Data Analysis Report

Click To Cart

A predictive modeling task

Name - Roshni Laha

Roll no. - 24IE10002

EDA : Visualization of data and correlation with target column

Features in the data :

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1800 entries, 0 to 1799
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Time_on_site          1800 non-null   float64
 1   Pages_viewed          1800 non-null   float64
 2   Clicked_ad            1800 non-null   int64  
 3   Cart_value            1800 non-null   float64
 4   Referral              1800 non-null   object 
 5   Browser_Refresh_Rate  1800 non-null   float64
 6   Last_Ad_Seen          1800 non-null   object 
 7   Purchase              1800 non-null   int64  
dtypes: float64(4), int64(2), object(2)
memory usage: 112.6+ KB
```

Fig1 : Showing column names, count, and Data type

The dataset has :

- 4 columns of float type
- 2 columns of int type (including target column 'Purchase')
- 2 columns of object type

There are no null values in any column

```
df.isnull().sum()
```

	0
Time_on_site	0
Pages_viewed	0
Clicked_ad	0
Cart_value	0
Referral	0
Browser_Refresh_Rate	0
Last_Ad_Seen	0
Purchase	0

dtype: int64

Fig2 : Showing count of null values in each column

There are no duplicate rows

```
df.duplicated().sum()
```

```
np.int64(0)
```

Fig3 : Showing there are no duplicate rows

Plots

```
sns.scatterplot(x=df['Time_on_site'],y=df['Pages_viewed'],hue=df['Purchase'])
```

```
<Axes: xlabel='Time_on_site', ylabel='Pages_viewed'>
```

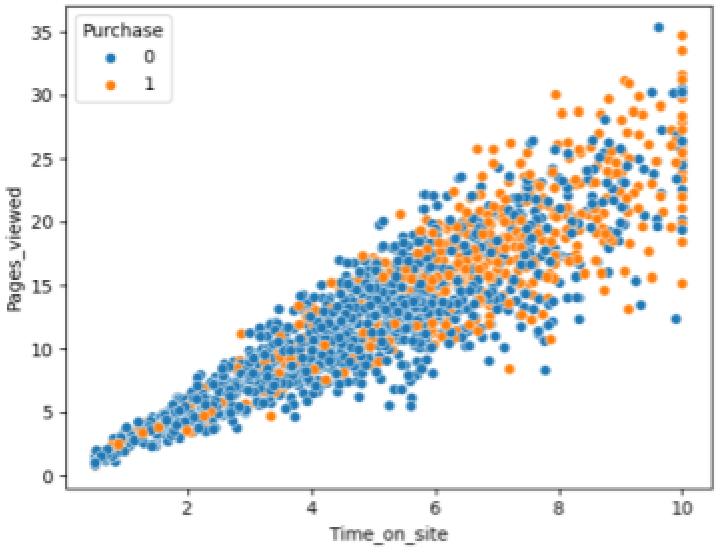

Fig4 : Scatterplot showing linear relationship between <u>Time on site and Pages Viewed</u>

```
sns.boxplot(x='Clicked_ad',y='Pages_viewed',data = df,hue='Purchase')
plt.xlabel('Clicked_ad')
plt.ylabel('Pages_viewed')
plt.show()
```

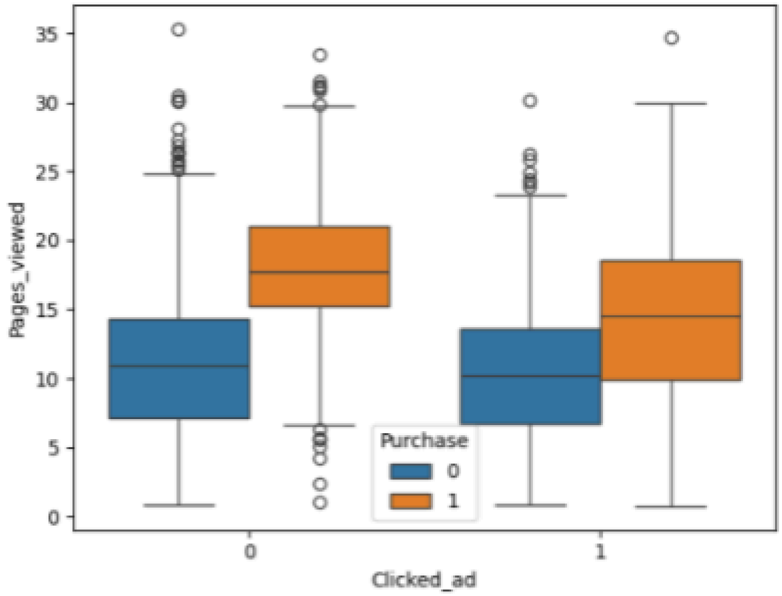

Fig5 : Box Plot between Pages viewed and Clicked ad

```
[15] sns.displot(data=df,x='Time_on_site',hue='Purchase',kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f7e652f6890>
```

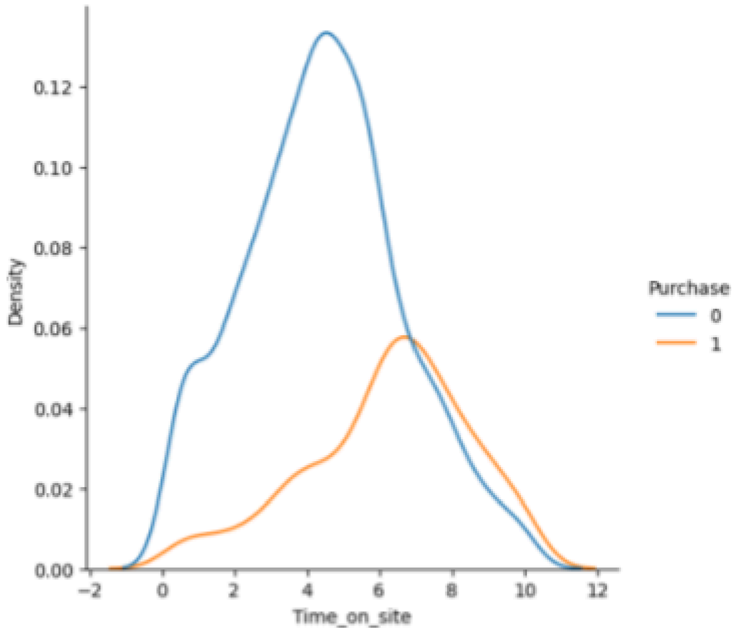

Fig6 : KDE plot of Time on site

```
sns.displot(data = df, x='Cart_value',hue='Purchase',kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f7e66e33f50>
```

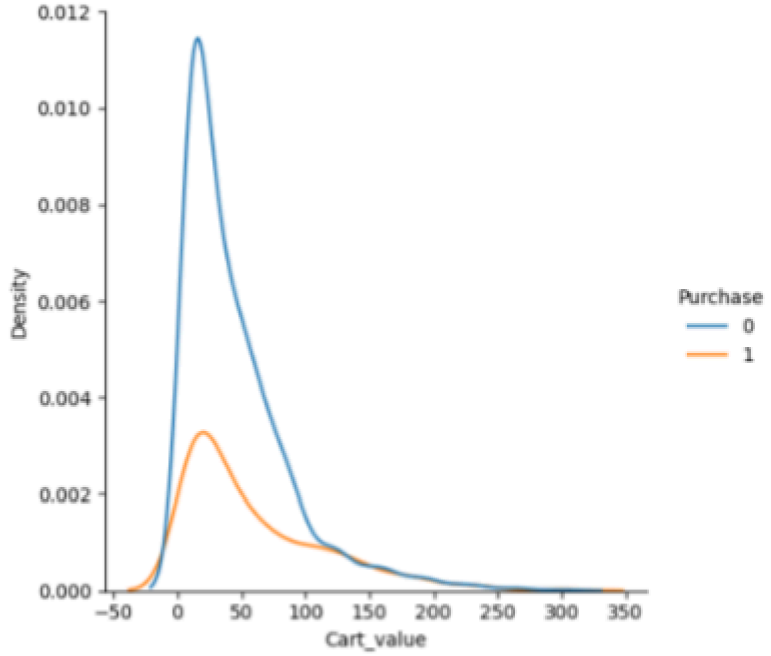

Fig7 : KDE plot of cart value

```
sns.displot(data=df,x='Browser_Refresh_Rate',hue='Purchase',kind='kde')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f7e5ee53cd0>
```

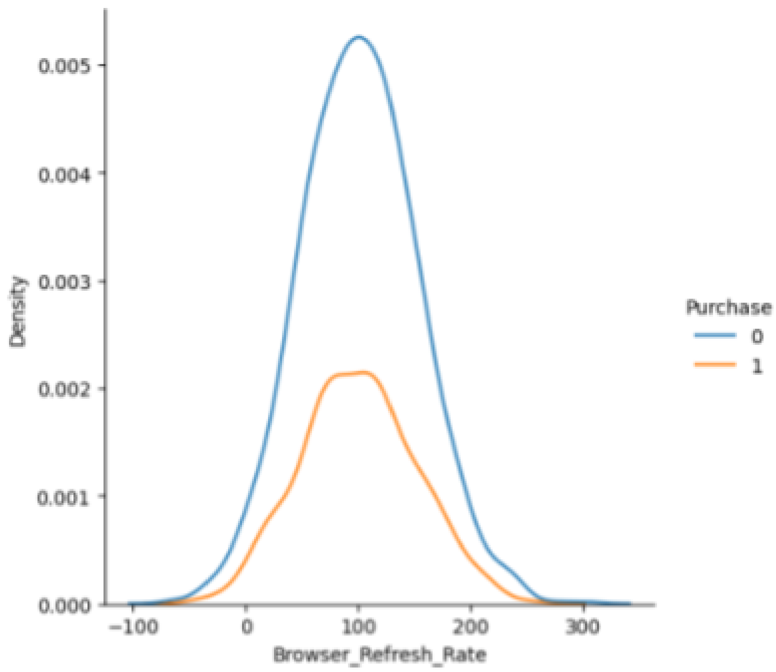

Fig8 : KDE plot of Browser Refresh Rate

Categorical columns

```
[31] df['Referral'].value_counts().plot(kind='pie',autopct='%.2f')
```

```
<Axes: ylabel='count'>
```

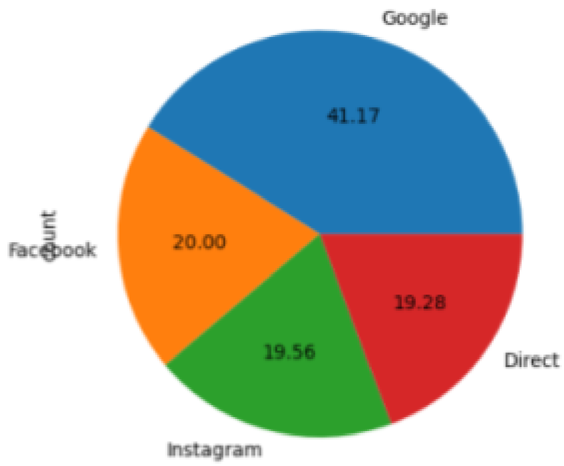

Fig9 : Showing percentage of referrals

```
[117] df_refers
```

Referral	Referral_count	Referral_count_for_purchase_true	Percentage_who_purchased
Direct	347	107	30.84
Facebook	360	111	30.83
Google	741	205	27.67
Instagram	352	103	29.26

```
[138] df_refers['Percentage_who_purchased'].plot(kind='pie')
```

```
<Axes: ylabel='Percentage_who_purchased'>
```

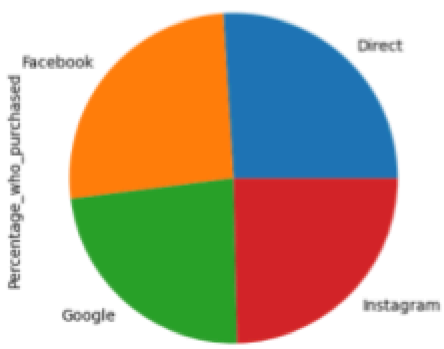

Fig10 : Showing percentage of people of each category who purchased

```
df['Last_Ad_Seen'].value_counts().plot(kind='pie',autopct='%.2f')
```

```
<Axes: ylabel='count'>
```

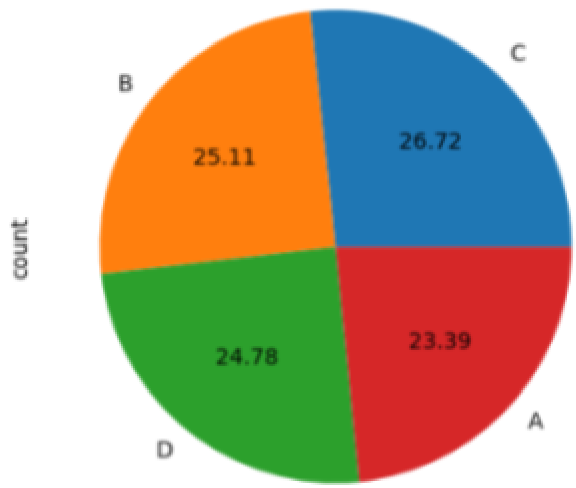

Fig11 : Showing percentage of each category of last ad seen

```
[120] df_last_ads
```

Last_Ad_Seen	Last_ads_count	Last_ads_count_for_purchase_true	Percentage_who_purchased
A	421	120	28.50
B	452	138	30.53
C	481	131	27.23
D	446	137	30.72

```
df_last_ads['Percentage_who_purchased'].plot(kind='pie')
```

```
<Axes: ylabel='Percentage_who_purchased'>
```

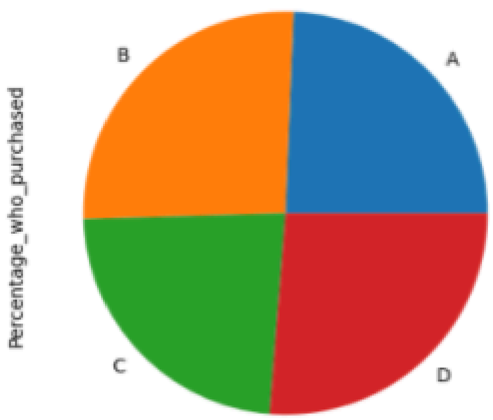

Fig12 : Showing percentage of each category who purchased

Correlation with Target Column

Purchase	count
0	1274
1	526

dtype: int64

```
df['Purchase'].value_counts(normalize= True).plot(kind='pie',autopct='%1.1f%%')
```

```
<Axes: ylabel='proportion'>
```

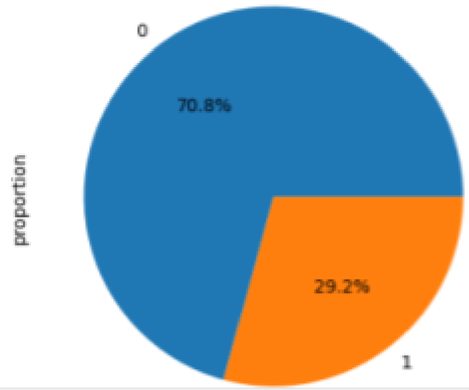

Fig13 : Showing the count of target variable - Purchase

```
sns.boxplot(x='Purchase',y='Time_on_site',data = df)
plt.xlabel('Purchase')
plt.ylabel('Time_on_site')
plt.show()
```

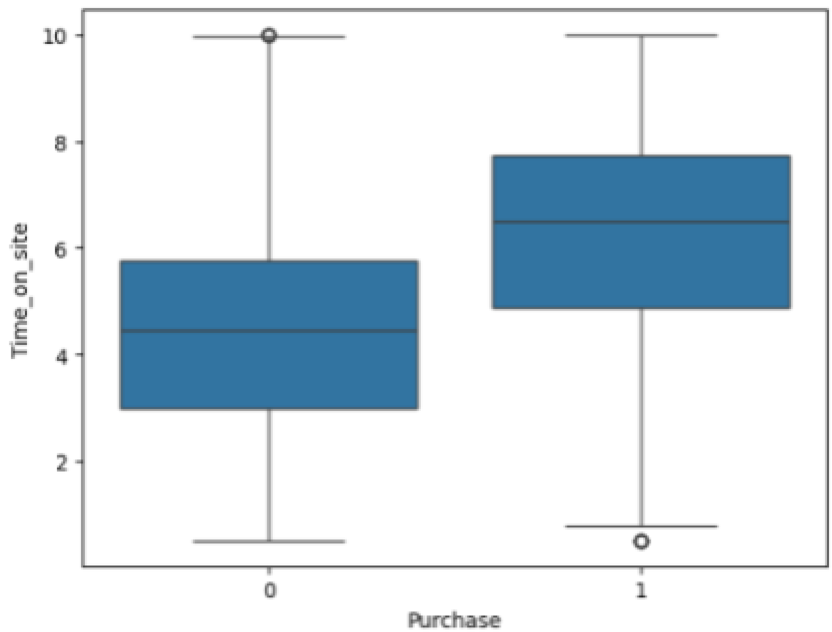

Fig14 : Box Plot between Purchase and Time on site

```
sns.boxplot(x='Purchase',y='Pages_viewed',data = df)
plt.xlabel('Purchase')
plt.ylabel('Pages_viewed')
plt.show()
```

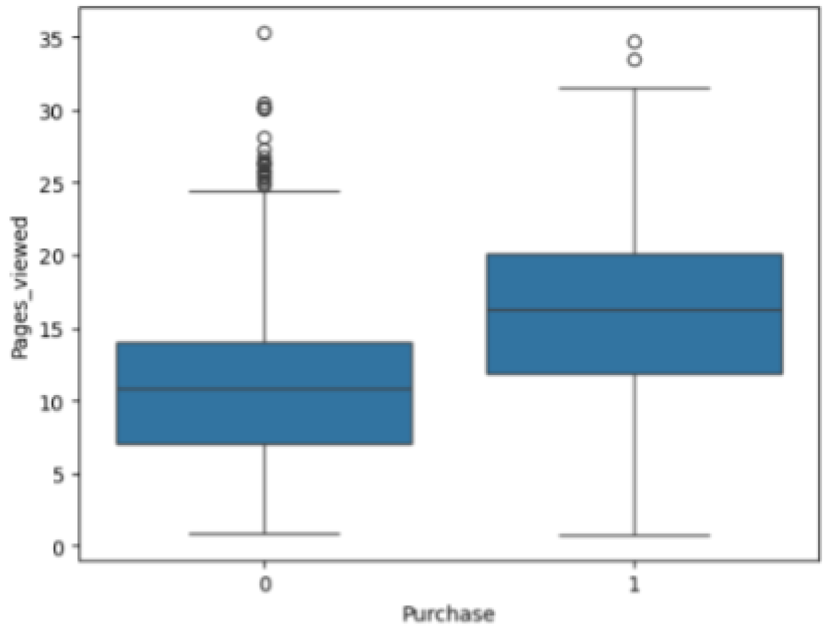

Fig15 : Box plot between Purchase and Pages viewed

```
sns.boxplot(x='Purchase',y='Browser_Refresh_Rate', data=df)
plt.xlabel('purchase')
plt.ylabel('Browser_Refresh_Rate')
plt.show()
```

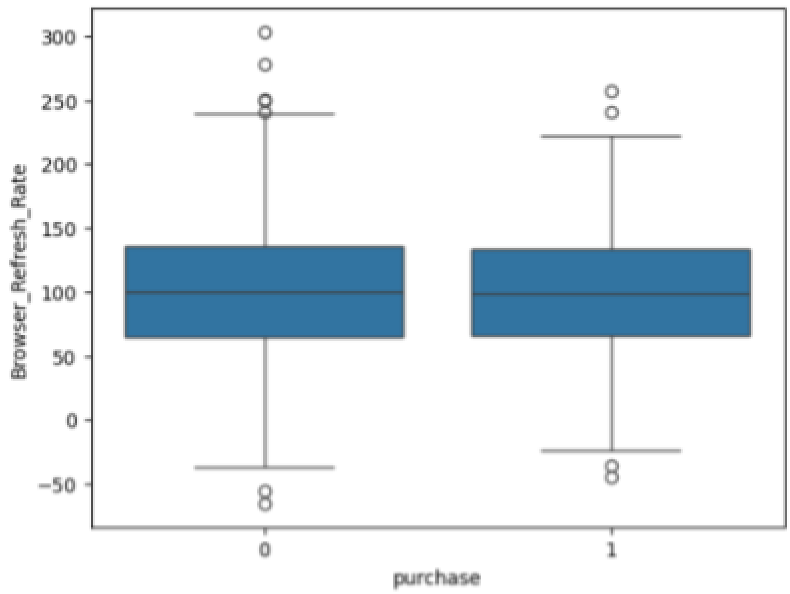

Fig16 : Box Plot between Purchase and Browser Refresh Rate

Key Observations :

- ☐ The dataset exhibits a class distribution of approximately 70% for Purchase = 0 and 30% for Purchase = 1. (**slightly imbalanced**)
- ☐ Time on site and pages viewed are linearly related (*Fig 4*).
- ☐ People who have purchased have more **Time on site** and more **Pages viewed** (*Fig 6 , Fig 14 , Fig 15*).
- ☐ **Browser refresh rate** is a not so important feature for the target column as seen from the box plot (*Fig 8, Fig 16)*.
- ☐ Google has the largest (41.17%) referral.
    - ☐ Nearly 30% of people from each **Referral** category have made a purchase (*Fig 10*) .
- ☐ All categories of Last ad seen have nearly equal (25%) count.
    - ☐ Nearly 30% of people from each **Last ad seen** category have made a purchase (*Fig 12*).
- ☐ Column Transformations :
    - ☐ **Browser_Refresh_Rate column is dropped** as it is quite irrelevant.
    - ☐ **Referral and Last_Ad_Seen** (categorical columns) are converted to numerical by **one hot encoding**.
    - ☐ The first one hot encoded column is not dropped. **Dummy variable trap is overlooked** as we are working with tree-based algorithms (AdaBoost and XGBoost) and no matrix inversion operations are involved.

Model Selection and Explanation

The models selected are **AdaBoost and Xg Boost.**

AdaBoost :

- It is a stagewise additive model.
- It uses **Decision Stumps** as its base model.
    - Decision Stumps are weak learners (Decision Trees with max_depth=1)

Overall Working of AdaBoost Classifier (Big Idea) :

- ➢ This algorithm first trains a base classifier (a decision stump) and makes predictions on the training set.
- ➢ The algorithm then **increases the relative weight of misclassified training instances**.
- ➢ Then it trains a second classifier, using the updated weights, and makes predictions on the training set.
- ➢ Again increases the relative weight of misclassified training instances.
- ➢ and so on….
- ➢ After all the predictors are trained, the ensemble makes predictions by **adding the predictions of each predictor multiplied by the weight of the predictor**.
    - Each predictor has a weight depending on their overall accuracy on the weighted training set.

Formulas :

Model :

$$h(x) = \sum_{i=1}^{m} (\alpha_i * h_i(x))$$

If h(x) is +ve then Purchase = 1 else Purchase = 0

- m is the number of predictors.
- α is the weight of each predictor.

α (weight of each predictor) :

$$\alpha = 0.5 * \ln(\frac{1 - error}{error})$$

Error :

error = Σ weight of misclassified training instances.

Weight of each training instance initially = $\frac{1}{no.\,of\,rows}$

New weight :

For misclassified points	New_wt = current_wt * e^{α_m}
For correctly classified points	New_wt = current_wt * $e^{-\alpha_m}$

$\Sigma weights\ must\ be = 1\ so\ weights\ must\ be\ normalised.$

Normalised weight = $\frac{weight}{\Sigma weight}$

Assumptions of the model :

- The dataset does not have missing values.
- The dataset has less outliers as adaboost has the tendency of overfitting.

The model makes prediction using the formula :

$$\boxed{h(x) = \sum_{i=1}^{m} (\alpha_i * h_i(x))}$$

Where m is the number of predictors.

This model is used because :

- Our dataset does not have any missing value.
- The Browser_Refersh_Rate feature has been dropped as it was irrelevant . It was also a noisy feature. So, the chances of overfitting are less.
- Moreover AdaBoost is an efficient boosting algorithm, which works great on small datasets like our customer_behaviour dataset.

For this dataset the best hyperparameters of AdaBoost are :
- n_estimators = 10
- learning _rate = 0.5

Accuracy = 79.56%
Precision score 0.72
Recall score 0.54
F1 score 0.62
AUC = 0.86
Confusion matrix :

	0	**1**
0	285 (True -ve)	29 (False +ve)
1	63 (False -ve)	73 (True +ve)

The AdaBoost model from scratch gives an accuracy of 76.89% .

XG Boost :

Formulas :

Prediction :

$$\boxed{\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)}$$

We add such f_k after each iteration so that objective function is minimised.

where f_k is a regression tree model.

Objective function :

$$\boxed{\mathcal{L}^{(t)} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)}$$

L is the loss function.

- MSE (mean squared error) for regression problems
- Log_loss for classification problems

$\hat{y}_i^{(t-1)}$ is the prediction of the i^{th} instance after $(t - 1)^{th}$ iteration.

$\Omega(f_t)$ is a built in regularization parameter

$$\boxed{\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2}$$

- T is the number of leaf nodes.
- w is the weight/output of leaf nodes.
- γ and λ are regularization parameters.

f_t is the last added DT after t^{th} iteration.

We can approximate the loss function part of objective function, to make it differentiable (so that we can carry out optimizations), using Taylor series about the point a = $\hat{y}_i^{(t-1)}$.

Objective function after approximation :

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n}\left[L(y_i,\ \hat{y}_i^{(t-1)}) + g_i f_t(x_i)) + \frac{1}{2} h_i f_i^{2}(x_i))\right] + \Omega(f_t(x_i))$$

$g_i = \dfrac{\partial L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$ (gradient)	$h_i = \dfrac{\partial^2 L(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)^2}}$ (hessian)

$L(y_i,\ \hat{y}_i^{(t-1)})$ is a constant , so removing it and expanding $\Omega(f_i(x_i))$: .

Final simplified Objective function :

$$\Rightarrow \quad \mathcal{L}^{(t)} = \sum_{j=1}^{T}\left[\sum_{i \in I_j} g_i w_j + \frac{1}{2}\left(\sum_{i \in I_j} h_i + \lambda\right) w_j^2\right] + \gamma T$$

For **objective function to be minimum** $\frac{\partial L}{\partial w_j}$ **= 0** . This gives

$$w_j = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Substituting this value in final objective function we get :

$$\mathcal{L}^{(t)} = \sum_{j=1}^{T}\left[-\frac{1}{2}\frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda}\right] + \gamma T$$

Also called similarity score

Loss reduction after a split is given by :

Similarity score $_{\text{left child node}}$ + Similarity score $_{\text{right child node}}$ - Similarity score $_{\text{root node}}$

Assumptions of the model :

XGBoost is a scalable tree boosting algorithm.

- The loss function must be differentiable.
- There must be less outliers (to avoid overfitting).
- The dataset must not be very small.

The model makes prediction using the formula :

$$\widehat{y_i} = \sum_{k=1}^{K} f_k(x_i)$$

This model is used because :

- XGBoost handles class imbalance (our model is slightly imbalanced).
- It provides high accuracy.
- It supports parallel processing and uses cache memory efficiently.

For this dataset the best hyperparameters of XGBoost are :

- n_estimators = 20
- learning _rate = 0.1
- colsample_bytree = 0.8
- max_depth = 3
- subsample = 0.8
- scale_pos_weight = 1

Accuracy = 79.11%
Precision score 0.72
Recall score 0.51
F1 score 0.59
AUC = 0.84
Confusion matrix :

	0	**1**
0	287 (True -ve)	27 (False +ve)
1	67 (False -ve)	69 (True +ve)

The XGBoost model from scratch gives an accuracy of 78.67% .

Comparative analysis

<table>
<tr><th></th><th>Adaboost</th><th>XGBoost</th></tr>
<tr><td>Accuracy score</td><td>79.56%</td><td>79.11%</td></tr>
<tr><td>Precision score</td><td>0.72</td><td>0.72</td></tr>
<tr><td>Recall score</td><td>0.54</td><td>0.51</td></tr>
<tr><td>F1 score</td><td>0.62</td><td>0.59</td></tr>
<tr><td>AUC</td><td>0.86</td><td>0.84</td></tr>
<tr><td>Confusion matrix</td><td><table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>285</td><td>29</td></tr><tr><td>1</td><td>63</td><td>73</td></tr></table></td><td><table><tr><td></td><td>0</td><td>1</td></tr><tr><td>0</td><td>287</td><td>27</td></tr><tr><td>1</td><td>67</td><td>69</td></tr></table></td></tr>
</table>

We can see that both the models perform in a similar manner, with **Adaboost being slightly better than XGBoost**.

- Both the models have **similar overall accuracy and precision**.
- AdaBoost outperforms XGBoost on Recall score , F1 score, and ROC-AUC, indicating better ability to capture true positives and more effective class separation (also visible from confusion matrix) .
- Higher AUC in AdaBoost indicates better discrimination between classes.

We are detecting customer purchase so **false positives must be avoided** as marketing to uninterested users is costly. **High precision models** are required. **Both the models have precision > 0.7** .

- XGBoost may not have performed well due to **class imbalance**.
- The slightly better performance of AdaBoost could be attributed to the relatively **small size of the dataset**.

- Another reason can be **less overfitting in AdaBoost** as it uses Decision stumps(shallow decision trees) whereas XGBoost uses full Decision trees
- **XGBoost** has a lot of hyperparameters so **tuning is more difficult** than tuning in AdaBoost.