DATA ANALYSIS REPORT

‘CLICK TO CART’

(A PREDICTIVE MODELING TASK)

Souptik Chattaraj 24CH10030

KODEINKGP IITKGP

Model Summary

- PROBLEM:

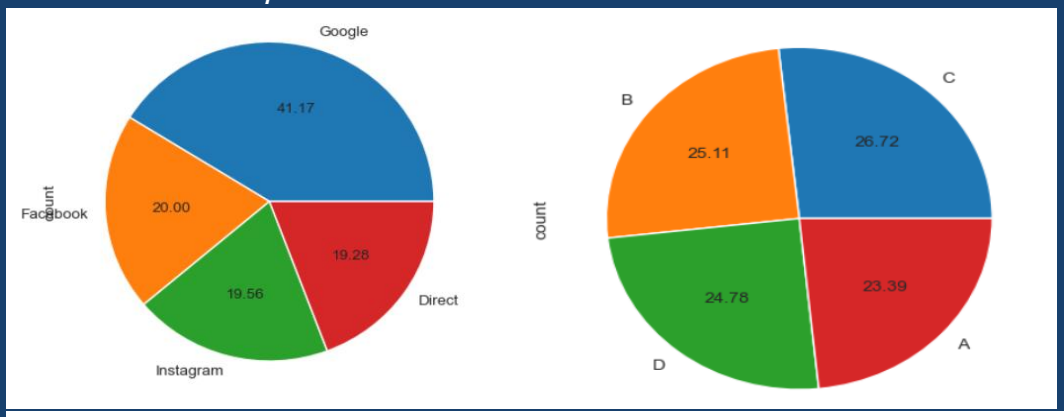
  Our task consists of two csv files which have a total of 2,250 rows in 8 columns. It is the data of an e-shopping website and the columns store information regarding time spent, pages viewed, referral, how many items the customer added to cart and so on, and finally the target column: Purchased or not. Based on this data, our model needs to predict whether the customer will make a purchase (1) or not (0). It is a standard binary classification problem.

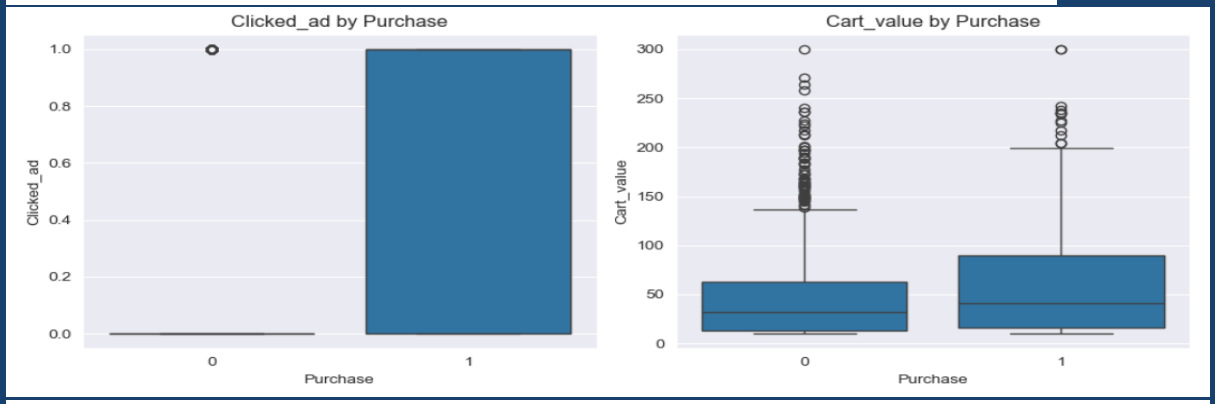
	Time_on_site	Pages_viewed	Clicked_ad	Cart_value	Referral	Browser_Refresh_Rate	Last_Ad_Seen	Purchase
753	3.29	10.96	1	53.38	Google	102.65	B	0
1203	7.47	21.94	0	19.20	Direct	79.54	B	0
656	5.93	14.43	1	84.08	Google	47.36	C	1
355	4.02	8.84	0	24.59	Direct	59.58	C	0
951	3.05	7.38	0	158.09	Google	119.67	B	0

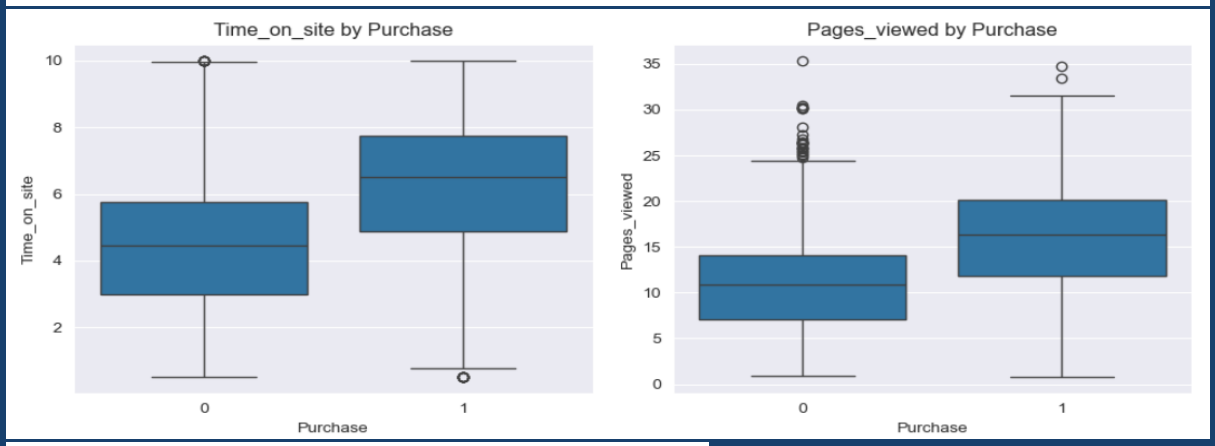
- EXPLORATORY DATA ANALYSIS:

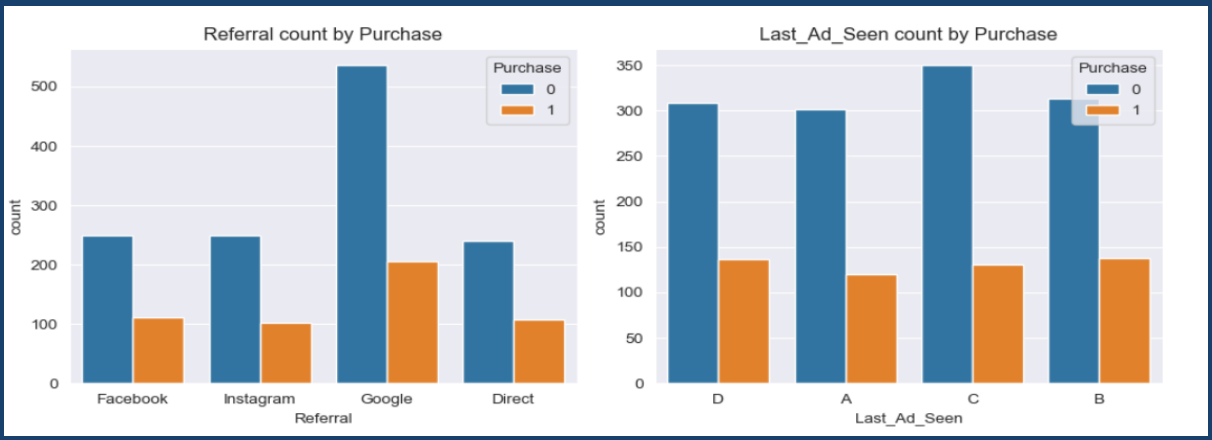
  The next step is to gain an understanding of our data. To this end, I have found out the shapes, mathematical features, null values, etc. of the data. Our data has 5 numerical columns and 2 categorical columns excluding our target column. I have done a box-plot of my numerical columns and count-plots of my categorical columns with regards to target. Some columns like cart value and pages viewed showed a large number of

outliers. Naturally, the number of Purchase is less than Not Purchase.



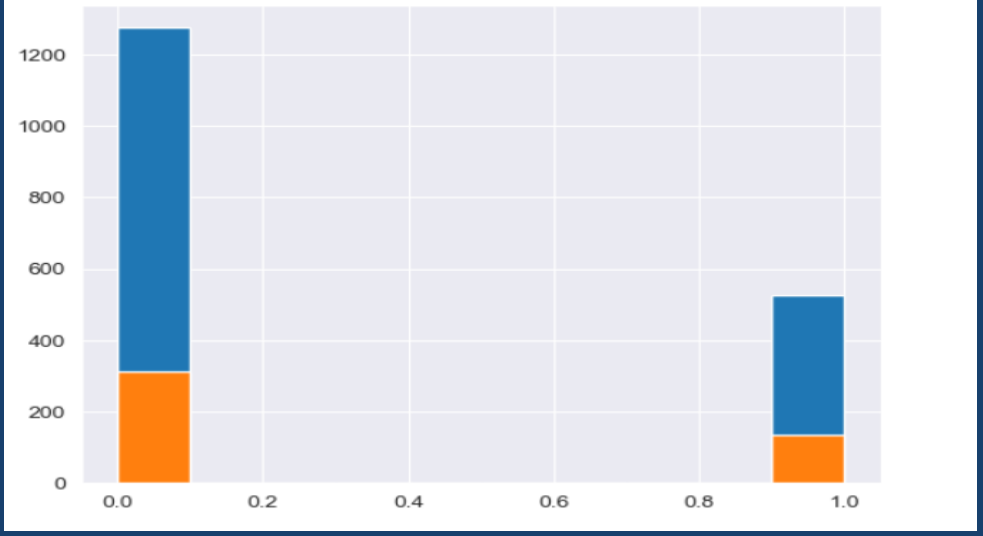






I next focused on my target data to see if there was any imbalance. Fortunately, there was a 70:30 class split in both training and test data, so I decided against SMOTE, Oversampling, Undersampling etc.

```
Classes in training data=  Purchase
0    0.707778
1    0.292222
Name: proportion, dtype: float64
Classes in test data =  Purchase
0    0.697778
1    0.302222
Name: proportion, dtype: float64
```



I also tried to find the correlation of my columns with respect to my target. To this end, I used chi-square and p-value metrics on my data. The column 'Browser_Refresh_Rate' showed very low correlation, whereas the categorical columns showed high p-values.

```
Correlation between Time_on_site and target: 0.3441, p-value: 0.0000
Correlation between Clicked_ad and target: 0.2839, p-value: 0.0000
Correlation between Pages_viewed and target: 0.3590, p-value: 0.0000
Correlation between Cart_value and target: 0.1379, p-value: 0.0000
```

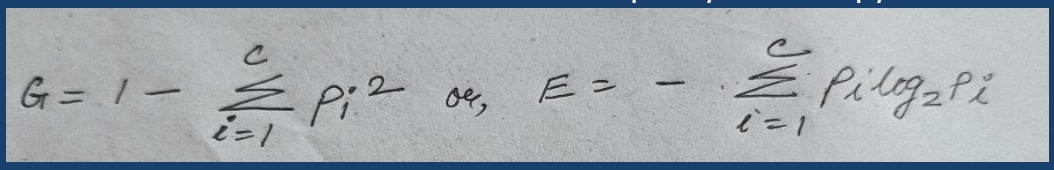
```
Chi-square test for Referral: p-value = 0.6243
Chi-square test for Last_Ad_Seen: p-value = 0.5977
```

- **FEATURE ENGINEERING:**

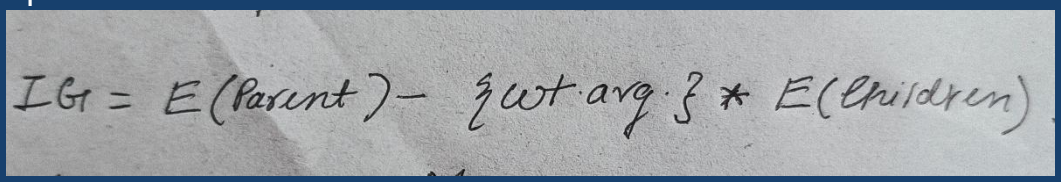
  I now began my data preprocessing steps. Having dropped the 'Browser_Refresh_rate' column due to -0.005 correlation, I split my data into training and test input and output. I kept my categorical columns in spite of high p-values because I felt they may have a not-so-apparent effect on data. I applied Column Transformer and inside it did One-Hot Encoding on the 2 categorical columns. I dropped the first one-hot encoded column to escape the dummy variable trap. I decided against removing outliers from my two most spread-out columns, because I felt it gives an impression that placing items in carts rarely means a confirmed Purchase. Since I used a Random Forest Classifier and XgBoost, the impact of outliers is not much. I also did not do standardization or normalization, because I did not use any distance-based algorithm that needed scaled values of dimensions. Besides, my data had no null values, so there was no requirement of standard missing value imputation techniques (Simple Imputer, KNN, MICE etc). As this was an exploratory project and not intended for deployment yet, I applied transformations manually instead of wrapping them in a Pipeline.

```python
#Applying Encoding for categorical columns

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

trf = ColumnTransformer(transformers=[
    ('trf1',OneHotEncoder(handle_unknown='ignore',sparse_output=False),['Referral']),
    ('trf2',OneHotEncoder(handle_unknown='ignore',sparse_output=False),['Last_Ad_Seen'])
    ], remainder='passthrough'
)



x_train_trans=trf.fit_transform(x_train)
x_test_trans=trf.transform(x_test)
```

- **MODEL SELECTION AND TRAINING:**
  - Random Forest Classifier is an ensemble learning algorithm with a group of decision tress employing bagging techniques. Its basic logic is: a number of Decision Trees are clubbed together, each of which is given a subset of training data and gives corresponding output. What RF does is it combines these Low Bias High Variance models into a Low Bias Low Variance model, which balances both overfitting and underfitting. Each tree is trained on a random subset of the data with replacement, and during tree construction, a random subset of features is chosen at each split. I used it because it is robust to noise, outliers, deals with high-dimensional data (10 columns).
  - Each DT in RF tries to minimise its Gini impurity or entropy:



  - Splits are made based on Information Gain:



  - Once trees are trained, prediction is made via majority voting across all trees:

$$\hat{y} = \underset{c \in C}{\arg\max} \sum_{m=1}^{M} 1(T_m(x) = c)$$

Where:

- T(m(x)): prediction of the m^{th} decision tree
- M: number of trees
- C: set of all classes
- 1: indicator function (1 if tree votes class c, 0 otherwise)

Xg Boosting (Extreme Gradient Boosting) is not actually an algorithm, it is a library based on Gradient Boosting. It has a large number of features: cross-platform, multi-language support, integrated with other libraries, parallel processing, GPU support etc. It consists of a preliminary model which computes the mean of all outputs and then several decision trees which successively operate on the same dataset to further optimize the model. It has also L1 and L2 regularization, handling missing values, sparsity awareness, highly-scalable etc. It is one of the most popular and widely used ML models.

XgBoost uses an objective function which it tries to minimise using gradient boosting techniques:

$$O = \sum L(y_i, \hat{y}_i) + \sum \Omega(f_k)$$

$$\Rightarrow \Omega(f) = \gamma T + \frac{1}{2} \lambda \sum w_j^2$$

For a potential split that divides data into left (L) and right (R) child nodes:

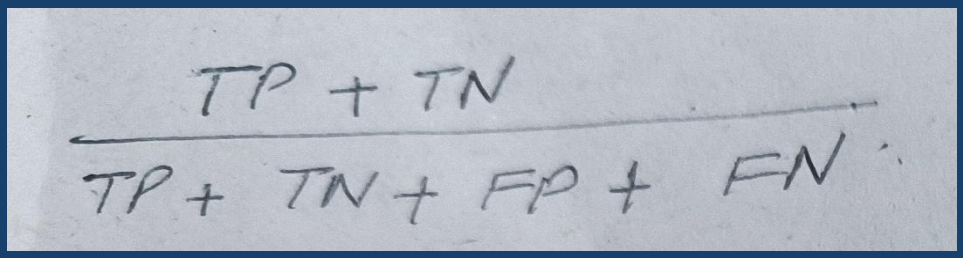
$$G_1 = \frac{1}{2}\left(\frac{g_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L^2 + G_R)^2}{H_L + H_R + \lambda} \right) - \gamma$$

Where:

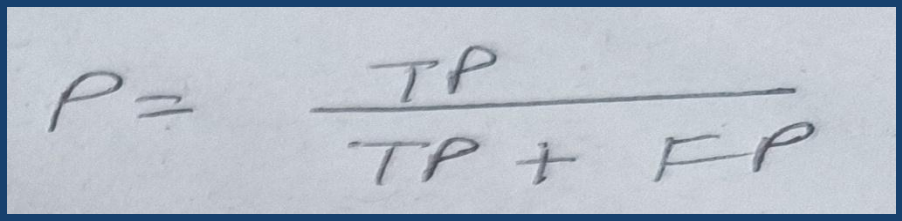
- GL, GR: Sum of first-order gradients in the left and right nodes
- HL, HR: Sum of second-order gradients (Hessians) in the left and right nodes
- λ: L2 regularization term on leaf weights
- γ: Penalty for creating a new leaf (controls tree complexity)

METRICS:

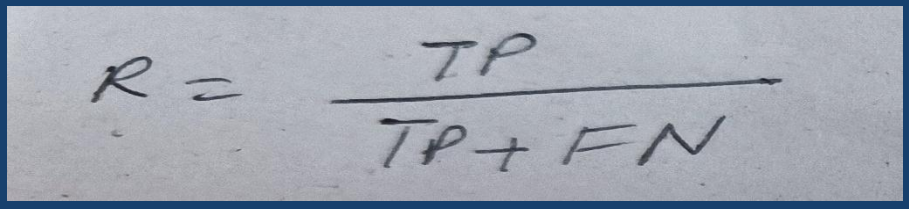
- Accuracy score: The ratio of correctly predicted samples to the total number of samples. Good for balanced datasets; misleading if classes are imbalanced.
- Formula:



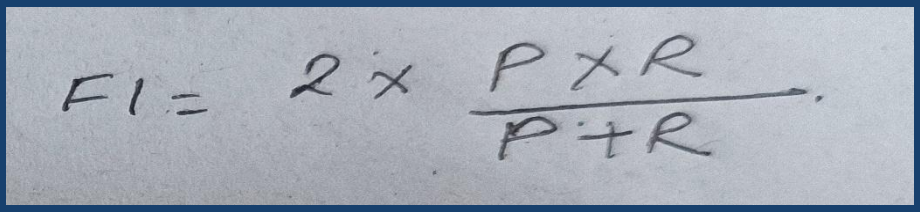
- Precision: The proportion of positive predictions that are actually correct. Important when false positives are costly (e.g., spam detection).
- Formula:



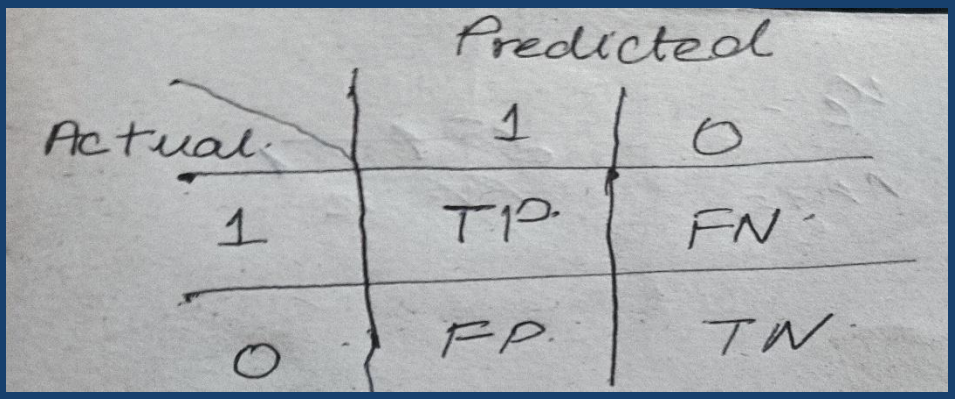
- Recall: The proportion of actual positives that were correctly predicted. Important when false negatives are costly (e.g., cancer detection).
- Formula:



- F1 score: The harmonic mean of precision and recall; balances the two. Good for imbalanced datasets or when both precision and recall matter.
- Formula:



- Confusion matrix: A 2×2 (for binary) table showing counts of predictions vs. actual classes. Helps visualize all types of errors.
- Formula:

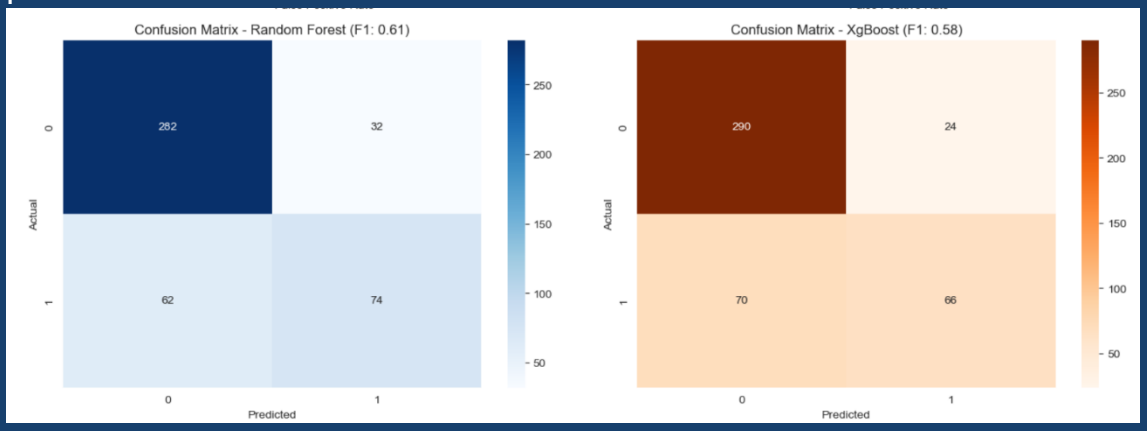


- ROC (Receiver Operating Characteristic) Curve: Plots True Positive Rate (Recall) vs. False Positive Rate at different thresholds. To see how well the model distinguishes between classes. A perfect classifier reaches the top-left corner.
- AUC (Area Under Curve) Score: A scalar value from 0 to 1 representing the ROC curve's area. The various values are interpreted as: 1(perfect classifier), 0.5-1(better than random), 0.5 or less(random).

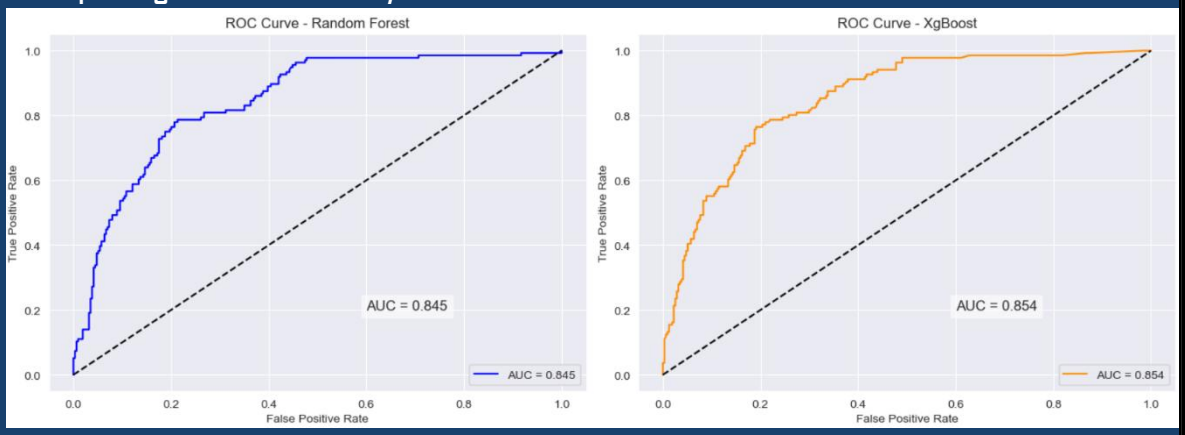
OUR PERFORMANCE ANALYSIS (BASED ON SINGLE-SPLIT):

- The following points are evident from the notebook:
- The RF Classifier has an accuracy of 79.11 and XGBoost Classifier 79.11. This implies XGBoost and RF perform equally well.

On the basis of the confusion matrices, Random Forest has a higher True Positive Rate (better at identifying buyers: 0.5441 vs. 0.4853), whereas XGBoost has a lower False Positive Rate (better at avoiding false alarms: 0.0764 vs. 0.1019). Now, assuming that in typical business contexts most people do not buy, a company would prefer to correctly identify those who are likely to buy. Hence, the lower false positive rate of XGBoost is more valuable here.



- From a comparative study of the ROC curves and AUC values, XGBoost is better at classifying random customers into prospective buyers or non-buyers than RF (0.854 against 0.845) although both are quite good individually.



CONCLUSION AND INSIGHTS:

Metric	Random Forest	XGBoost
Accuracy (%)	79.11	79.11
True Positive Rate	0.5441	0.4853
False Positive Rate	0.0764	0.1019
F1 score	0.61	0.58

AUC score	0.845	0.854

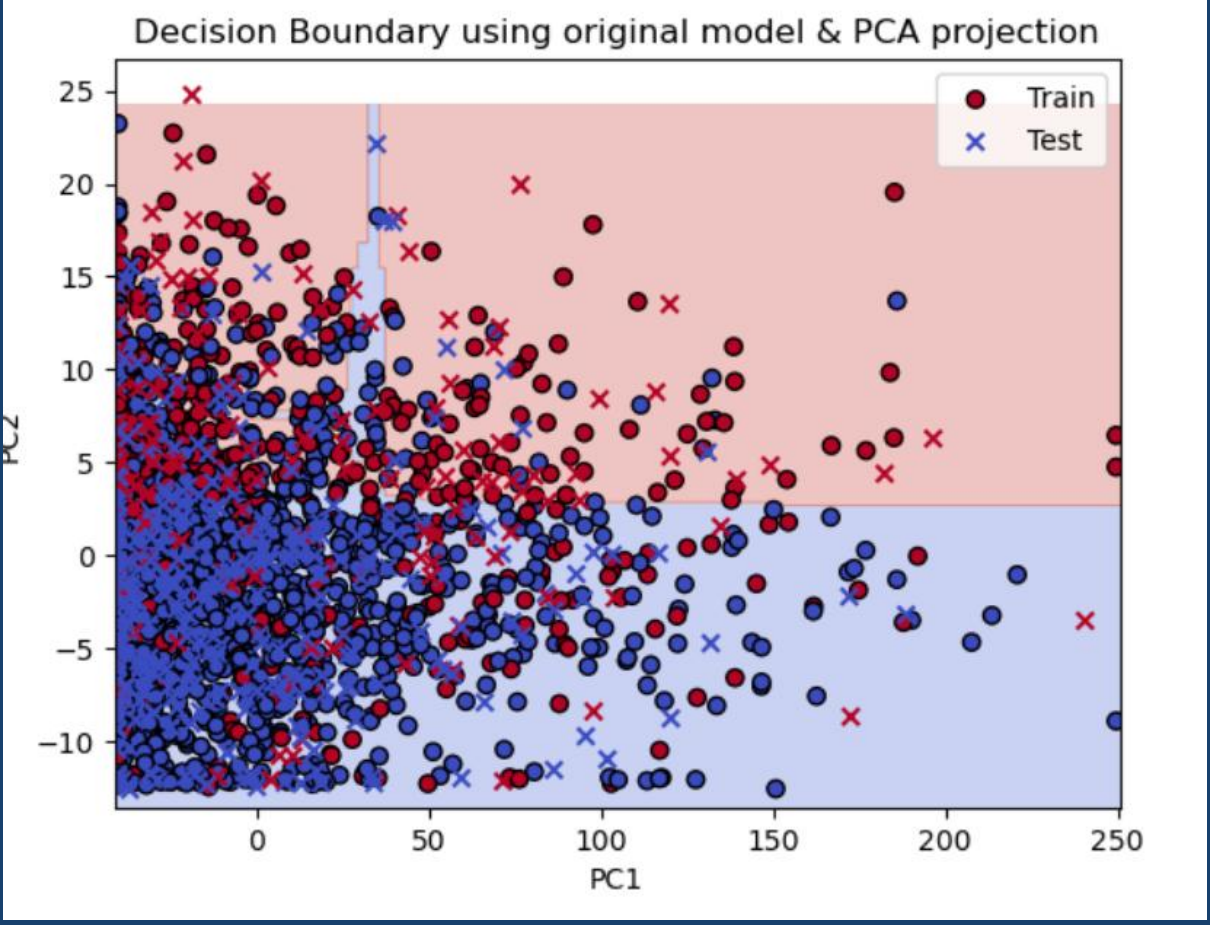
Based on the results above, XGBoost demonstrates superior performance compared to Random Forest across vital evaluation metrics — accuracy, F1 score, and AUC. These results indicate that XGBoost is the more effective and reliable model for this classification task. Considering both empirical performance and theoretical strengths, XGBoost is recommended as the preferred model for deployment or further tuning in this project.

Typically, it is seen that more often than not, XGBoost outperforms traditional models. XGBoost likely excelled due to:

- Its boosting mechanism, which focuses on correcting previous errors,
- Built-in regularization, helping prevent overfitting,
- Fine-tuned optimization, making it more sensitive to subtle patterns in the data.

DECISION BOUNDARY OF THE BETTER MODEL:

Since my data was in 10 dimensions, it is not possible to represent it without using dimensionality reduction techniques. To that end, I used Principal Component Analysis (PCA with 2 components) and tried to plot my decision boundary for XGBoost Classifier model. PCA is a technique that reduces dimensionality of data by transforming it into a set of axes (principal components) that capture maximum variance.



This plot shows that the decision boundary is non-linear and complex. The training (brown O) and test (blue X) are scattered across both regions, showing XGBoost performed a learned decision rule. Most training points lie in their expected region and show good fit. However, in low dimensions, the data appears a lot cluttered-up and most likely there is some overfitting as evident from the very small regions that have been ear-marked.

HYPERPARAMETER TUNING:

I have tuned the following hyperparameters using Randomized Search:

- Random Forest Classifier

n_estimators=200: (RandomForest) Number of trees in the forest.

max_depth=8: (RandomForest) Maximum depth of each decision tree.

min_samples_split=6: Minimum number of samples required to split an internal node.

criterion='log_loss': Function to measure the quality of a split using logarithmic loss.

- XGBoosting Classifier

objective=binary:logistic': Sets XGBoost to perform binary

classification by outputting the class with logistic probability.

max_depth=3: Limits the maximum depth of each tree to prevent overfitting.

eta=0.1: Learning rate; controls how much each tree contributes to the final prediction.

n_estimators=100: Number of boosting rounds or trees to be built (XGBoost-specific).

subsample=0.8: Fraction of training data used for building each tree to prevent overfitting.

colsample_bytree=0.8: Fraction of features used per tree to improve generalization.

use_label_encoder=False: Disables the automatic label encoding in newer XGBoost versions.

eval_metric='logloss': Evaluation metric set to binary-class log loss.

Future improvements could include automated hyperparameter tuning via GridSearchCV or using a validation curve to avoid overfitting.