Prediction Challenge Analysis Report

For this project, the objective was to develop a predictive model that can determine whether a customer would make a purchase or not based on various product and pricing data. Overall, my goal was to achieve a model as accurate as possible while retraining its reliability in prediction.

The first step in my plan was to preprocess the currency data to ensure it would be compatible with both the training and testing datasets. I extracted the Currency and Amounts using regular expressions, specifically parsing the discount_price and actual_price columns in both the training and testing datasets. This involved extracting the currency codes (e.g. USD) and the numerical amounts from each of their columns. I also reformatted the data slightly by removing commas, and then converted all the amounts to floats for consistency. Next, I converted each currency by calculating conversion rates through dividing 100 by the 100_USD_worth values from the currency dataset. These rates were then applied to convert both discount_price and actual_price from their original currencies to USD. I also dropped the conversion_rate column after converting, as they were no longer needed. To address missing values, I assigned each missing entry of any numerical column with their respective median values. For categorical columns, I filled missing entries with 'Unknown'.

To enhance the model's predictive capabilities, I added several new characteristics. This is known as feature engineering and allows for additional calculated parameters to be used by the models. For instance, I calculated the discount percentage and discount amount to quantify the value that discounts have on customers' decisions. The discount percentage was calculated by the percent difference between actual_price_usd and discount_price_usd. Similarly, the discount amount is simply the USD difference between acutal_price_usd and discount_price_usd. This is helpful as it indicates the specific value of the discount per purchase. Additionally, I created a weighted rating by combining product ratings with the number of ratings, which helps account for both the quality and quantity of customer feedback. I achieved this by using the formula rating_value = $x + a * y^2$, where x is the ratings column, y is the number of ratings, and a is a constant. 'a' represents the weight of the ratings on the final weighted rating, so a value of 0.1 allows for the ratings to contribute to the weight without overshadowing the original rating. Lastly, I categorized the price after discount into 'Low', 'Medium', and 'High' ranges to facilitate analysis across different price segments. Specifically, I used <20 USD as low, 20-100 USD as medium, and >100 USD as high. These engineered features provide the model with broader information, allowing it to capture additional patterns related to customer purchasing behavior.

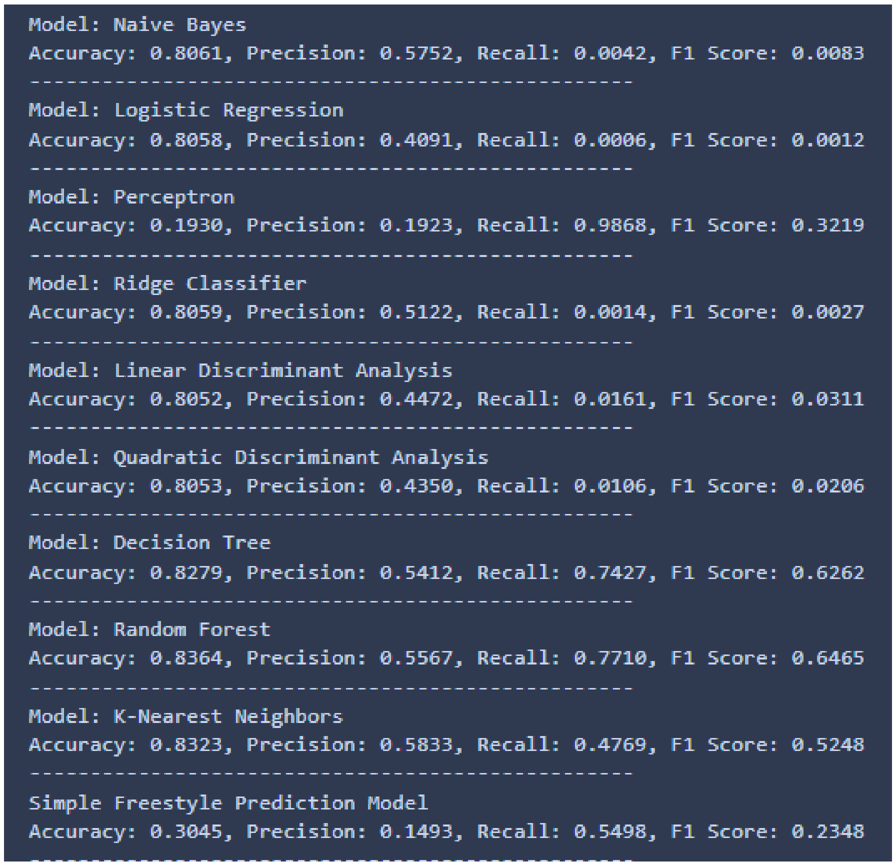
Next, I wanted to achieve a way to efficiently apply a variety of models to the datasets, so that it is easier to identify the best-performing one. The models I selected are meant for both simple and complex datasets to ensure a broad evaluation. Overall, the models included: Naive Bayes, Logistic Regression, Perceptron, Ridge Classifier, Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Decision Tree (which uses Gini Impurity as the splitting criterion by default), Random Forest, and K-Nearest Neighbors. GPT was able to help me create a method to rapidly test most of these models at once, grabbing run algorithms from a variety of public Python libraries. These are seen in the list of import statements at the top of my code.

One specific change I had to incorporate was hyperparameter tuning. This was needed on models where any added parameters impacted performance, especially seen in the Decision Tree and Random Forest models. With the help of GPT, I used a grid search with cross-validation via the GridSearchCV library to identify the best parameter settings for each model. Parameters such as n_estimators, max_depth, min_samples_split, and class_weight were varied and tested to enhance overall model performance. For example, I found that increasing the number of trees in the Random Forest (the n_estimators paramter) and adjusting the maximum depth of the trees (increasing max_depth) helped improve overall accuracy.

Finally, I needed a way to evaluate the accuracy of each model. To do this, I split the results of each model into certain categories. The first is the overall accuracy, which is the ratio of correct predictions out of all predictions made. The second is precision, which is the ratio of true positive predictions to the total number of predictions. The third is recall, which is the ratio of true positive predictions to the actual number of positives. The last is the F1 score, which was recommended to use by GPT, and is simply the average of the precision and recall scores. The result of each model is displayed below:

Overall, the results show that the Random Forest model emerged as the best performer with an accuracy of 83.64%, precision of 55.67%, recall of 77.10%, and an F1 score of 0.6465. The high recall value shows that the model has a good ability to identify actual purchases, thus minimizing missed opportunities.

While Random Forest slightly outperforms the Decision Tree in most metrics, the Decision Tree remains a very strong model. Note again that the criterion for splitting trees in the



```
Model: Naive Bayes
Accuracy: 0.8061, Precision: 0.5752, Recall: 0.0042, F1 Score: 0.0083
--------------------------------------------------
Model: Logistic Regression
Accuracy: 0.8058, Precision: 0.4091, Recall: 0.0006, F1 Score: 0.0012
--------------------------------------------------
Model: Perceptron
Accuracy: 0.1930, Precision: 0.1923, Recall: 0.9868, F1 Score: 0.3219
--------------------------------------------------
Model: Ridge Classifier
Accuracy: 0.8059, Precision: 0.5122, Recall: 0.0014, F1 Score: 0.0027
--------------------------------------------------
Model: Linear Discriminant Analysis
Accuracy: 0.8052, Precision: 0.4472, Recall: 0.0161, F1 Score: 0.0311
--------------------------------------------------
Model: Quadratic Discriminant Analysis
Accuracy: 0.8053, Precision: 0.4350, Recall: 0.0106, F1 Score: 0.0206
--------------------------------------------------
Model: Decision Tree
Accuracy: 0.8279, Precision: 0.5412, Recall: 0.7427, F1 Score: 0.6262
--------------------------------------------------
Model: Random Forest
Accuracy: 0.8364, Precision: 0.5567, Recall: 0.7710, F1 Score: 0.6465
--------------------------------------------------
Model: K-Nearest Neighbors
Accuracy: 0.8323, Precision: 0.5833, Recall: 0.4769, F1 Score: 0.5248
--------------------------------------------------
Simple Freestyle Prediction Model
Accuracy: 0.3045, Precision: 0.1493, Recall: 0.5498, F1 Score: 0.2348
--------------------------------------------------
```

Decision Tree model was Gini Impurity, which is the default choice in scikit-learn's DecisionTreeClassifier. Both Decision Trees and Random Forests are tree-based models, but Decision Tree uses a single tree structure that makes decisions based on feature splits to classify data, whereas a Random Forest is a collection of multiple Decision Trees. Random Forest builds several trees using different subsets of the data and features, then aggregates their predictions to make a final decision. The reason I believe that Random Forest performed particularly well was due to this approach, which significantly improves accuracy and robustness by reducing overfitting and variance. Random Forests are usually more accurate than single Decision Trees in general, as evidenced by their performance in this project.

All in all, I learned from this project that while simpler models like Naive Bayes provided some tangible level of accuracy, their recall rates were significantly lower, making them less reliable for identifying true purchases. In contrast, the tree models not only achieved a higher accuracy but also maintained a balanced precision and recall, resulting in a much higher F1 score.