

Ecommerce Prediction Task



Author:

Mohd Hammad Ansari

Second Year Undergrad Student

IIT Kharagpur

Abstract:

This project, Click to Cart: A Predictive Modeling Task, focuses on predicting user purchasing behavior in e-commerce platforms using session-level interaction data. The main goal is to determine whether a user will make a purchase during a particular visit, enabling businesses to enhance personalization, optimize digital marketing strategies, and improve revenue outcomes.

To simulate real-world conditions, the dataset provided is rich in behavioral signals yet includes noise and irrelevant features, mimicking the complexity encountered in actual e-commerce data. Key features include time spent on site, pages viewed, ad interactions, and device types.

This task emphasizes practical applications such as tailoring real-time recommendations, efficiently allocating advertising budgets, and minimizing session abandonment. By developing robust machine learning models that extract meaningful patterns from user behavior, the project aims to contribute to more intelligent, data-driven decision-making in digital commerce.

Table of contents:

- Introduction to Problem Statement
- Exploratory Data Analysis (EDA)
- Model Selection and Explanation
- Model Training and Evaluation
- Hyperparameter Tuning
- Comparative Analysis

Introduction to the Problem Statement:

This dataset simulates anonymized e-commerce customer session logs and is designed for binary classification: predicting whether a customer will make a purchase during their session.

Features:

- **Time_on_site**: The time a customer spent on the website during the session.
- **Pages_viewed** : number of pages visited by the user during the session.
- **Clicked_ad** :Indicates whether the user clicked on a product advertisement.
- **Cart_value**: The total value of products in the user's cart during the session (in USD).
- **Referral** (*categorical*): The source from which the user was referred to the website. Values: "Google", "Facebook", "Instagram", "Direct"
- **Browser_Refresh_Rate** (*float*): A synthetic, potentially distracting technical metric representing the frequency of browser refreshes.
- **Last_Ad_Seen** (*categorical*): The name of the last ad campaign the user was exposed to. Values: "A", "B", "C", "D".

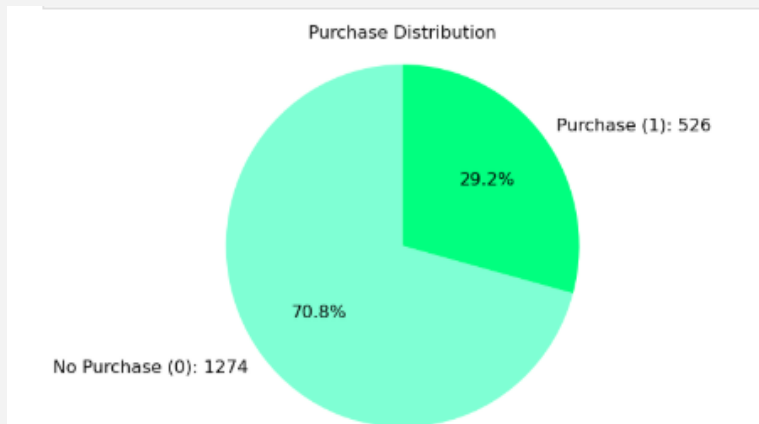
Key Challenges:

1. **Noise**: The dataset contains irrelevant or redundant columns (e.g., duplicate metrics, system logging artifacts) intended to test the effectiveness of feature selection techniques.
2. **Temporal Dynamics**: User behavior during sessions may vary based on factors such as time of day, ongoing promotional campaigns, or seasonal trends, introducing temporal variability into the data.
3. **Class Imbalance**: Purchase events are relatively rare compared to non-purchase sessions, leading to a skewed class distribution that requires specialized handling (e.g., resampling, appropriate evaluation metrics).

Objective:

Developing a robust classification model to predict whether a user session indicates purchase intent (label 1) or not (label 0). Utilize the training data for feature engineering, model selection, and performance validation to ensure accurate and reliable predictions.

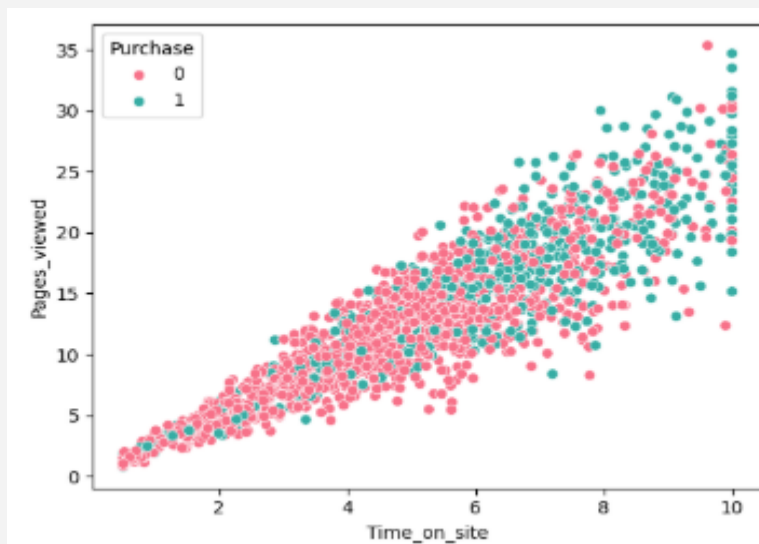
Exploratory Data Analysis:



Analyzing the Purchase distribution in the training set of size 1800.

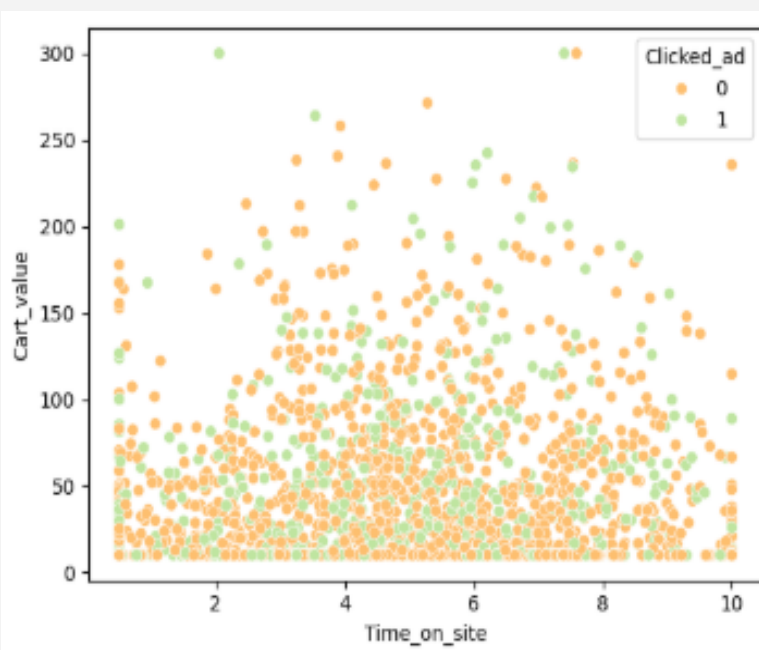
526 users have purchased comprising of 29.2%

1274 users have not purchased comprising of 70.8%



This plot gives us a significant insight regarding the importance of time spent by the user on e commerce website in determining the purchase

It is observed that in general those who spent more time are having a greater probability of purchasing



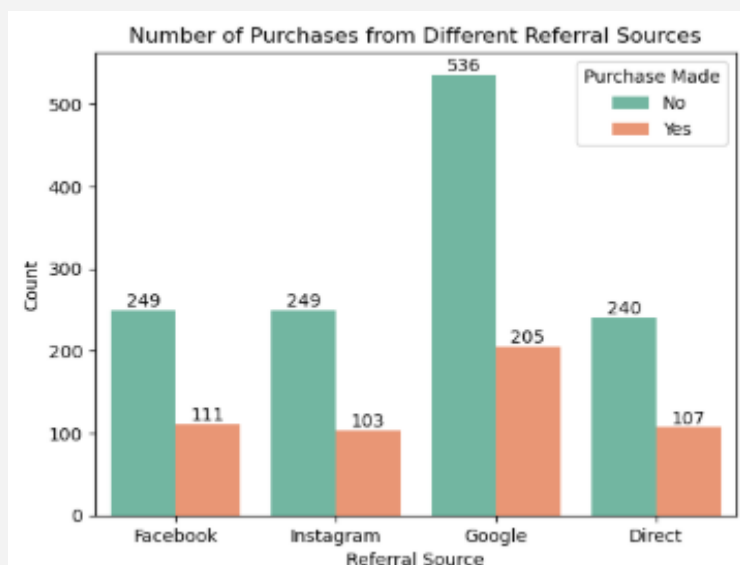
This plot and the next plot is very crucial for a company to determine how much cost it should spend on advertisement

It is observed that those who clicked the advertisement had relatively less cart values and those users have also spent a bit lesser time (overall) on website



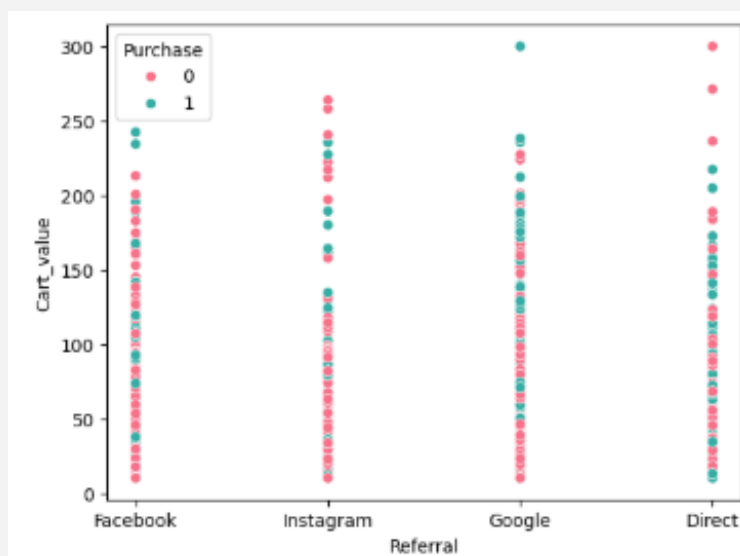
Insights: Potential Uplifts from ad campaigns

It can clearly be observed that those who Clicked Ad and made a purchase have spent relatively less time on the site whereas those who didn't come up on website due to ads and have less time on site are not making a purchase



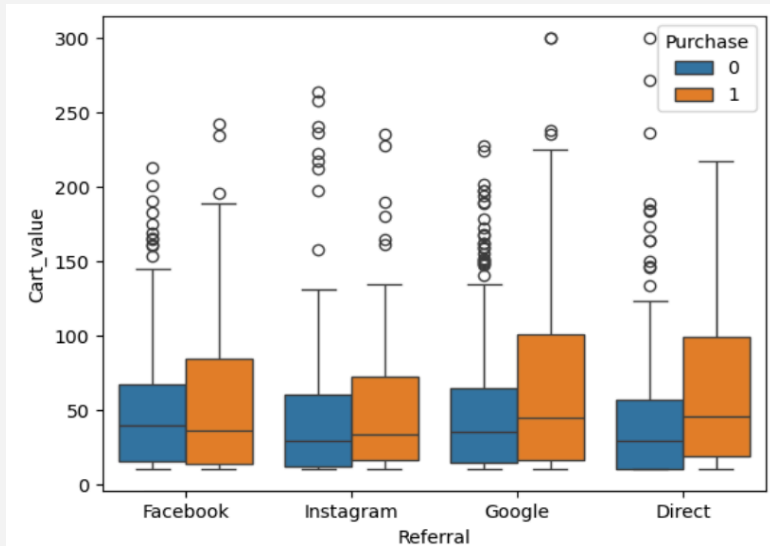
Advertising on which Social Media Platform is beneficial for the company

It is observed that Google is responsible for getting more customers on the company's website. While facebook has a greater ratio of purchases being made/total count.



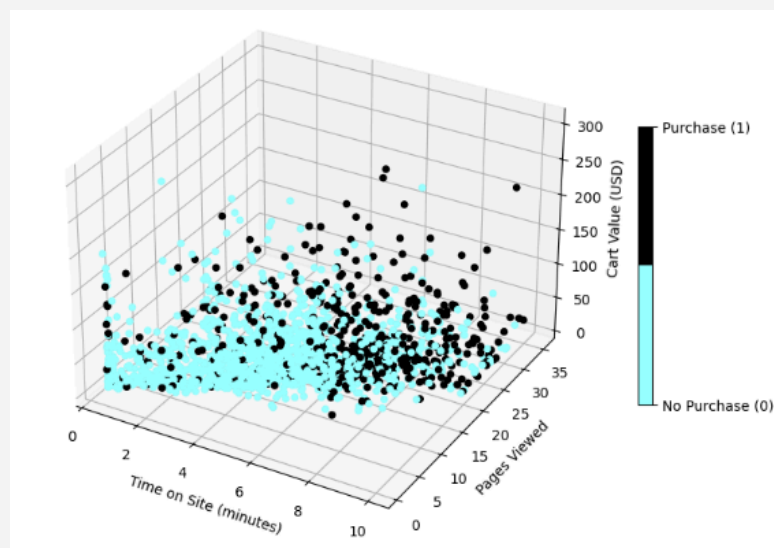
Insights: Cart value with respect to the referral.

Customers who visit website after watching ads on Google have purchased stuff at a greater cart value than other platforms, such customers are more in number too



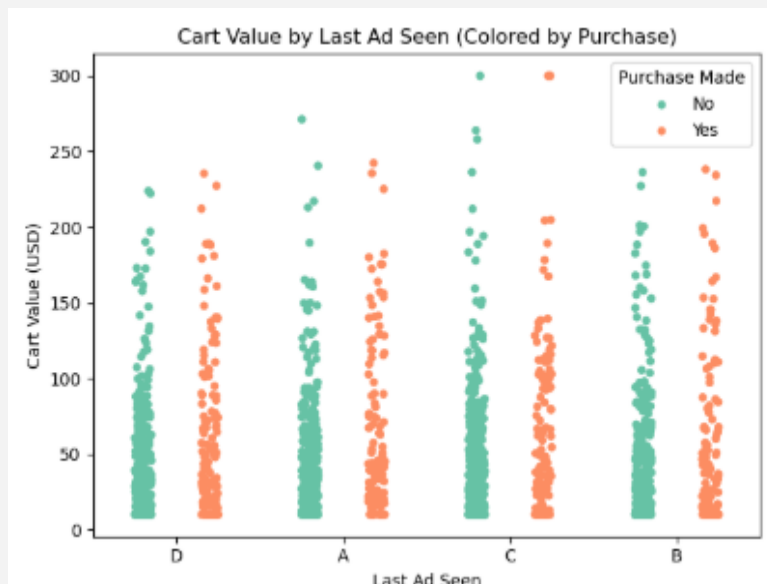
Outliers in the Cart Values:

The most amount of outliers in Cart Values are provided by Google only



3D Scatter Plot with features Pages Viewed, Time on Site, Cart Value, Purchase:

It can be clearly seen that customers who spend more time either by viewing pages or spending time on site have a greater chance of making a purchase

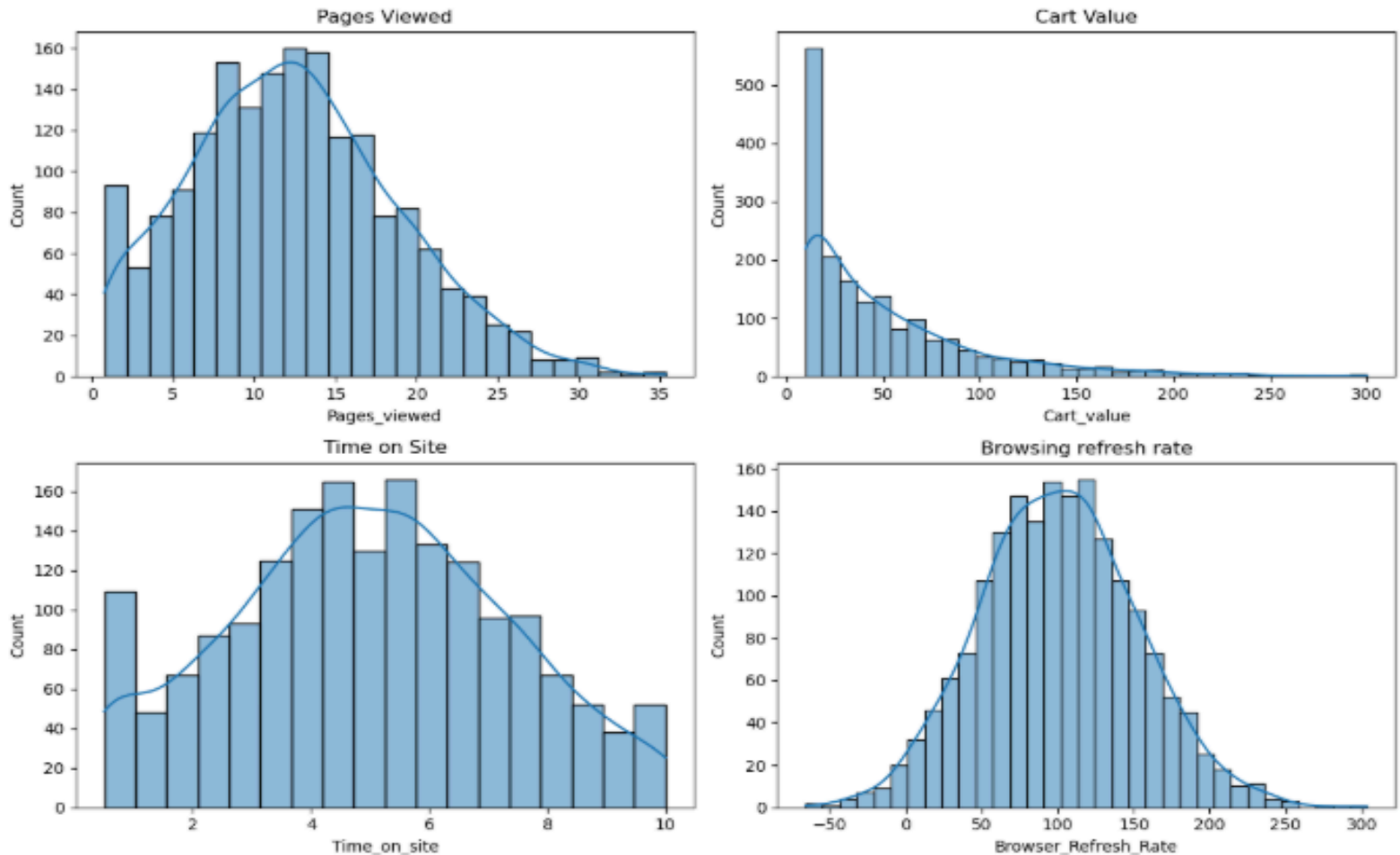


Which Specific Ad Campaign drives a higher cart value with a surety of Purchase:

It is observed that Ad 'B' is getting more customers who purchase stuff with a greater Cart Value.

Analyzing Feature Distribution and

Detecting Outliers:



After Analyzing we realise there are more outliers present in Cart Values feature column, whereas the Time on site, Browsing Refreshing Rate, Pages Viewed are comparatively having less outliers

Skew Scores:

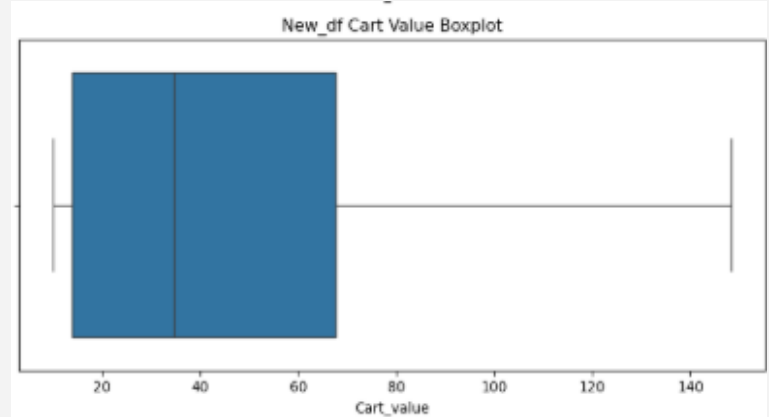
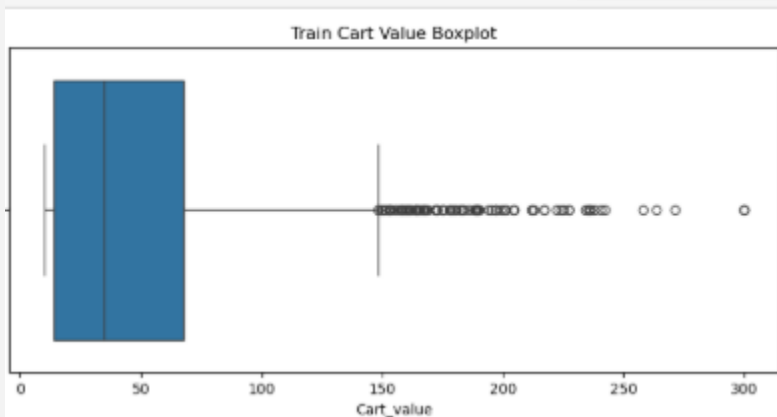
1. Cart_value: 1.78094
2. Pages_viewed: 0.3756775
3. Browser_Refresh_Rate: 0.104536
4. Time_on_site: 0.0313

Removal of Outliers from Cart Values using Capping:

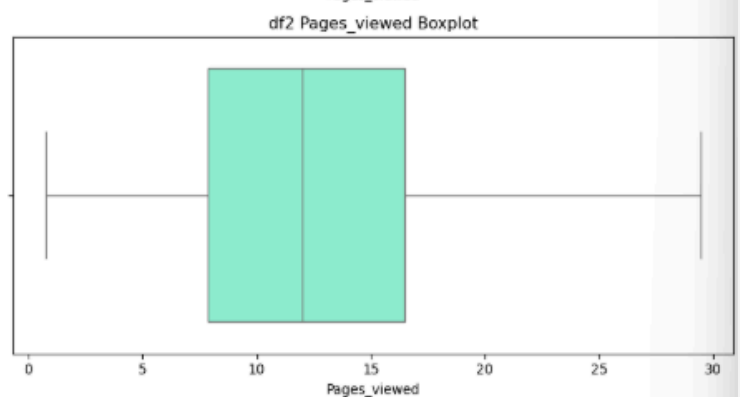
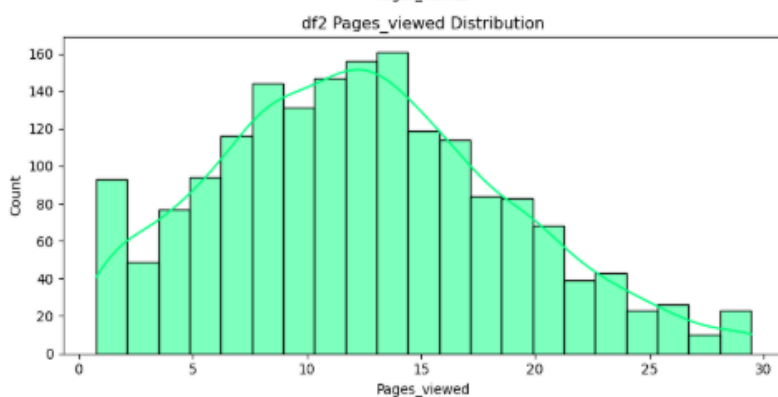
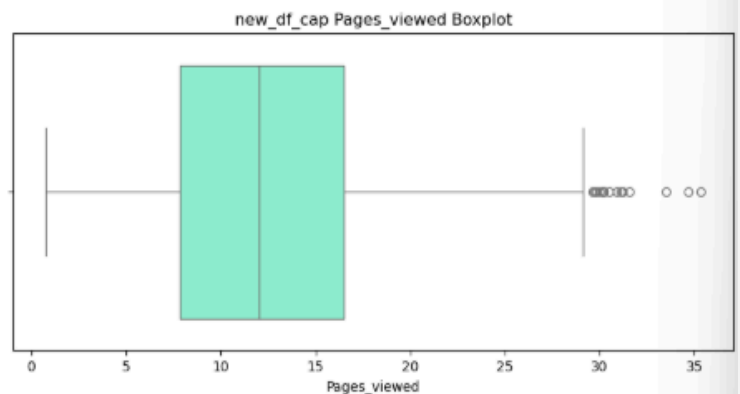
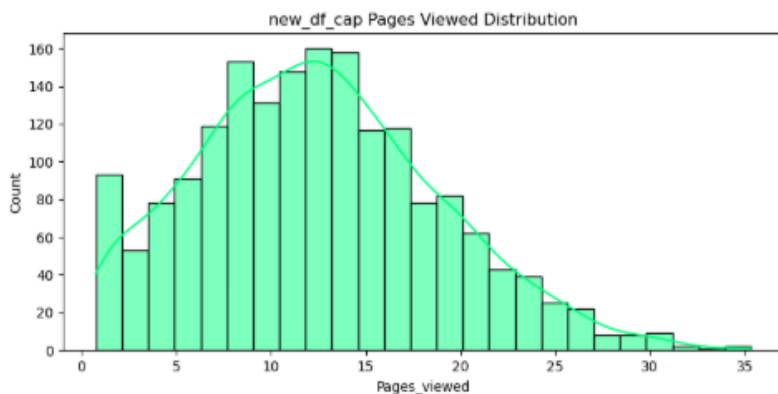
```
new_df_cap = train.copy()
new_df_cap['Cart_value'] = np.where(
new_df_cap['Cart_value'] > upper_limit,
upper_limit,
np.where(
new_df_cap['Cart_value'] < lower_limit,
lower_limit,
new_df_cap['Cart_value']
)
```

Removal of Outliers using Capping method:

- Evaluated Upper Limit: Percentile 75 + 1.5*IQR
- IQR= Percentile 75-Percentile 25
- Evaluated Lower Limit: Percentile 25 - 1.5*IQR



Removal of outliers from Pages_viewed using Capping:



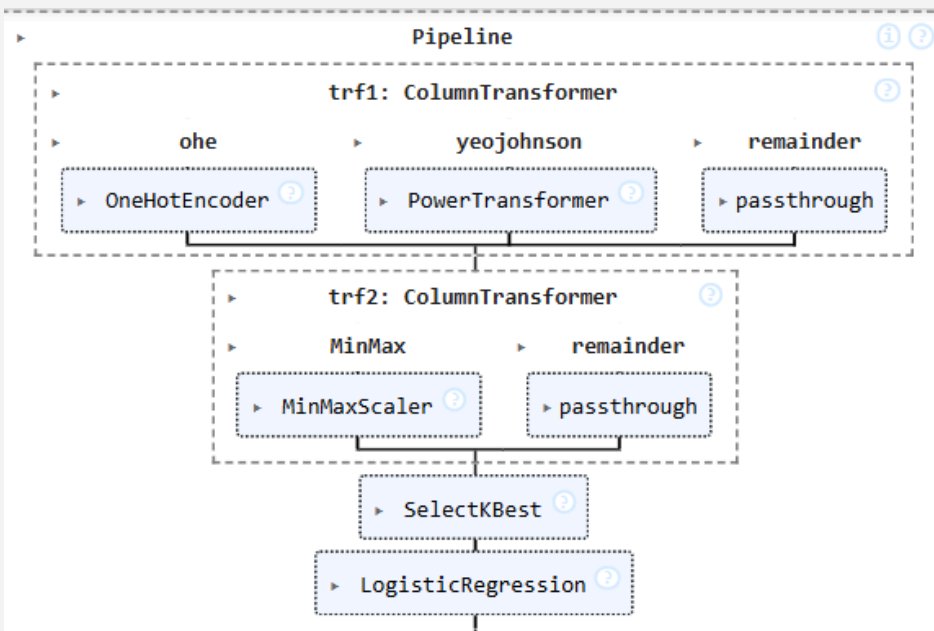
Similar Procedure was followed for Browser_Refresh_Rate and Zscore normalization was done on the Time_on_Site feature column.

Model Training and Evaluation:

Implementing Logistic Regression via Machine Learning Pipeline

- Mathematical aspects of Logistic Regression are deeply covered in the Jupyter Notebook (ECommercePrediction.ipynb)
- In Transformer 1:
 - ❖ Column Transformer
 1. One Hot Encoding was performed on the columns 'Referral', 'Last_Ad_Seen', with drop='First'
 2. Yeo-Johnson transformation was implemented on the numeric features in order to Stabilize variance and Make distributions more Gaussian (normal)
- In Transformer 2:
 - ❖ Min Max Scaling was performed on the numerical featured columns to get values scaled between 0 and 1 as they are crucial for:
 1. Algorithms like k-nearest neighbors (KNN), SVM, and neural networks which are sensitive to the scale of input features
 2. Improving Convergence Speed
- In Transformer 3:
 - ❖ Anova F-Test: In order to select the K-Best Features(Mathematical derivation and its evaluation criteria explained in notebook).
 1. High F-Value: Important Features
 2. Low F-Values: Less Important features
- In Transformer 4:
 - ❖ Logistic Regression was implemented with:
 1. L2 Penalty that is evaluated using L2 normalization
 2. C=1.0(responsible for determining Regularization)
 3. Iterations :1000

Mathematical Derivation and Intuitions for Confusion Matrix, Recall, F1 Score, Precision is explained in the Jupyter Notebook.



Accuracy achieved:

0.771111

Confusion Matrix :

```
array([[280, 34],
       [ 69, 67]], dtype=int64)
```

F1 Score: 0.565

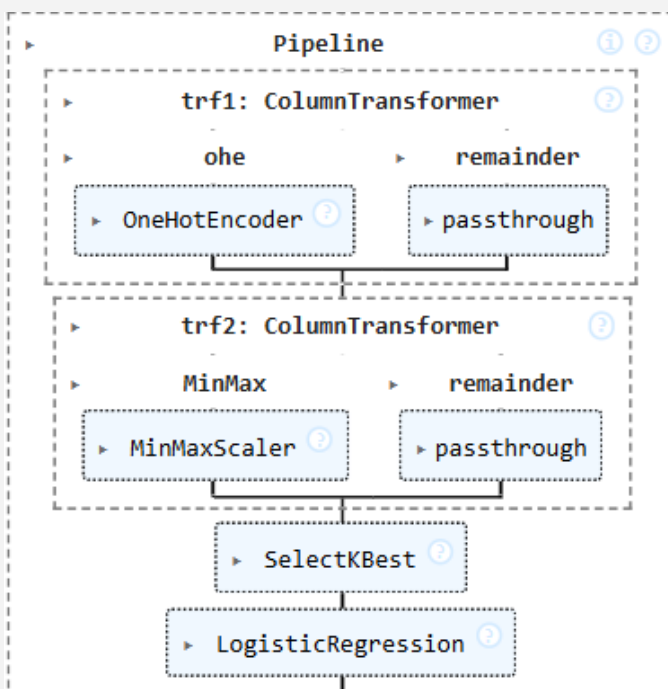
Precision: 0.663

Recall: 0.492

Implementing **Logistic Regression** again via **Machine Learning Pipeline (Feature Engineering Included)**

An Interacting Feature (A single column '**Time_Spent**' consisting of **Time_on_site*Page_Viewed** was replaced with these 2 separate feature columns).

Nearly similar procedure was followed but Yeo Johnson Transformation was not included in the pipelines and thus not applied this time.



Instead of using **Anova F-test** classifier in SelectKBest we have used **chi2** classifier this time with **k=5**.

Accuracy achieved: 0.7822222

F1 Score: 0.58823

Precision: 0.686274

Recall: 0.514

Confusion Matrix :

```
array([[282, 32],
       [ 66, 70]], dtype=int64)
```

Implementing GridSearchCV on **Logistic Regression** with Optimization Algorithm: Stochastic Average Gradient Accelerated (SAGA) on a Machine Learning Pipeline

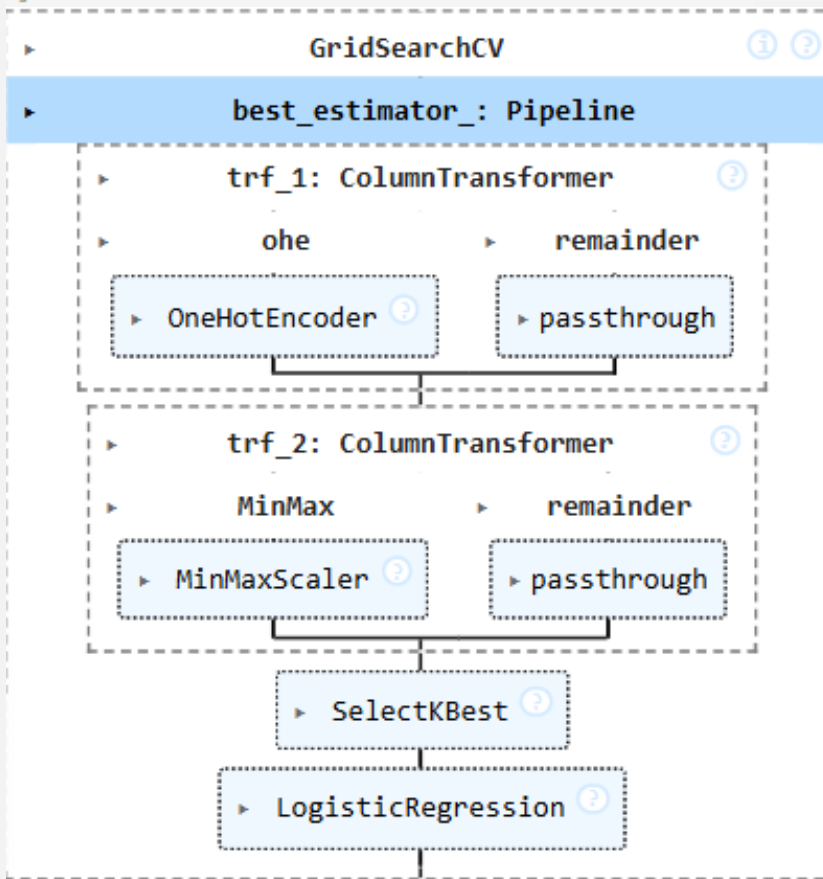
Mathematical aspects of Regularization terms of Logistic Regression(L1,L2,Elastic Net) are deeply covered in the Jupyter Notebook (ECommercePrediction.ipynb)

SAGA:

- ❖ Supports All Regularization Penalties: L1,L2,Elastic Net
- ❖ Works with Large Datasets: It processes **mini-batches** of the data and hence is faster.

```
param_grid = [  
    {  
        'trf_4_penalty': ['l1', 'l2'],  
        'trf_4_C': [0.001, 0.01, 0.1, 1, 10, 100]  
    },  
    {  
        'trf_4_penalty': ['elasticnet'],  
        'trf_4_C': [0.001, 0.01, 0.1, 1, 10, 100],  
        'trf_4_l1_ratio': [0.3, 0.5, 0.7]  
    }  
]
```

```
grid_search = GridSearchCV(  
    estimator=pipe3,  
    param_grid=param_grid,  
    scoring='accuracy',  
    cv=5,  
    n_jobs=-1  
)  
grid_search.fit(Xtrain, ytrain)
```



Best Parameters:

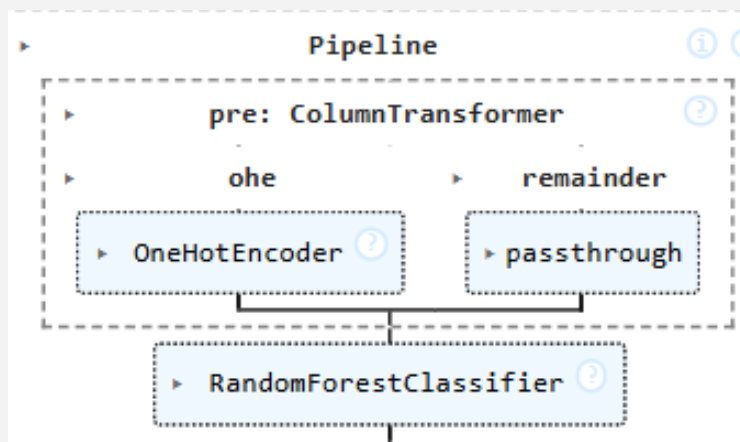
C=1, l1_ratio=0.5,
Penalty=ElasticNet

Accuracy achieved:
0.782222

Implementing Random Forest Classifier via Machine Learning Pipeline

Mathematics regarding Gini Impurity, entropy, Decision Trees, Information Gain are briefly explained in the Jupyter Notebook.

- ★ Implemented One hot encoding on the categorical column of dataset with drop='First' to avoid multicollinearity
- ★ At last implemented Random Forest Classifier with :
 - n_estimators=100
 - min_samples_split=5
 - n_jobs=-1



Accuracy achieved: 0.77111

F1 Score: 0.61

Precision: 0.63

Recall: 0.59

Confusion Matrix :

```
[[267  47]
 [ 56  80]]
```

🌲 Do Random Forest Classifiers Need Yeo-Johnson or Scaling?

- They are **invariant** to feature scaling:
 - They **split data by feature thresholds**, not by distance or gradient.
- Don't assume **normality** or **linear relationships**.
 - They can naturally handle **non-normal**, **skewed**, and **nonlinear** data distributions.
- They are robust to **different feature scales and units**.

Implementing GridSearchCV on Random Forest Classifier

```
param_grid = {
    'classifier_n_estimators': [100, 200, 300, 400, 500],
    'classifier_max_depth': [None, 5, 10, 15, 20],
    'classifier_max_features': ['sqrt', 'log2', None],
    'classifier_min_samples_split': [2, 5, 10],
    'classifier_min_samples_leaf': [1, 2, 4],
    'classifier_bootstrap': [True, False]
}
```

```
grid_search = GridSearchCV(
    estimator=pipe,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=2
)
```

Results :

- **Best Parameters:**
 - ❖ Bootstrap: True
 - ❖ Max Depth: None
 - ❖ Max Features: None
 - ❖ Min Sample Split: 5
 - ❖ Number of Estimators: 300
 - ❖ Min Sample Leaf: 2
- **Accuracy achieved: 0.78**

Implementing Ensemble Learning with Voting Classifier Using Bagged Base Models and Random Forest

- ❖ Mathematical Explanation for this technique is explained in Jupyter Notebook
- ❖ This Ensemble Model Comprises of 5 separate classification models :

```
ROW_SAMPLE_RATIO = 0.8
FEATURE_SAMPLE_RATIO = 0.6

bagged_svm = BaggingClassifier(
    estimator=SVC(kernel='rbf', probability=True, class_weight='balanced'),
    n_estimators=50,
    max_samples=ROW_SAMPLE_RATIO,
    max_features=FEATURE_SAMPLE_RATIO,
    bootstrap=True,
    bootstrap_features=True,
    random_state=42
)
```

```
bagged_knn = BaggingClassifier(
    estimator=KNeighborsClassifier(n_neighbors=5),
    n_estimators=50,
    max_samples=ROW_SAMPLE_RATIO,
    max_features=FEATURE_SAMPLE_RATIO,
    bootstrap=True,
    bootstrap_features=True,
    random_state=42
)
```

```
bagged_lr = BaggingClassifier(
    estimator=Pipeline([
        ('scaler', StandardScaler()),
        ('lr', LogisticRegression(class_weight='balanced'))
    ]),
    n_estimators=50,
    max_samples=ROW_SAMPLE_RATIO,
    max_features=FEATURE_SAMPLE_RATIO,
    bootstrap=True,
    bootstrap_features=True,
    random_state=42
)
```

```
random_forest = RandomForestClassifier(
    n_estimators=50,
    max_depth=4,
    class_weight='balanced',
    max_features=FEATURE_SAMPLE_RATIO,
    random_state=42
)
```

```

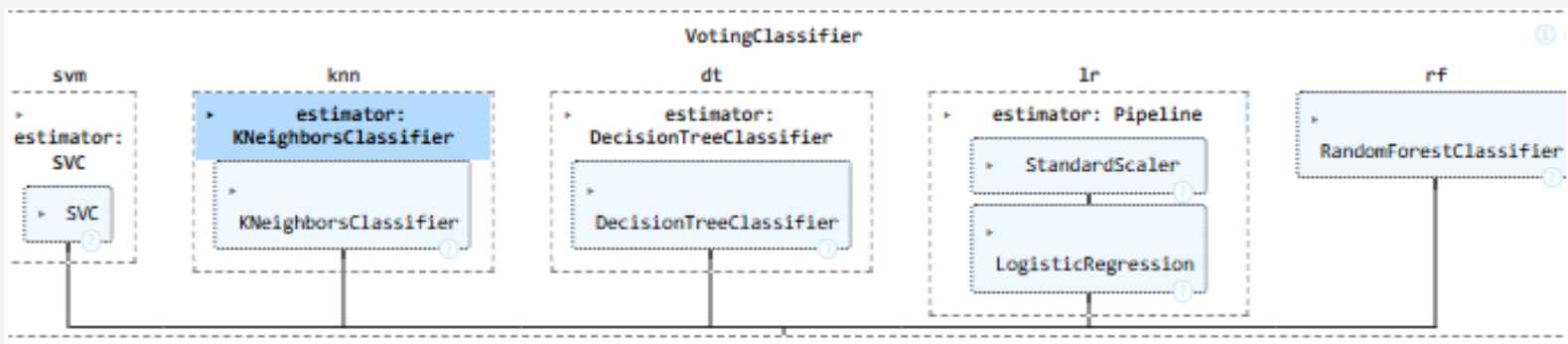
bagged_dt = BaggingClassifier(
    estimator=DecisionTreeClassifier(max_depth=4, class_weight='balanced'),
    n_estimators=50,
    max_samples=ROW_SAMPLE_RATIO,
    max_features=FEATURE_SAMPLE_RATIO,
    bootstrap=True,
    bootstrap_features=True,
    random_state=42
)

```

```

ensemble = VotingClassifier(
    estimators=[
        ('svm', bagged_svm),
        ('knn', bagged_knn),
        ('dt', bagged_dt),
        ('lr', bagged_lr),
        ('rf', random_forest)
    ],
    voting='soft',
    n_jobs=-1
)

```



Results :

Accuracy achieved: 0.80

F1 Score: 0.67

Precision: 0.67

Recall: 0.68

Confusion Matrix :

```

[[268  46]
 [ 44  92]]

```

Implementing ADABOOST ALGORITHM CLASSIFIER

- ❖ Mathematical aspects of ADABOOST and SAMME.R algorithm are deeply covered in the Jupyter Notebook (ECommercePrediction.ipynb)
- ❖ Why using SAMME.R:
 1. **More nuanced decisions:** Learners contribute more when they're more confident.
 2. **Smoother margins:** Reduces sharp transitions at decision boundaries.

```
from sklearn.ensemble import AdaBoostClassifier
base_estimator = DecisionTreeClassifier(max_depth=1, random_state=42)

adaboost = AdaBoostClassifier(
    estimator=base_estimator,
    n_estimators=5,
    learning_rate=1.0,
    random_state=42,
    algorithm='SAMME.R'
)

adaboost.fit(df_encoded, Y2)

YPRED = adaboost.predict(df_encoded2)

accuracy = accuracy_score(yt2, YPRED)
print(f"Precision: {precision_score(yt2, YPRED):.4f}")
print(f"Recall: {recall_score(yt2, YPRED):.4f}")
print(f"F1 Score: {f1_score(yt2, YPRED):.4f}")

print("\nConfusion Matrix:")
print(confusion_matrix(yt2, YPRED))
print(f"AdaBoost Classifier Accuracy: {accuracy:.2f}")
```

Accuracy: 0.80

F1 Score: 0.6245

Precision: 0.7327

Recall: 0.54

Confusion Matrix :

```
[[ 287   27]
 [   62   74]]
```

Implementing GridSearchCV on ADABOOST Classifier

```
param_grid = {
    'n_estimators': [50, 100, 200, 10, 150],
    'estimator__criterion': ['gini', 'entropy'],
}
```

Accuracy: 0.7933

F1 Score: 0.6543

Precision: 0.6617

Recall: 0.6471

```
grid_search = GridSearchCV(
    estimator=ada,
    param_grid=param_grid,
    cv=2,
    n_jobs=-1,
    verbose=2
)
```






Confusion Matrix :

```
[[ 269   45]
 [   48   88]]
```

Best Parameters :

Gini, n_estimators =10

Implementing Gradient Boosting Algorithm

- ❖ Mathematical aspects of **Gradient Boosting** algorithm are deeply covered in the Jupyter Notebook (ECommercePrediction.ipynb)
- ❖  Why Boosting Doesn't Require Normalization or Normal Distribution
 - ★  Tree-Based Weak Learners Are Invariant to Feature Scaling:
 - Most boosting algorithms use **decision trees** as their weak learners. Trees: Split data based on feature thresholds (e.g., "*Is feature 1 > 2.5?*").
 - **Do not use distances, dot products, or gradients of features.**
 -  So: **Normalization or standardization isn't needed.**
 - ★ No Assumption of Feature Distributions
 - Learns from **additive models of weak classifiers**, so it adapts to any structure in the data—linear or nonlinear, skewed or symmetric.
 -  So: **Skewed or non-normal data is not a problem.**
 - ★  Boosting Works by **Reweighting** Mistakes, Boosting algorithms iteratively by:
 - Focusing on examples the current model gets wrong.
 - Adding models to **correct those mistakes**.
 - Reweighting samples rather than manipulating feature space directly.

```
gb_model = GradientBoostingClassifier(  
    learning_rate=0.1,  
    n_estimators=100,  
    max_depth=3,  
    subsample=1.0,  
    random_state=42,  
)
```

Accuracy: 0.79

F1 Score: 0.64

Precision: 0.66

Recall: 0.62

```
gb_model.fit(df_encoded3, Y2)
```

Confusion Matrix :

```
[[271  43]  
 [ 52  84]]
```

```
YPRE_GB = gb_model.predict(df_encoded2)
```

Implementing GridSearchCV on Gradient Boosting Classifier

```
param_grid = {  
    'n_estimators': [50, 100, 200, 150],  
    'learning_rate': [0.01, 0.1, 0.5, 1.0],  
    'max_depth': [3, 5, 7],  
    'subsample': [0.8, 1.0],  
    'max_features': ['sqrt', 'log2', None]  
}
```

```
grid_search = GridSearchCV(  
    estimator=gbm,  
    param_grid=param_grid,  
    cv=5,  
    n_jobs=-1,  
    verbose=2  
)
```

Results :

- **Best Parameters:**
 - ❖ Learning rate : 0.1
 - ❖ Max Depth: 3
 - ❖ Max_features: sqrt
 - ❖ Number of Estimators: 50
 - ❖ Sub Sample: 1.0
- **Accuracy achieved: 0.7956**

Precision: 0.6930

Recall: 0.5809

F1 Score: 0.6320

Confusion Matrix:

[[279 35]

[57 79]]

Implementing Support Vector Machines(SVMs)

```
svm_model = SVC(  
    kernel='rbf',  
    C=1.0,  
    probability=True,  
    random_state=42  
)
```

```
svm_model.fit(A1, A2)
```

```
YPRE_SVM = svm_model.predict(B1)
```

Accuracy: 0.77

F1 Score: 0.55

Precision: 0.67





Recall: 0.47

Confusion Matrix :

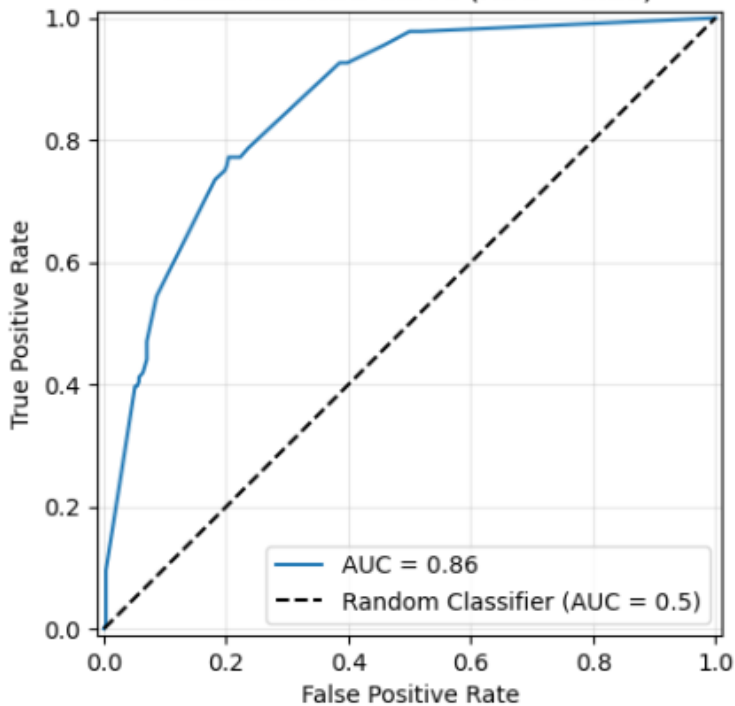
[[282 32]

[72 64]]

ROC CURVES FOR THE VARIOUS MODELS

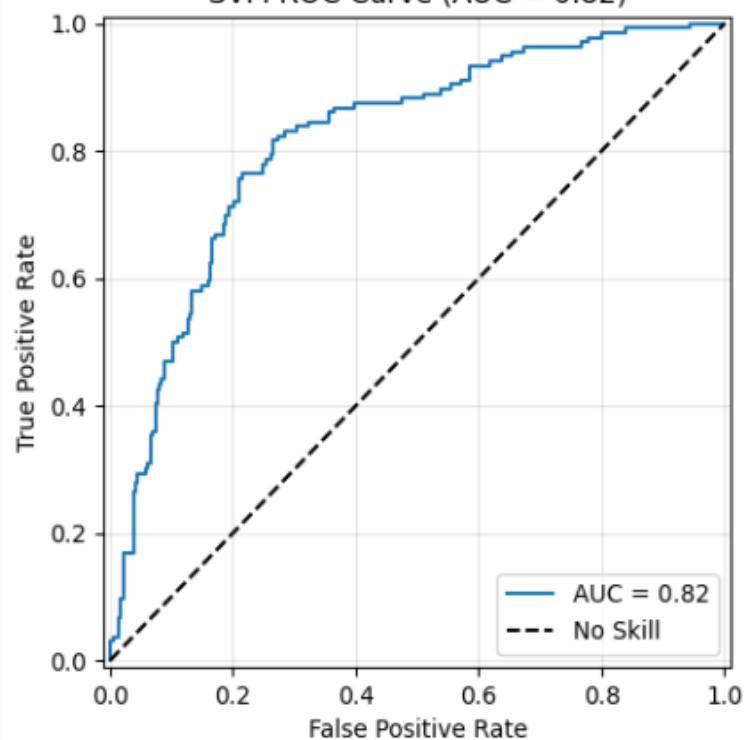
- ❖ A ROC curve is a graphical plot used to evaluate the performance of a binary classification model by showing the trade-off between:
 - ❖  True Positive Rate (TPR) (Sensitivity, Recall)
 - ❖  False Positive Rate (FPR)
- ❖  What the ROC Curve Shows:
 - ★  Illustrates how effectively your model distinguishes between the two classes.
 - ★ Useful for comparing the performance of different models.
- ❖ Characteristics of a Good ROC Curve:
 - The curve closely follows the top-left corner of the plot.
 - A higher Area Under the Curve (AUC) indicates better model performance in class separation.
- ❖ **Intuition Behind ROC Curve Evaluation:**
 - ➔ Think of the False Positive Rate (FPR) as a cost and the True Positive Rate (TPR) as a benefit. A better model produces a curve that offers high benefit (TPR) with low cost (FPR)—hence, a curve that rises steeply and stays near the top-left is more desirable.

AdaBoost ROC Curve (AUC = 0.86)

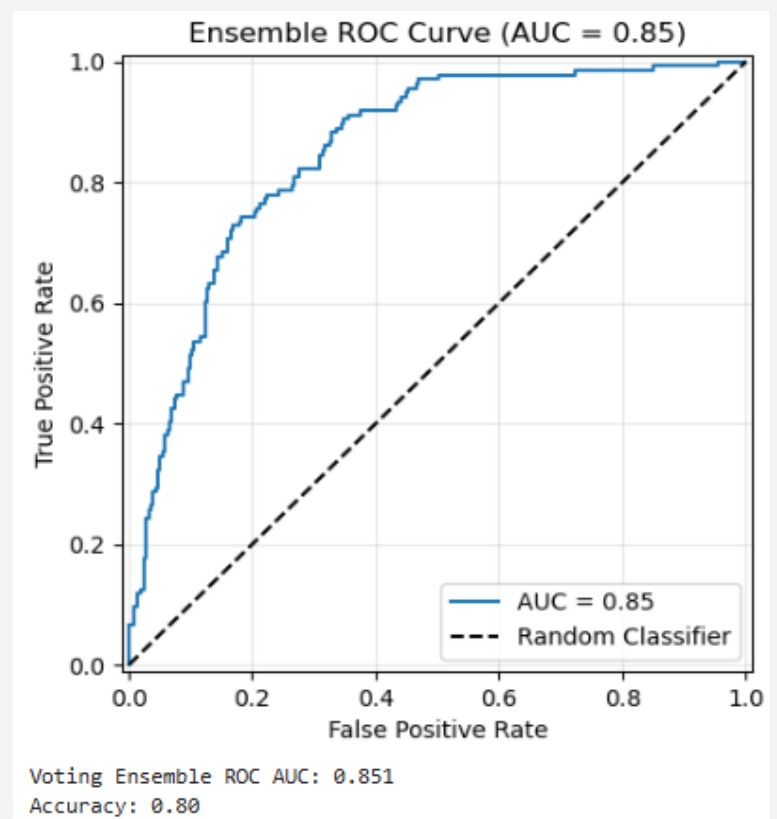
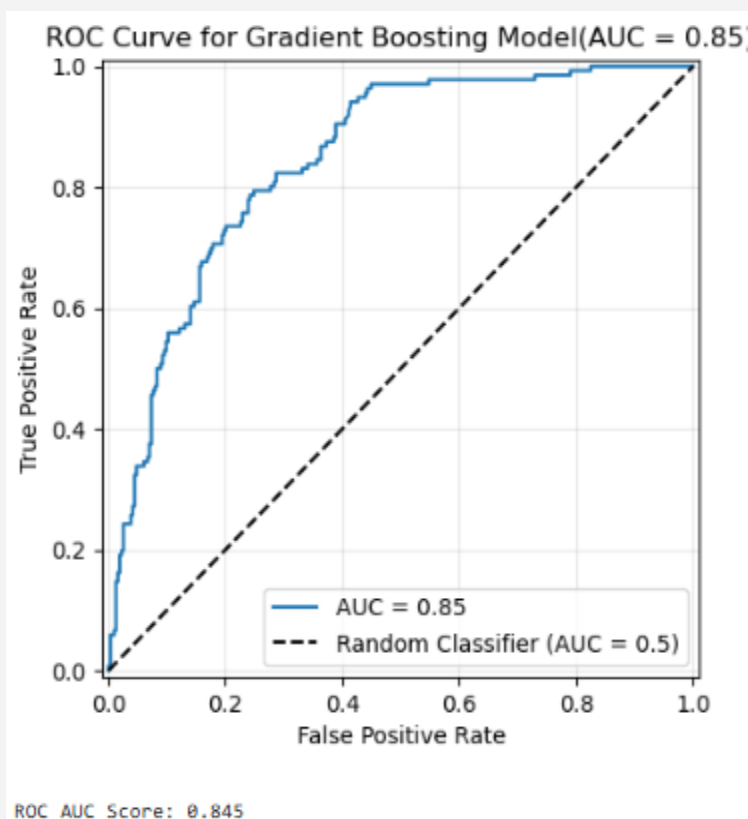
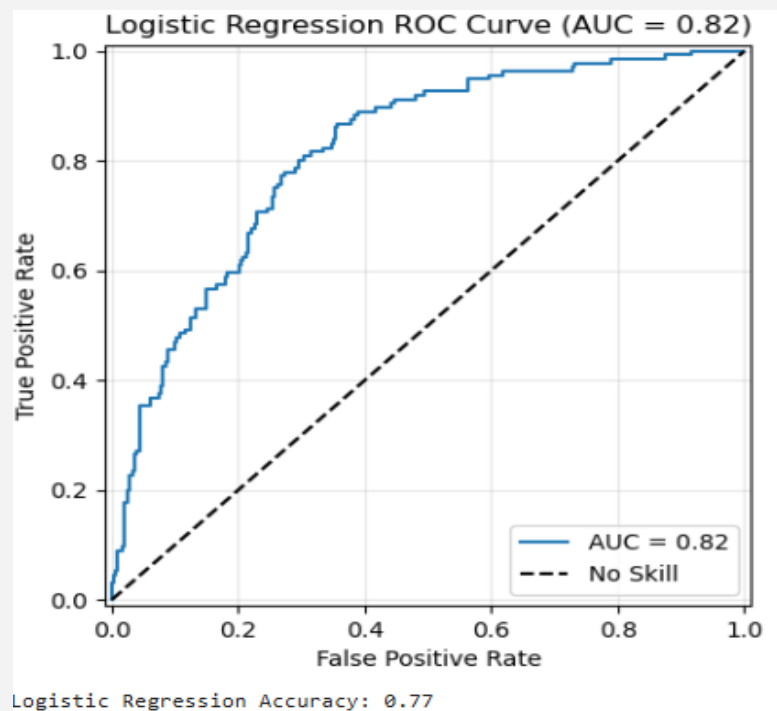
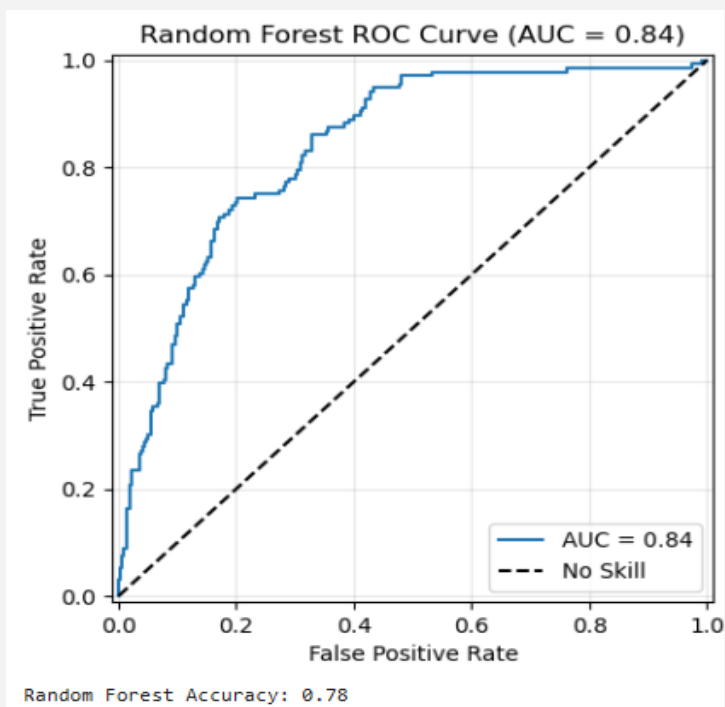


AdaBoost Classifier Accuracy: 0.80
ROC AUC Score: 0.862

SVM ROC Curve (AUC = 0.82)



SVM Accuracy: 0.77



Comparative Analysis of E-Commerce Purchase-Intent Models

Before beginning the analysis it should be kept in mind that:

"Datasets capture customer behavior [...] including noise and non-contributory columns to simulate the complexity of real-world data environments"

Data Preprocessing and Feature Engineering:

- ❖ Dropping Columns : Before evaluating various Models, dropping of the feature column 'Browser Refresh Rate' gave better results as if it was of no use
- ❖ Yeo–Johnson (Power) Transform:
 - ★ A Yeo–Johnson power transform was applied to numeric features to make them more Gaussian-like (stabilizing variance and reducing skew). Such transforms are useful when predictors are highly skewed or heteroscedastic.
 - ★ Since we observed a reduction in accuracy upon applying of the Yeo-Johnson Transformation in Logistic Regression Model , it can be incited that:
 - Certain features, such as *Cart_value*, may have inherent thresholds that become obscured after transformation.
 - Scaling irrelevant or noisy features can unintentionally amplify the noise instead of enhancing the meaningful signal.
- ❖ Feature Scaling: Tree-based methods (Random Forest, Boosting) split on feature thresholds and are *invariant* to monotonic transformations so they do not require scaling.

Model Performance Comparison

Model/Pipeline	Accuracy	F1 Score	Precision	Recall
Logistic Regression (with YJ transform + scaling)	0.7711	0.565	0.663	0.492
Logistic Regression (no YJ transform; scaled features)	0.782	0.588	0.686	0.514
Random Forest (no scaling on features)	0.78	0.610	0.630	0.590

Voting Ensemble (SVM, KNN, LR, DT, RF)	0.800	0.670	0.670	0.680
AdaBoost (Decision Trees, no scaling)	0.80	0.6245	0.7327	0.54
AdaBoost (with GridSearch CV tuning)	0.7933	0.6543	0.6617	0.6471
Gradient Boosting (no scaling)	0.795	0.640	0.660	0.620
Support Vector Machine (scaled data)	0.770	0.550	0.670	0.470

Logistic Regression

With Power Transform & Scaling:

- ❖ In the first pipeline, numeric features were Yeo–Johnson transformed and min–max scaled before training a logistic model with L2 regularization ($C=1.0$). This model achieved 77.11% accuracy ($F1=0.565$, precision=0.663, recall=0.492). **The F1 and recall were relatively low, indicating it was conservative in predicting purchase (fewer false positives, more false negatives).**

Without Yeo–Johnson Transform:

- ❖ Surprisingly, removing the Yeo–Johnson step (but keeping scaling) **improved performance. Why? Logistic regression does *not* assume input features are normally distributed.**
- ❖ The Yeo–Johnson transform enforces Gaussian-like distributions, which can be counterproductive if the original feature distributions already correlate well with the log-odds. In other words, the transformation changed the features (albeit monotonically) and apparently **distorted the predictive signal** for this task. This is consistent with the principle that Yeo–Johnson is mainly useful for meeting regression assumptions or stabilizing variance, neither of which is strictly necessary for logistic classification. Additionally, scaling already ensures numeric stability, so the extra transform was redundant. The result suggests that for this

dataset, the original features contained linear signals in their raw form, and making them “more Gaussian” did not yield additional benefit. Indeed, logistic regression’s performance often saturates around the dataset’s inherent information limit

Support Vector Machines and KNNs

- ❖ **Scaling Effects:** In summary, distance-based models (KNN/SVM) indeed require properly scaled features. We observed that the ensemble including SVM/KNN (with scaled data) performed better than SVM alone, suggesting these learners provided diverse decision boundaries. Without scaling, SVM/KNN would have had even lower accuracy (not shown). This contrasts with tree models (below) which naturally handled the raw feature scales.
-

Decision Tree / Random Forest

- ❖ **Random Forest:**
 - ★ Using 100 trees (min_samples_split=5) on unscaled data yielded ~77.11% accuracy initially, and tuned parameters gave 78%. These models used raw feature values (only one-hot encoding, no scaling). Random forests inherently ignore feature scale: splits are based on threshold comparisons, not distance metrics. A decision tree “does not require feature scaling or normalization, as they are invariant to monotonic transformations. Moreover, trees handle non-normal or skewed distributions naturally, so the absence of scaling or Gaussianization transforms did not hurt.
 - ★ **Feature Importance:** Random forests also implicitly perform feature selection: they will preferentially split on predictive features (e.g., *Cart_value* or *Pages_viewed*) and largely ignore irrelevant ones (e.g., *Browser_Refresh_Rate*). This helps robustness to noisy features.
 - ★ **Performance:** The RF’s accuracy (~78%) was on par with logistics. Its F1 (0.61) was slightly higher, and recall (0.59) better than logistic (~0.5), indicating it predicted more positives correctly. This may be because tree splits can capture nonlinear interactions or thresholds (e.g. “if *Cart_value* > X then predict purchase”), which simple logistic cannot. However, RF precision (~0.63) was lower than its recall, indicating more false positives. Overall, RF delivered competitive performance without heavy tuning or scaling.
-

Boosting Algorithms

❖ AdaBoost (Decision Stumps/Trees):

- AdaBoost (with unscaled features) achieved **80.0% accuracy** ($F1 \approx 0.6245$) — higher than any single model here — and after GridSearch reached 79.33% ($F1 \approx 0.6543$). AdaBoost builds an ensemble of weak learners (typically shallow trees) by focusing on previously misclassified examples
- Its iterative reweighting “ensures that subsequent weak learners focus on the hardest examples, gradually improving performance”
- The final model is a weighted vote of all weak classifiers. Importantly, like random forests, AdaBoost uses decision trees as base learners, so **it is invariant to feature scaling**. As with random forests, “decision trees... only need absolute values for branching”, so no normalization is required.
- The advantage of AdaBoost is that it often **reduces both bias and variance** by combining many weak models

❖ Gradient Boosting;

- ★ The gradient-boosted trees (e.g. XGBoost/GBDT) achieved about **79–79.5% accuracy** ($F1 \approx 0.64$). Like AdaBoost, GB uses trees as weak learners and builds them additively, correcting residual errors at each step. It similarly requires **no feature scaling and handles skewed data**.
- ★ Its performance was close to AdaBoost. Slight differences can arise from how errors are minimized (residual gradients vs AdaBoost’s weights), but both boosting methods generally outperform a single random forest or logistic by flexibly fitting complex relationships.

❖ Why Boosting Excelled:

- ★ Boosting algorithms consistently yielded top accuracy (~80%). This is expected: ensembling via boosting “minimizes both bias and variance” by emphasizing difficult samples. In particular, AdaBoost’s dramatic weight-updates on misclassified points help it learn patterns that a single model misses. The result is a strong classifier that often beats individual models on structured data like this. Importantly, because boosting uses trees, it inherently ignores feature scale and distribution. As one explanation notes, tree-based ensembles can adapt to any structure “–linear or nonlinear, skewed or symmetric–” without normalizing inputs.

Ensemble Voting Classifier

- ❖ **Setup & Performance:** A final voting ensemble combined SVM, KNN, logistic, decision tree, and random forest (each as a base learner) to predict by majority vote. This diverse model achieved the **highest accuracy of 80.0%** and balanced $F1=0.67$, $\text{precision}=0.67$, $\text{recall}=0.68$. It also had the best recall, indicating it caught the most true positives of all methods.
- ❖ **Why It Worked:** Voting ensembles improve over single models by aggregating their predictions. The presence of scaled vs unscaled learners in the vote did not prevent high accuracy. The ensemble's superior performance suggests the dataset's limiting factor is not feature scaling but intrinsic signal: by combining models, we got the most information out of the data.

Conclusions

- ❖ Across all experiments, **ensemble methods (voting, boosting)** delivered the best accuracy (~80%). This is expected: combining multiple perspectives usually yields gains. The voting classifier aggregated linear and nonlinear models to balance precision and recall.
- ❖ AdaBoost achieved similar accuracy by sequentially correcting errors of decision trees. Random Forest and Gradient Boosting (tree ensembles) performed very well without any feature scaling due to their scale-invariance.
- ❖ Linear models (logistic, SVM) reached ~77–78% when properly preprocessed. The Yeo–Johnson transform was unnecessary and in fact reduced logistics accuracy, highlighting that **normalizing feature distribution is not a general improvement for classification**. Instead, careful use of regularization and model tuning proved more important for logistic performance.
- ❖ **Range of Model Accuracies:**
 - All models fell in the 75–80% range. This consistency implies the task's difficulty sets a performance ceiling: **there may be inherent noise or overlapping classes**. Many classifiers (with different biases) converged near this accuracy, suggesting **limited additional signal to exploit**. The ensemble slightly surpasses that by leveraging every model's decisions. In summary, **all models hitting ~80% indicates we have approached the data's predictive limit**. Additional accuracy gains would likely require **better features or more data, rather than alternative algorithms**.