opportunity to explore real-world e-commerce behavior and apply supervised learning techniques to predict purchasing decisions.

Our task is designed as a binary classification problem with the measure being whether a customer made a purchase or not. The dataset is composed of 12,330 different user sessions over a 1-year span (year unspecified). Of the 12,330 sessions in the dataset, 10,422 of them did not end up in a sale, and the rest (1908) did end up shopping (see Figure 1.1). In Table 1.1, we can observe the 18 features from the dataset with a short description of each. We would like to point out that `BounceRates`, `ExitRates`, and `PageValues` are averaged features imported from the Google Analytics figures of the retailer's site. According to [1], "The value of the `BounceRates` feature for a Web page refers to the percentage of visitors who enter the site from that page and then leave ("bounce") without triggering any other requests to the analytics server during that session. The value of the `ExitRates` feature for a specific Web page is calculated as for all pageviews to the page, the percentage that were the last in the session. The `PageValues` feature represents the average value for a Web page that a user visited before completing an e-commerce transaction."

Feature Name	**Data Type**	**Description**
Administrative	N	Page visits about account management
Administrative_Duration	N	Total time (s) user spent in acct. manag.
Informational	N	Page visits about general info about site
Informational_Duration	N	Total time (s) user spent in info pages
ProductRelated	N	Page visits about products
ProductRelated_Duration	N	Total time (s) user spent in product pages
BounceRates	N	Avg. bounce rate of the pages during session
ExitRates	N	Avg. exit rate of the pages during session
PageValus	N	Avg. page value of the pages during session
SpecialDay	N	Closeness of the site visit to a holiday
OperatingSystems	C	Operating system of user
Browser	C	Browser of user
Region	C	Geographic region of user
TrafficType	C	Traffic source by which user arrived (ex: SMS)
VisitorType	C	Type of user (New, Returning, Other)
Weekend	C	Indicates if session was on weekend or not
Month	C	Month session took place
Revenue *(a)*	C	Indicates if session resulted in sale or not

Table 1.1: Table listing and describing the features in the dataset

(a) Target Feature

(b) Numerical (N) and Categorical (C)

1.3 EDA

The exploratory data analysis begins by outlining the initial patterns in the dataset and highlighting the most relevant observations. We then will examine correlations between features and consider transformations to address skewed variables and improve model readiness. A clustering analysis follows to uncover groupings of user behaviors that may not align with the target variable alone. Finally, we will evaluate feature importance to identify the variables most relevant in our dataset.For more details on any step in this process, refer to the Jupyter notebook in the GitHub repository titled *Final_EDA.ipynb*.

1.3.1 Initial Observations

Before exploring individual features, we examined the distribution of the target variable. A simple bar chart (Figure 1.1) makes the imbalance clear: the dataset contains far more non-purchasing sessions than purchasing ones. This imbalance is important to keep in mind for later modeling decisions.

Next, we examined the distributions of the numerical features and found that most were noticeably right-skewed (Figure 1.2). This pattern aligns with the imbalance in the dataset: non-purchasing users tend to spend less time on the site and visit fewer pages, producing many low-value observations. In a later section, we address this skewness through appropriate transformations to improve model performance. Observing how the target variable interacts with the numerical variables, we find that certain variables help distinguish one class of user from the other (Figure 1.3). For example, non-purchasing users (in blue) display a much wider box, indicating that `ExitRates` varies far more among sessions that did not result in a purchase (in orange). This spread suggests greater inconsistency in how non-purchasers leave pages in the site compared to purchasers.

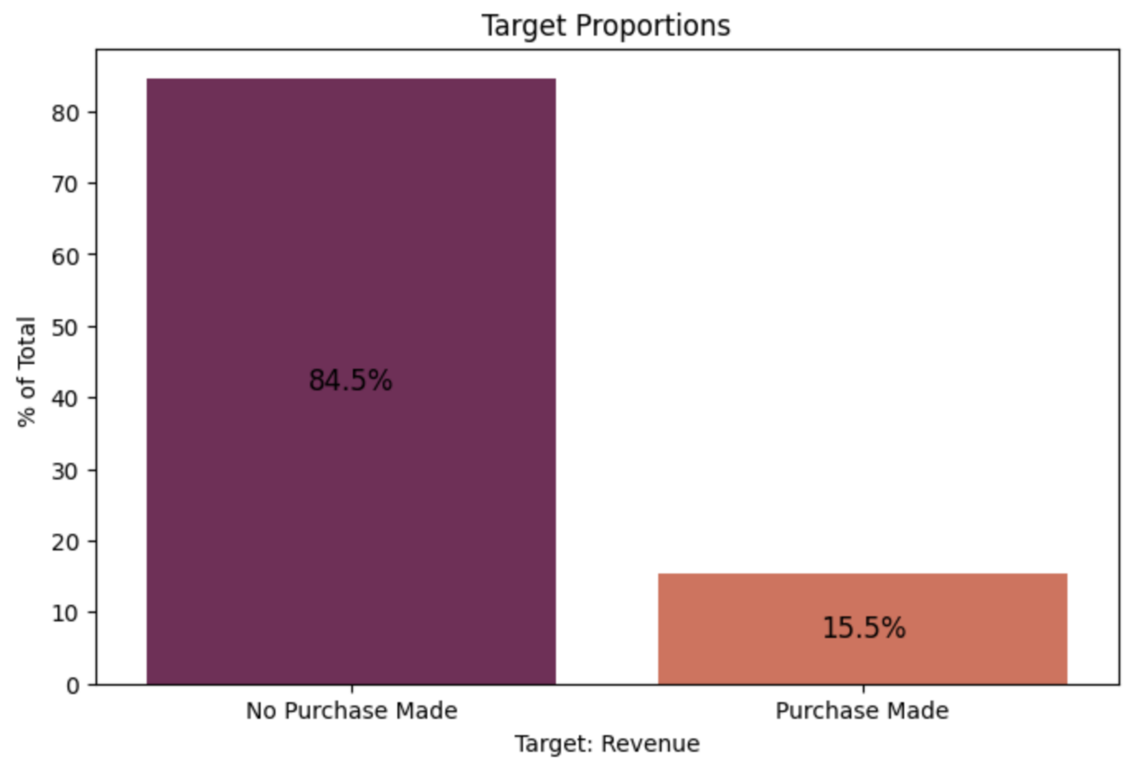

Figure 1.1: Target (Revenue) Proportion Bar Plot

We also examined the categorical features to understand broader behavioral patterns . The month variable showed substantially more sessions in March, May, and November, suggesting clear seasonality in site activity (Figure 1.4). Most users were returning visitors, which aligns with typical purchasing behavior where individuals browse, leave, and later return to complete a transaction (Figure 1.5). Finally, the proportion of returning versus new visitors was similar across purchasing and non-purchasing sessions, indicating that visitor type alone is not a strong differentiator of purchase outcomes (Figure 1.6).

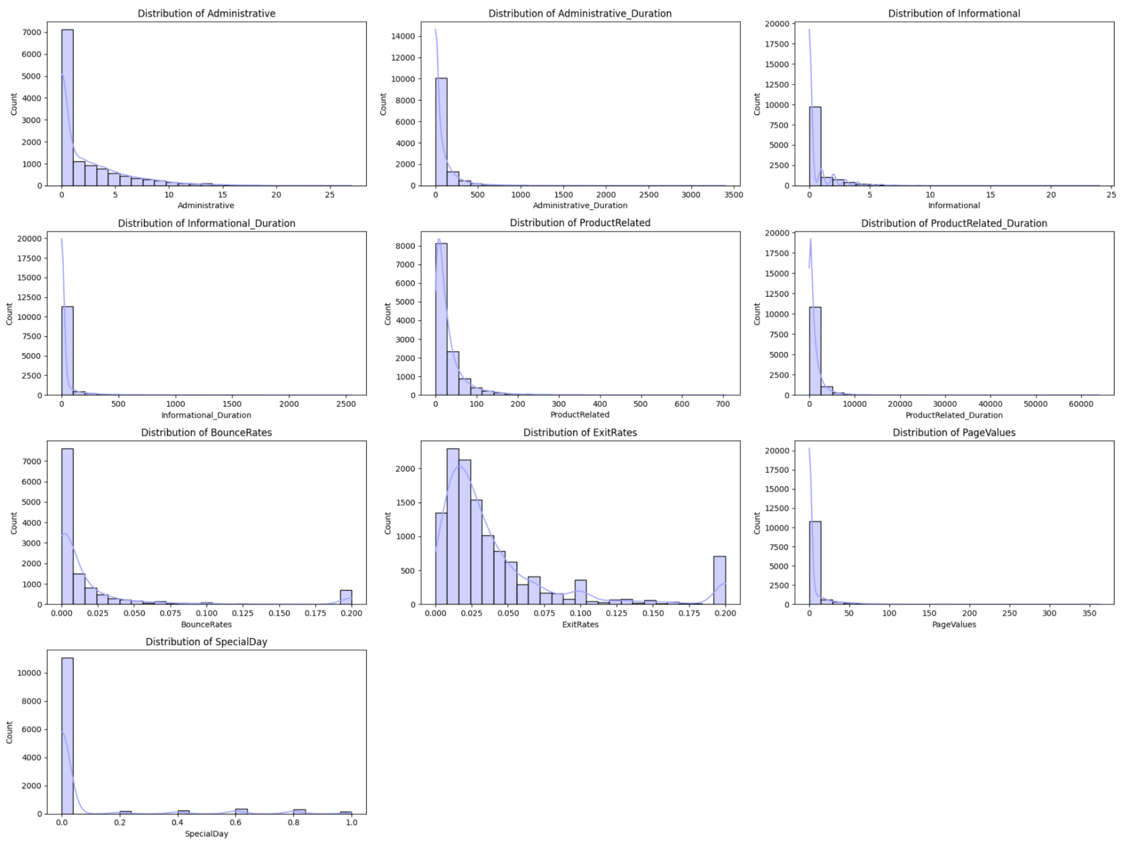

Figure 1.2: Distributions for Numerical Variables

With the initial patterns established, we next examine how the features relate to one another and to the target variable. This involves analyzing key correlations and applying transformations to address skewness and prepare the data for modeling.

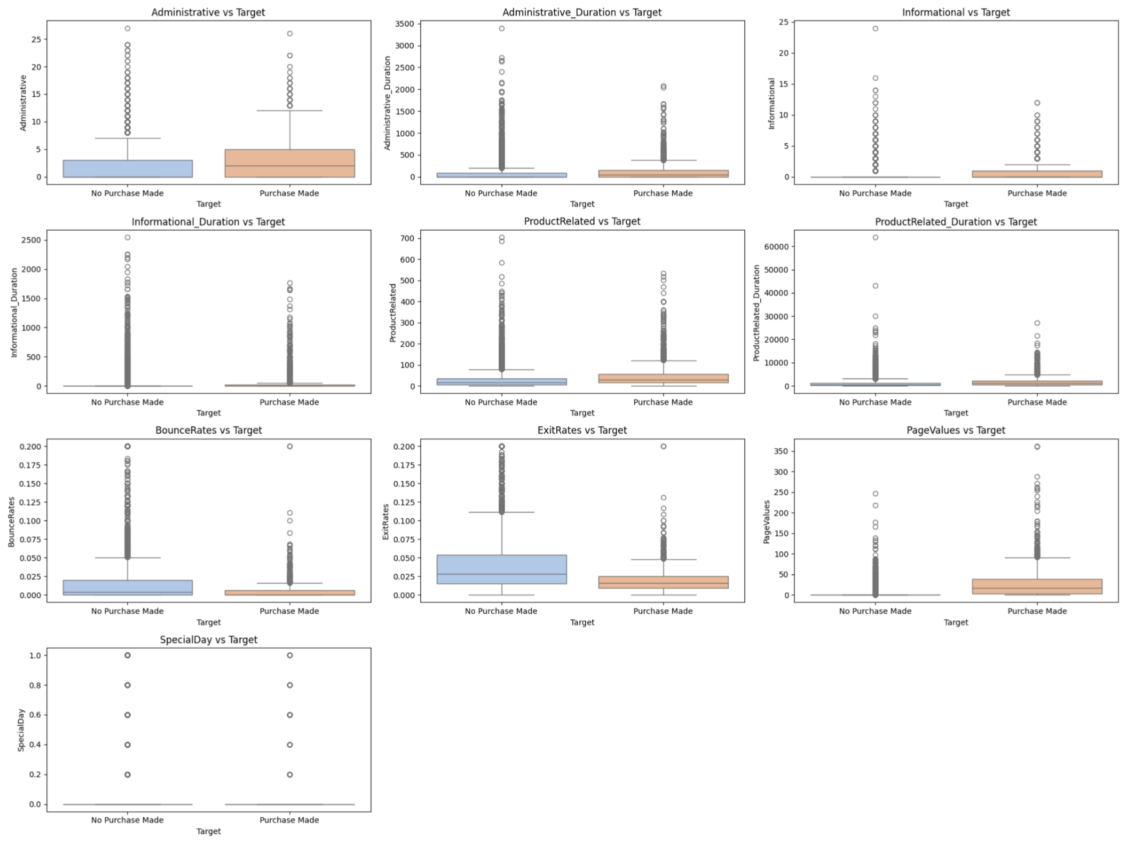

Figure 1.3: Box Plots for Numerical Variables against Target

1.3.2 Correlation and Transformations

Before applying any transformations, we examined the correlations among all numerical features (Figure 1.7). The heatmap highlights several clear relationships. First, all Web page counters and their corresponding duration variables exhibit moderate to strong positive correlation, which is expected because they measure related aspects of the same behavior. There is also a moderate correlation between visits

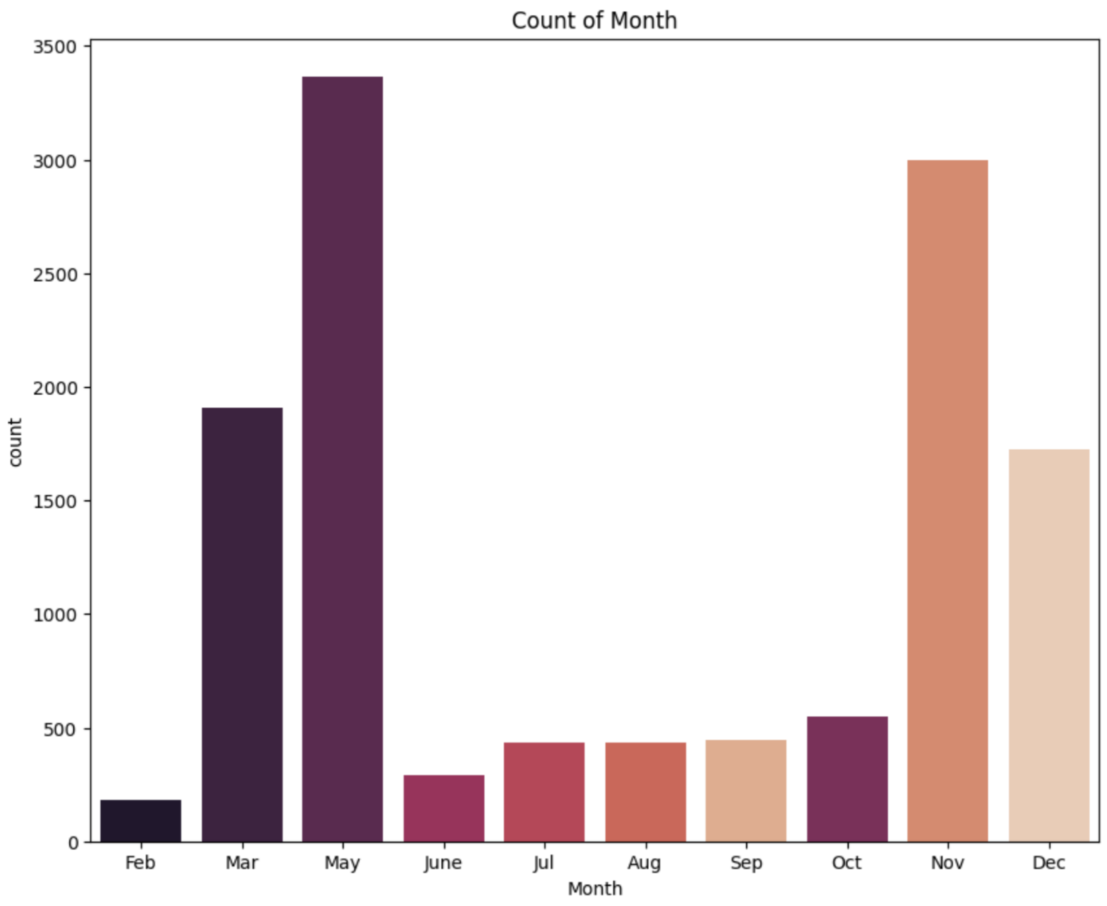

Figure 1.4: Bar Plot showing count of sessions per month

to `Administrative` and `ProductRelated` pages ($r \approx 0.43$), suggesting that users who spend time on account-management pages also tend to explore product pages. In addition, `BounceRates` and `ExitRates` are almost perfectly correlated, indicating substantial redundancy; we will engineer a "leaving" feature from these two variables to assess whether one or both can be removed from the model. Finally, `PageValues` shows the strongest association with the target variable `Revenue` ($r \approx 0.49$). This

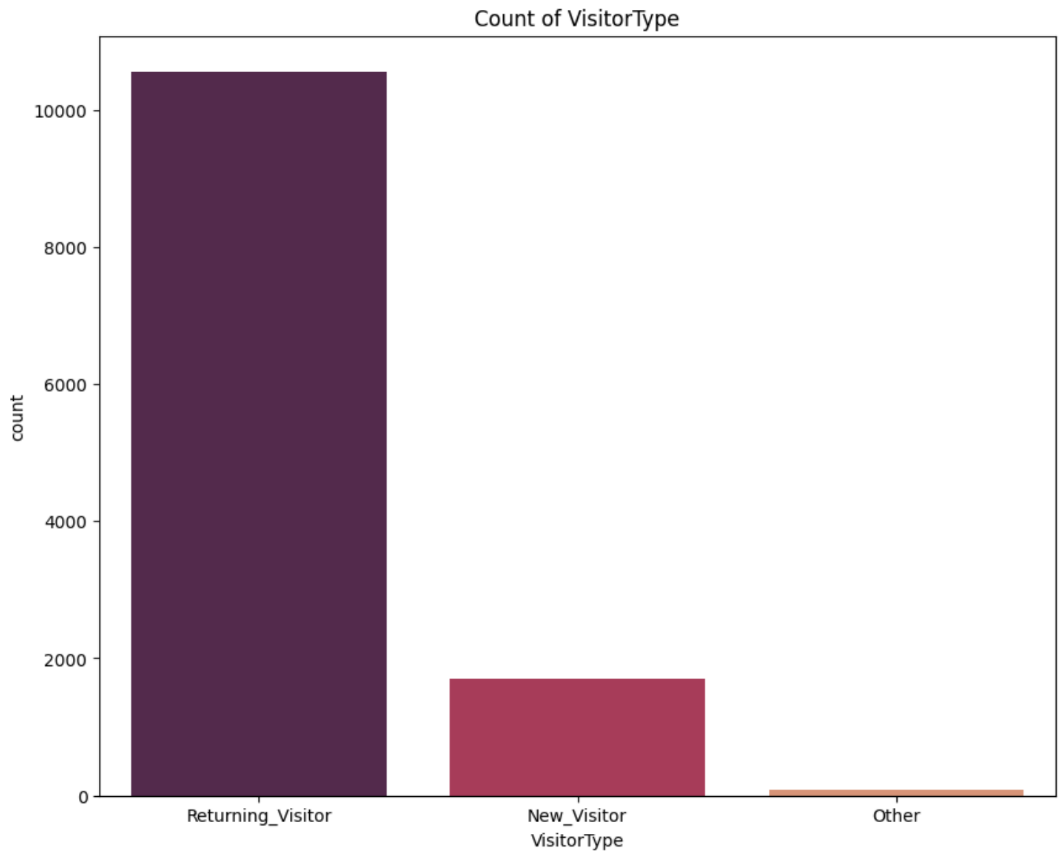

Figure 1.5: Bar Plot showing count of visitor types

aligns with its definition as the expected monetary value per page view.

For our next correlation heatmap (Figure 1.8), we engineered two new features. `LeavingRate` is the average of `BounceRates` and `ExitRates`. `TotalDuration` is the sum of `Administrative_Duration`, `Informational_Duration`, and `ProductRelated_Duration`. `TotalDuration` revealed that `Product_Related_Duration` is the strongest indicator of the total duration a user spent on the website before exiting. `LeavingRate` did

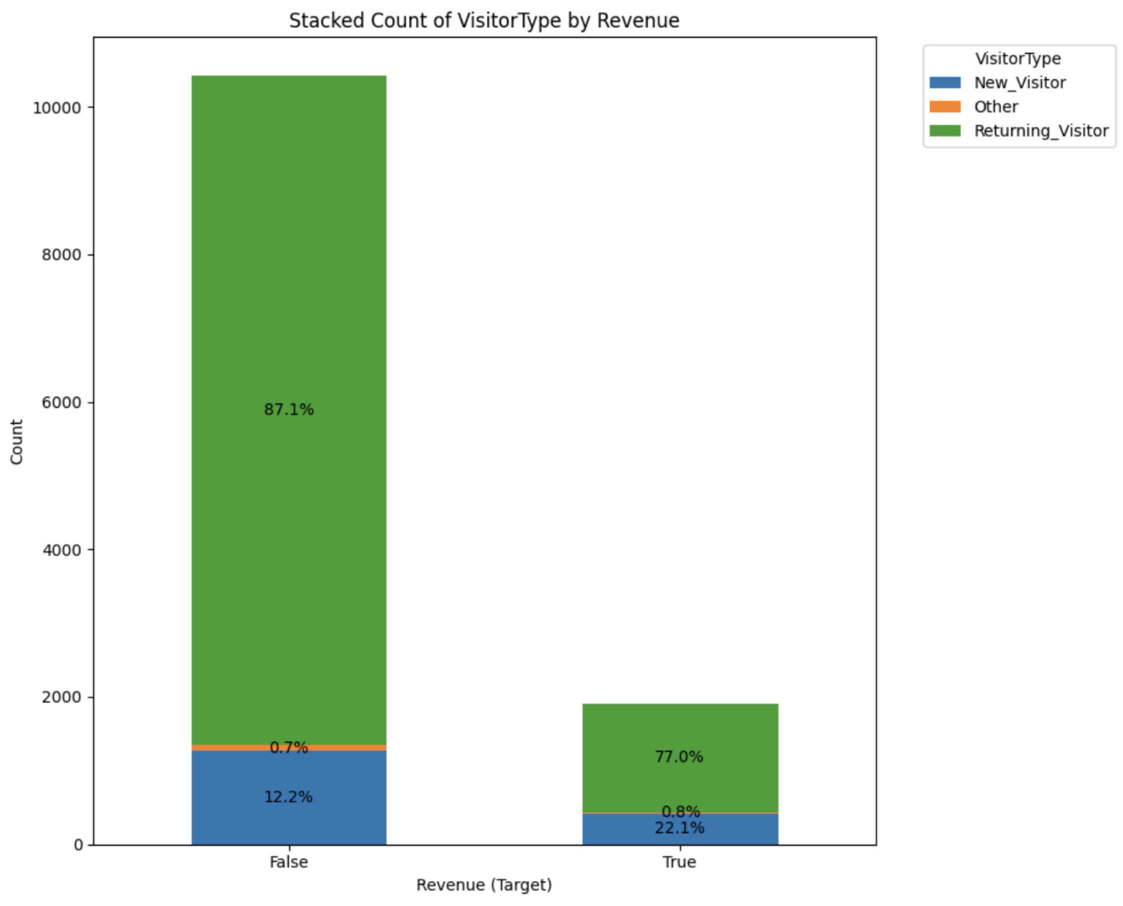

Figure 1.6: Stacked bar plot showing visitor type proportions versus target

not reveal any other vital information. While interesting, these engineered features are not significantly better than the original features.

The scatterplots in Figure 1.9 visualize the relationship between the number of pages visited in each category and the total time spent on those pages, with points colored by whether the session resulted in revenue. After seeing the initial correlation in the heatmap, the scatterplot is used to analyze their relationship further. Both

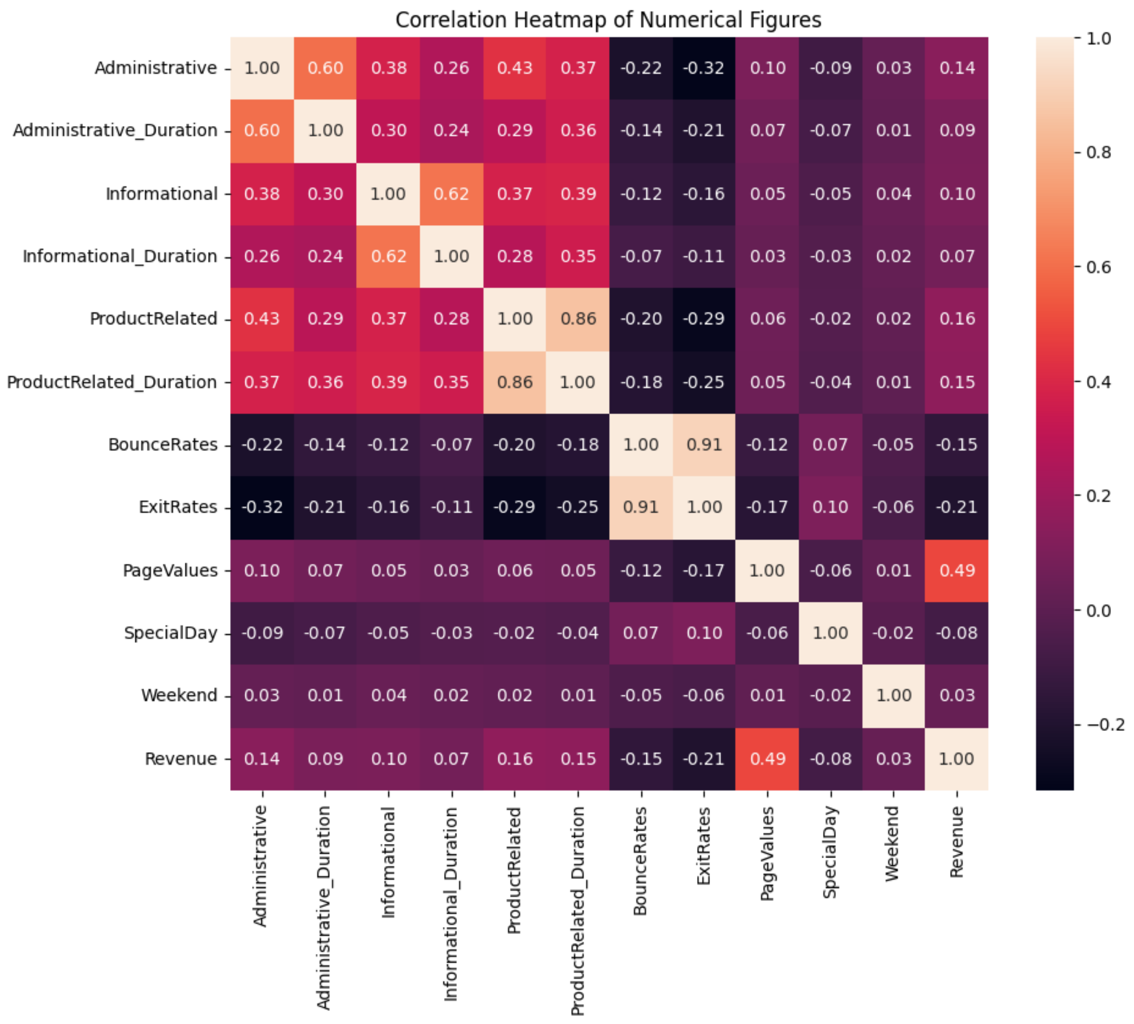

Figure 1.7: Initial Correlation Heatmap of Numerical Figures

`Administrative` and `Informational` appear to have a weak and noisy upward trend with page visits and duration (as expected). `ProductRelated` revealed a possible linear trend in duration and visits. Purchasing and non-purchasing sessions are mixed throughout the plots, though purchase sessions appear somewhat more often

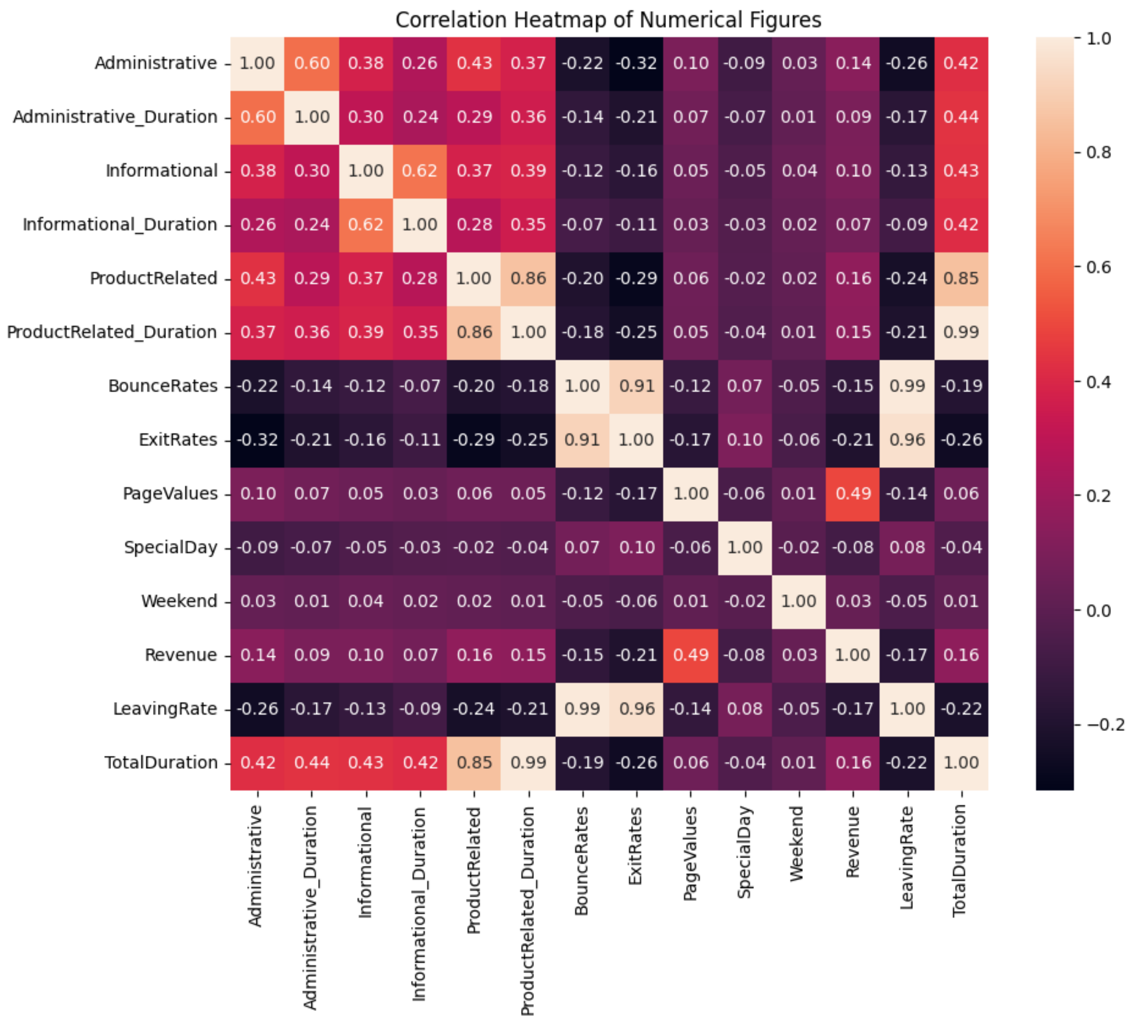

Figure 1.8: Correlation Heatmap of Numerical Figures with Engineered Features

at higher page counts and longer durations, especially for `Product_Related` pages. Interestingly, it appears that most visual outliers in all three graphs are no-purchase sessions.

An additional transformation was inspired by the skewed distributions in Figure 1.2.

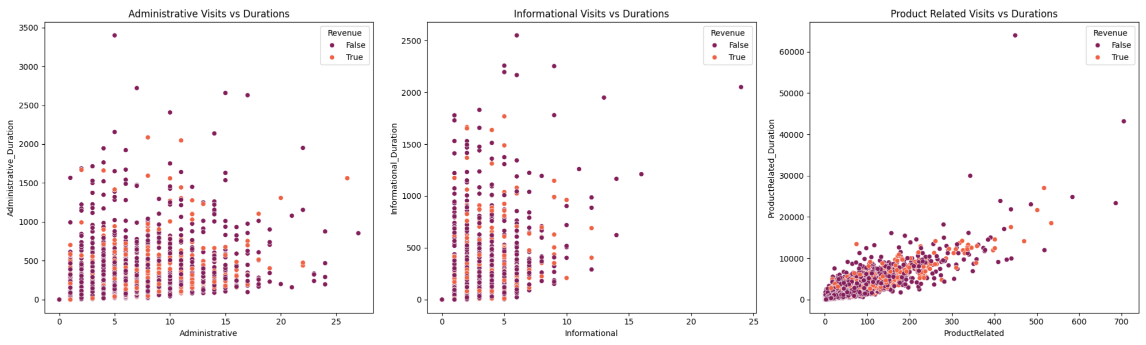

Figure 1.9: `Administrative`, `Informational`, and `ProductRelated` Page Visits vs Duration with `Revenue`

A simple 'log(x+1)' transformation was performed (Figure 1.10) and the only major improvement was for `ProductRelated` and its duration. Both distributions gained a more consistent bell shape, but still faced issues with major outliers and long tails.

1.3.3 Clustering

While the problem is a supervised learning task, we decided to perform some unsupervised learning on the data to see if we could exploit some natural structure for the classification task. We decided to use the k-means algorithm for its ease of implementation.

We began by plotting the inertia (total intra-cluster variation) of k-means instances for different values of k. Each k-means was trained using the min-max normalized numerical columns of the predictive features (the target feature, `Revenue`, was not used). The elbow method (Figure 1.11) was used to visually evaluate which number

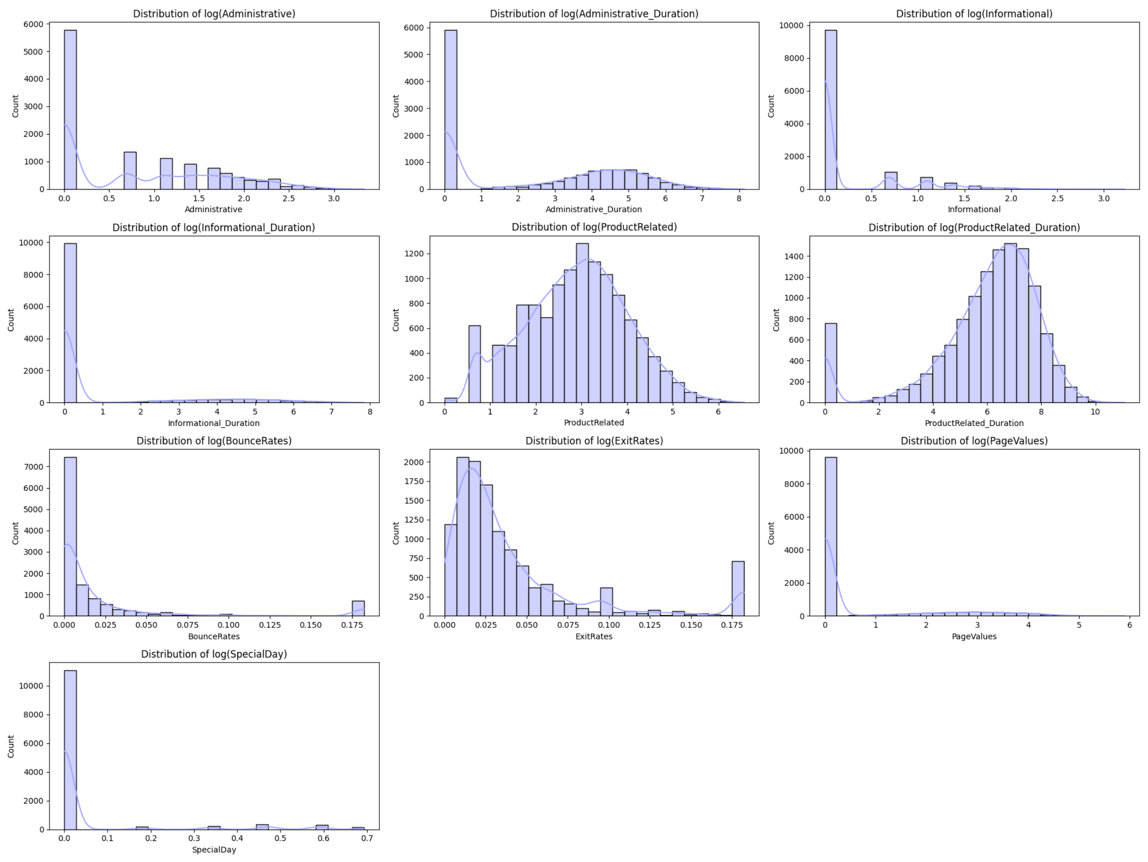

Figure 1.10: Log Transformation of Distributions of Numerical Values

of clusters was best suited to the data. To identify an elbow on the inertia graph, we look for a point of inflection where the rate of inertia decrease becomes less steep with the addition of more clusters.

When plotting the distribution of the `Revenue` variable among clusters(Figure 1.12), it appeared that the cluster labels were able to make some meaningful differentiation between purchasers and non-purchasers.

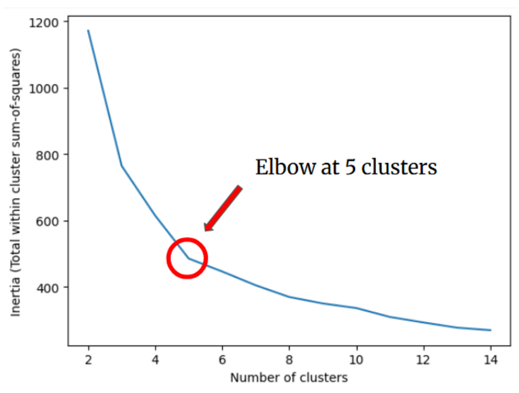

Figure 1.11: The strongest visible elbow was at 5 clusters.

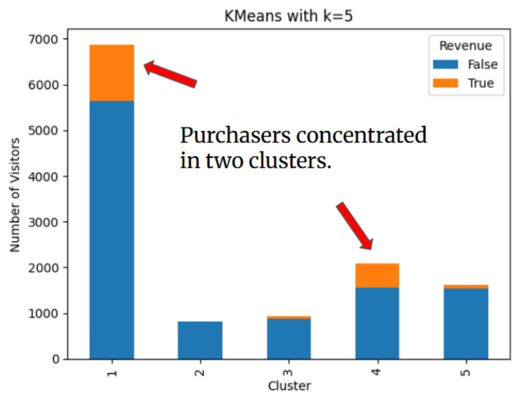

Figure 1.12: The purchasing sessions in the dataset (`Revenue` = True) appeared to be concentrated in two clusters.

Since the purchasers in the dataset appeared to be concentrated in a limited number of clusters, we suspected that k-means cluster labels may be a useful feature for the classification task.

1.3.4 Feature Selection

Next, we performed feature selection. This process can help model accuracy, increase training time and also reduce overfitting our models later on. First, we assessed the numerical features through an ANOVA F-test. The results concluded that the feature `PageValues` is the most significant predictor of revenue, with an F-score of 3895. This confirms our claim that the user's engagement with high-value pages is stronger than the duration of their session. We also noticed that `ExitRates` with F = 531 was another significant indicator of revenue, indicating it as an effective churn signal.

For the categorical variables, we performed a Chi-Square test. As seen in our EDA, seasonality played a big role, with `Month_Nov` scoring the highest with a score of 219. This makes sense as there is often heavy traffic through shopping sites during the holidays. Also we created `c3` and `c5`, both engineered features of clusters. c5 achieved a Chi-Square score of 72, showing how it outperforms our regular features like `Region` and `TrafficType`.

Finally, we used cross-validation using a Random Forest (RF) classifier to find feature interactions. The RF model was very consistent with our statistical tests, showing

`PageValues`, `ExitRates`, and `ProductRelated_Duration` as the top three predictors. All of the significant features can be seen in the following graphic.

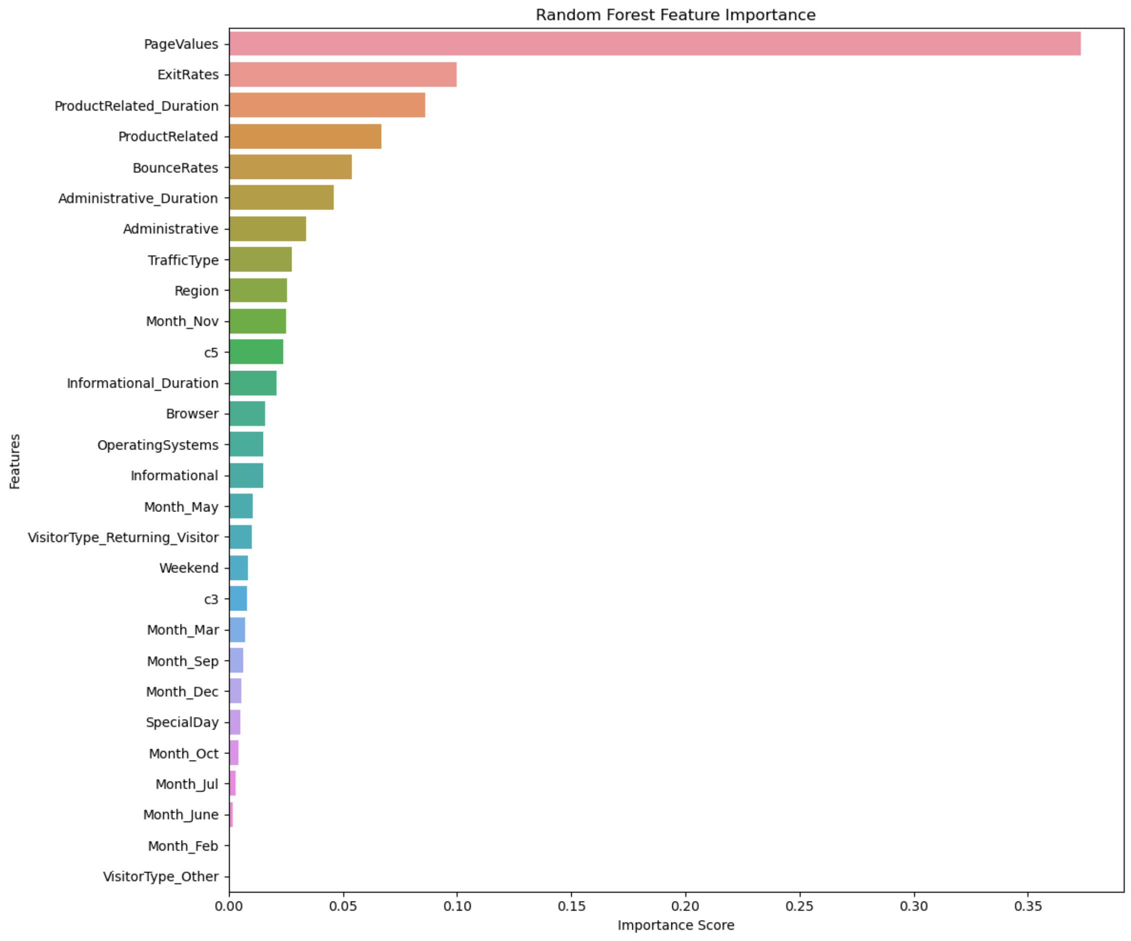

Figure 1.13

After all of this was completed, we prioritized these features for our final modeling pipeline while removing redundant variables to avoid target leakage.

CHAPTER 2: Modeling

2.1 Modeling

2.1.1 Pipeline

From our insights in EDA, we decided to transform the dataset by log-transforming the `ProductRelated` and `ProductRelated_Duration` features to obtain a more symmetric distribution. We also decided to add a feature containing the one-hot encoded 5-means cluster label of each observation. The preprocessing pipeline was implemented using the Pipeline and ColumnTransformer objects in the sci-kit learn library.

When fitting the preprocessing pipeline for each training split, an instance of 5-means clustering was initialized and fit on the training data. Whenever the resulting estimator was used to make predictions on unseen data, that clustering instance was used to add cluster labels to the incoming observations. In this way, we were able to transform the data to add informatively-engineered features without causing leakage from the test or target data.

2.1.2 KNN (K-Nearest Neighbors)

K-Nearest Neighbors (KNN) classifies a new observation by looking at the K most similar points in the training data and assigning the majority class among them. Its main strengths are simplicity, no assumptions about data distribution, and the ability to capture nonlinear boundaries. However, it can be computationally expensive on large datasets, sensitive to irrelevant or unscaled features, and prone to poor performance when classes are imbalanced. For this task, KNN is a reasonable choice because the user-behavior data contains nonlinear relationships and it serves as a baseline for other models.

We ran 3 KNN models:

1. Vanilla KNN (no hyperparameter tuning)
2. Vanilla KNN with SMOTE (no hyperparameter tuning but with resampling)
3. Tuned KNN (3 tuned hyperparameters)

We chose to tune the following hyperparameters:

- K: Number of neighbors that the Sci-Kit Learn KNN algorithm uses to determine a given observation's class. Odd values preffered to avoid ties.
- P: Determines the distance measure used by the KNN algorithm. P = 1 means that manhattan distance is used and P = 2 means that euclidean distance is used.

KNN Model	**Recall**	**Precision**	**AUC Score**	**Weighted F1**
Vanilla Model	0.81	0.73	0.84	0.88
Vanilla Model with SMOTE	0.74	0.80	0.87	0.87
Fine Tuned	0.82	0.72	0.87	0.88

Table 2.1: KNN modeling results

(a) All metrics are macro averages except Weighted F1

- Weights: The weights hyperparameter controls whether all neighbors contribute equally or whether closer neighbors have a stronger influence on the predicted class. With the *uniform* weight function each neighbor contributes equally to the prediction, while with the *distance* weight function, nearer neighbors carry more influence and can improve performance when closer points are more informative than farther ones.

The final tuned model had the following hyperparameters: K = 13, P = 2, and Weights = *distance*. As we can see in Table 2.3, the fine tuned KNN model having the highest recall at 0.82, indicating it identified purchasers more effectively than the other variants. Precision decreased slightly relative to the vanilla and SMOTE models, reflecting a tradeoff between catching more positives and introducing more false positives. Overall, AUC scores remained the same across all KNN models, showing that tuning and resampling had limited impact on the classifier's overall ability. Weighted F1 score is promising as well.

2.1.3 Logistic Regression

Logistic Regression is a conceptually simple model for classification. For binary classification, it is assumed that the log-odds of an observation belonging to the positive class can be modeled as a linear function of the input features. When training, iterative methods are usually used to calculate the parameters in order to maximize the likelihood of observing the given labels for the training set.

Logistic regression is appealing because it is very simple to train, and once trained, predictions on new data can be made extremely quickly. However, since the parameters are learned using log-likelihood loss, the model can be subject to overfitting (this may be mitigated by regularization techniques). Additionally, since logistic regression only generalizes a linear model, it is typically not well-suited to finding nonlinear decision boundaries in complex data sets. In this investigation, we hoped that including the cluster label features might add some nonlinear character to the estimator even if the algorithm was strictly linear. Similarly to KNN, we trained and evaluated three logistic regression models: vanilla, vanilla with SMOTE resampling, and with tuned hyperparameters.

The only meaningful hyperparameter we tuned was C, which controlled the strength of L2 regularization of the parameter estimates. An optimal value of C was calculated using 10-fold stratified cross-validation.

The final tuned model had the hyperparameter $C = 70.17$. However, inspection showed that changing the value of C often had little effect on the weighted F1 score

Log Regression Model	**Recall**	**Precision**	**AUC Score**	**Weighted F1**
Vanilla Model	0.68	0.85	0.90	0.87
Vanilla Model with SMOTE	0.69	0.85	0.90	0.87
Fine Tuned	0.68	0.85	0.90	0.87

Table 2.2: Logistic Regression modeling results

(a) All metrics are macro averages except Weighted F1

calculated through cross-validation.

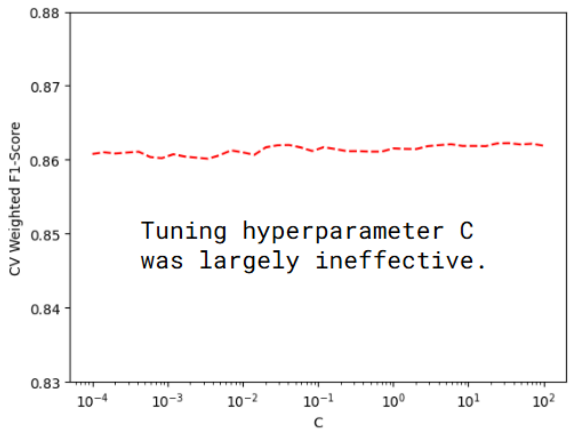

Figure 2.1: C versus CV Weighted F1-Score Graph

Neither SMOTE resampling nor tuning the C hyperparameter appeared to have any appreciable difference on the performance metrics of the logistic regression model.

2.1.4 Random Forest Classification

Random Forest is an Ensemble method that combines the output of multiple decision trees. The method is not only robust to outliers but has many factors that can be modified to fine-tune models. Random Forest through Sci-Kit Learn's package also has specific fine-tuning tools for modifying how class imbalance is handled, making it and excellent option for our dataset.

We ran 3 total Random Forest Classification models:

1. Vanilla Random Forest (no hyperparameter tuning)
2. Vanilla Random Forest with SMOTE (no hyperparameter tuning but with resampling)
3. Tuned Random Forest (6 tuned hyperparameters)

The vanilla model revealed a strong model with an AUC of 0.93. The model struggled with predicing Purchases made but excelled at predicting when a session would result in no-purchases.

We try SMOTE to use resampling to possibly help with the class imbalance and ideally improve the model's ability to predict a purchase. The AUC actually went down by 1% and the precision for purchases made went down as well.

Knowing this, I wanted to see how Sci-Kit Learn's built in class balancing would handle our data set as well as the interaction with the hyperparameters used.

We chose to tune the following hyperparameters:

- Estimators: Checking the amount of trees used between 50-500.
- Max Depth: Checking to see if restricting the depth of the tree protects against overfitting. Testing 10, 20, and no max depth.
- Max Features: The number of features in the random subset at each node. This will ensure that the trees are not predicting the same, strong predictors and allows for more variety in the tree. We will be testing no max and a randomized subset of the squareroot amount of total features to encourage variety in trees.
- Class Weight: three options for the model to adjust the class weights. Balanced, balanced subsample, and no adjustment.
- Minimum Sample Split: The minimum amount of elements in a leaf before it can split, increasing this number can help prevent overfitting We are testing 2 (the basis), 5 and 10 needed before splitting a node.
- Criterion: Testing the quality of the split between Gini and Entropy.

After an 80+ minute wait running the grid search, we got the resulting hyperparameters:

- Estimators: 500 - the highest amount of trees in the forest, the model could see minor improvements if increased we would greatly increase the time to run the model but possibly see minor improvements.

- Max depth: None, the trees can go as deep as necessary. This is balanced by the minimum sample split's result.
- Max Features: Squareroot - this demonstrates that there was a tree similarity issue where all the trees preferred to use the exact same, strong features. Implementing this will allow for stronger diversity in the file.
- Class Weight: Balanced - Errors on purchases are penalized more heavily than on no purchases.
- Minimum Sample Split: 5, this prevents overly-trained branches.
- Criterion: Entropy. Entropy is more sensitive to changes in small probabilities which contributed to estimators and max depth not needing restrictions.

Even after tuning, the improvement over the SMOTE model is less than 1% the weighted average and the tuned model had a less than 1% improvement for the ROC AUC. While the "No Purchases" f1 score stays the same, the "Purchase" f1 score raises, meaning the tuned model identifies buyers slightly better without hurting performance on non-buyers.

In practice, this Random Forest is most useful for predicting sessions that will not lead to a purchase, so its output could be used to trigger early discounts, proactive chat, or targeted follow-up emails instead of sending the same outreach to every visitor.

Random Forest Classification	**Recall**	**Precision**	**AUC Score**	**Weighted F1**
Vanilla Model	0.75	0.85	0.93	0.89
Vanilla Model with SMOTE	0.77	0.82	0.92	0.90
Fine Tuned	0.80	0.84	0.93	0.91

Table 2.3: Random Forest Classification modeling results

(a) All metrics are macro averages except Weighted F1

2.1.5 SVM (Support Vector Machine)

Support Vector Machines (SVM) is a supervised learning algorithm that classifies data based on finding the best hyperplane that maximizes the margin. We wanted to use SVM for this task because of its effectiveness in modeling non-linear relationships and putting input into a higher dimension. We also want to note that SVMs compared to other models are computationally intensive.

Similar to KNN and Logistic Regression, we used our created custom pipeline that standardizes numerical features and includes our engineered K-means cluster labels, trained and evaluated three SVM models: vanilla, vanilla with SMOTE resampling, and a `GridSearchCV` SVM. We explored the following hyperparameters:

1. C: The regularization parameter that controls the trade-off between a smooth decision boundary and classifying points accurately. A lower C creates a larger margin, while a higher C creates a harder boundary. Grid $[0.1, 1, 10, 50]$ was used.

2. Kernel: $[\text{Linear}, \text{RBF}]$. The Linear kernel serves as baseline, while the RBF (Radial Basis Function) kernel was included to show a non-linear and high dimensional relationship.

3. Gamma: The Gamma value determines how much influence a certain training point has on the boundary. Grid $[\text{Scale}, 0.1]$ was used.

Optimal hyperparameters found were $C = 50$, Kernel = RBF, and Gamma = Scale.

SVM Model	**Recall**	**Precision**	**AUC Score**	**Weighted F1**
Vanilla Model	0.68	0.86	0.90	0.87
Vanilla Model with SMOTE	0.79	0.81	0.90	0.89
Fine Tuned	0.73	0.83	0.84	0.88

Table 2.4: SVM modeling results

(a) All metrics are macro averages except Weighted F1

As shown in Table 2.4, the Vanilla with SMOTE model showed the strongest overall performance, with a Recall of 0.79 and a Weighted F1 Score of 0.89. This highlights the importance of adding SMOTE resampling, as this model outperformed our hyperparameter tuning model. The Fine-Tuned model that used RBF Kernel actually has a surprising decrease in AUC compared to the linear baseline. This leads us to believe that the non-linear relationship may have led to some overfitting.

CHAPTER 3: Conclusion

3.1 Summary

In our study, we analyzed the Online Shoppers Purchasing Intention Dataset to determine which session features predict a transaction. Through means of Explanatory Data Analysis and Feature Selection, we can conclude that the engagement metrics `PageValues` and `ExitRates` were the most significant indicators of a purchase.

We chose four supervised learning algorithms: K-Nearest Neighbors, Logistic Regression, Random Forest, and Support Vector Machines. Our Fine-Tuned RF classifier performed the highest with an AUC of 0.93 and a Weighted F1 score of 0.91. However, this came at a high computational cost as our model was fine-tuned for over 80 minutes. However, Logistic Regression performed very well with an AUC score of 0.90 with negligible training time. Our KNN model excelled best with evaluating positive cases with a Recall of 0.82, which would minimize the missed sales. SVM matched the Logistic Regression's AUC of 0.90, but also has a high computational cost, with the fine-tuning model taking around 30 minutes to train.

While the RF model is the most powerful predictor in terms of performance metrics, **Logistic Regression** remains the most practical choice for real-time analysis due to its optimal balance of metrics and low computational cost.

3.2 Applications

The model's aptitude for identifying when a session will result in no sale can be utilized for targeted advertising. During a session, the logistic model's speed may allow for live evaluations of a user's session. If a session is predicting a "No Purchase" targeted advertisements and coupons can be used to entice customers during a session to make a purchase. This can also be used to create follow up evaluations to determine individualized coupons. Possible triggers include:

- High Product Page Duration early in the session can trigger a modest coupon being sent as a follow up.
- Holiday triggers emphasized in November.
- Cluster labels to adapt targeted intervention and ensure coupons are being used sparingly.
- Low Page Values & High Exit Rates could indicate that a sale cannot be achieved. This could allow for further testing of strength of coupon, User Interface modification, and possibly trigger a chatbot to assist returning customers in the moment.

3.3 Limitations and Future Work

The largest limitation stemmed from the dataset's anonymity, which limited the potential of the interpretability of behavioral patterns. It required extreme caution when drawing conclusions about real customers. Additional contextual identifiers or metadata would have allowed for clearer insights. Additionally, computational capacity was a major limiting factor when modeling. More robust models like our random forests and SVMs required long training times, restricting the extent of hyperparameter tuning and likely preventing these methods from reaching their full potential. For the same reason, neural network approaches were avoided, as their training demands exceeded the available resources.

Future extensions of this project could involve evaluating ensemble methods such as bagging, blending, and boosting. These methods may capture more complex interactions than the baseline models considered here, and even involve some of our baseline models in the process. Additional data, especially across multiple years, would help reveal stronger seasonal trends and improve generalization. Despite these limitations, this project offered tremendous learning value, demonstrating how supervised models can meaningfully interpret shopper behavior. This project helped shape for us the practical considerations that real-world web analytics work involves.

CHAPTER 4: LIST OF REFERENCES

[1] C. O. Sakar, S. O. Polat, M. Katircioglu, and Y. Kastro, “Real-time prediction of online shoppers’ purchasing intention using multilayer perceptron and lstm recurrent neural networks,” *Neural Computing and Applications*, vol. 31, pp. 6893 – 6908, 2018.