Marketing Analytics

Superstore Marketing Campaign

CUSTOMER CHURN PREDICTIVE MODELING

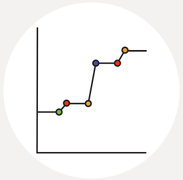
AGENDA

Objective

Variables

Data Cleaning

Descriptive Statistics



Correlation Analysis

Graphical Representation

Result

Insights & Recommendations

Objective

Develop a predictive model using decision trees to identify customers at high risk of churn based on their purchasing patterns and engagement metrics.

Evaluation Metrics:

Accuracy, Precision and Recall, F1 Score, Confusion Matrix etc.

Variables:

Dependant Variable-Is high risk churn

Independent Variables- NumStorePurchases , NumDealsPurchases, NumWebPurchases, etc.

Independant Variables-

- Response
- ID
- Year Birth
- Complain
- KidHome
- Dt_Customer
- TeenHome
- Education
- Marital
- Education
- FishProduct
- Wines
- SweetProduct
- Fruits
- Income
- MeatProduct
- GoldProds
- Deals
- Catalog
- StoreBuy
- WebVisits
- WebBuy

Dataset - Superstore

```
# Import dataset. Display first 10 rows
superstore = pd.read_csv('superstore_data.csv')
superstore.head(10)
```

	Id	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	MntFishProducts	MntSweetProducts
0	1826	1970	Graduation	Divorced	84835.0	0	0	6/16/2014	0	189	...	111	189
1	1	1961	Graduation	Single	57091.0	0	0	6/15/2014	0	464	...	7	0
2	10476	1958	Graduation	Married	67267.0	0	1	5/13/2014	0	134	...	15	2
3	1386	1967	Graduation	Together	32474.0	1	1	11/5/2014	0	10	...	0	0
4	5371	1989	Graduation	Single	21474.0	1	0	8/4/2014	0	6	...	11	0
5	7348	1958	PhD	Single	71691.0	0	0	3/17/2014	0	336	...	240	32
6	4073	1954	2n Cycle	Married	63564.0	0	0	1/29/2014	0	769	...	15	34
7	1991	1967	Graduation	Together	44931.0	0	1	1/18/2014	0	78	...	0	0
8	4047	1954	PhD	Married	65324.0	0	1	11/1/2014	0	384	...	21	32
9	9477	1954	PhD	Married	65324.0	0	1	11/1/2014	0	384	...	21	32

10 rows × 22 columns

Data Cleaning - Determining High/Low Risk

```
superstore['Risk_Status'] = superstore['Recency'].apply(lambda recency: "High Risk" if recency >= 30 else "Low Risk")
```

```
superstore['Risk_Status']
```

```
0         Low Risk
1         Low Risk
2         Low Risk
3         Low Risk
4         Low Risk
           ...
2235     High Risk
2236     High Risk
2237     High Risk
2238     High Risk
2239     High Risk
Name: Risk_Status, Length: 2240, dtype: object
```

Data Cleaning - Dropping Unnecessary & Missing Values

```
# Dropping unneccesary variables columns, modify if needed
superstore = superstore.drop(['Id','Year_Birth','Dt_Customer','Recency'],axis=1)
```

```
# Check for missing values
superstore.isnull().sum() #sum the number of rows that has missing/null values
```

```
Education              0
Marital_Status         0
Income                24
Kidhome                0
Teenhome               0
MntWines               0
MntFruits              0
MntMeatProducts        0
MntFishProducts        0
MntSweetProducts       0
MntGoldProds           0
NumDealsPurchases      0
NumWebPurchases        0
NumCatalogPurchases    0
NumStorePurchases      0
NumWebVisitsMonth      0
Response               0
Complain               0
Risk_Status            0
dtype: int64
```

Data Cleaning - Dropping Unnecessary & Missing Values

```
# Drop the missing values
superstore = superstore.dropna(axis=0) # dropna: remove missing values from the dataset, axis=0 means dropping by rows.
```

```
# Check for missing values after removing na
superstore.isnull().sum()
```

```
Education              0
Marital_Status         0
Income                 0
Kidhome                0
Teenhome               0
MntWines               0
MntFruits              0
MntMeatProducts        0
MntFishProducts        0
MntSweetProducts       0
MntGoldProds           0
NumDealsPurchases      0
NumWebPurchases        0
NumCatalogPurchases    0
NumStorePurchases      0
NumWebVisitsMonth      0
Response               0
Complain               0
Risk_Status            0
dtype: int64
```

Data Cleaning - Determining Likelihood of Customers Churning (High/Low Risk)

In [34]:
```
# Check for balance in target variable
superstore['Risk_Status'].value_counts(normalize=True)
```

Out[34]:
```
High Risk    0.690433
Low Risk     0.309567
Name: Risk_Status, dtype: float64
```

There are 70% customers in this dataset that belong to high-risk category (not returning after 30 days), and 30% that belong to low-risk category (most recent purchase is under 30 days).

Data Cleaning-Changing Categorical Variables into ‘Category’ types

```
# Changing categorical variables into 'category' types
superstore['Education'] = superstore['Education'].astype('category')
superstore['Marital_Status'] = superstore['Marital_Status'].astype('category')
superstore['Risk_Status'] = superstore['Risk_Status'].astype('category')
```

```
superstore['is_high_risk'] = superstore['Risk_Status'].map({'High Risk': 1, 'Low Risk': 0})
```

- Superstore dataset are being converted into the 'category' data type. 'Education', 'Marital_Status', and 'Risk_Status' into 'category' types
- A new column named 'is_high_risk' is being created in the **superstore** dataset. The values in this column depend on the values in the 'Risk_Status' column. If 'Risk_Status' is 'High Risk', the corresponding value in 'is_high_risk' will be 1. If 'Risk_Status' is 'Low Risk', the corresponding value in 'is_high_risk' will be 0.

Descriptive statistics

```
superstore.describe()
```

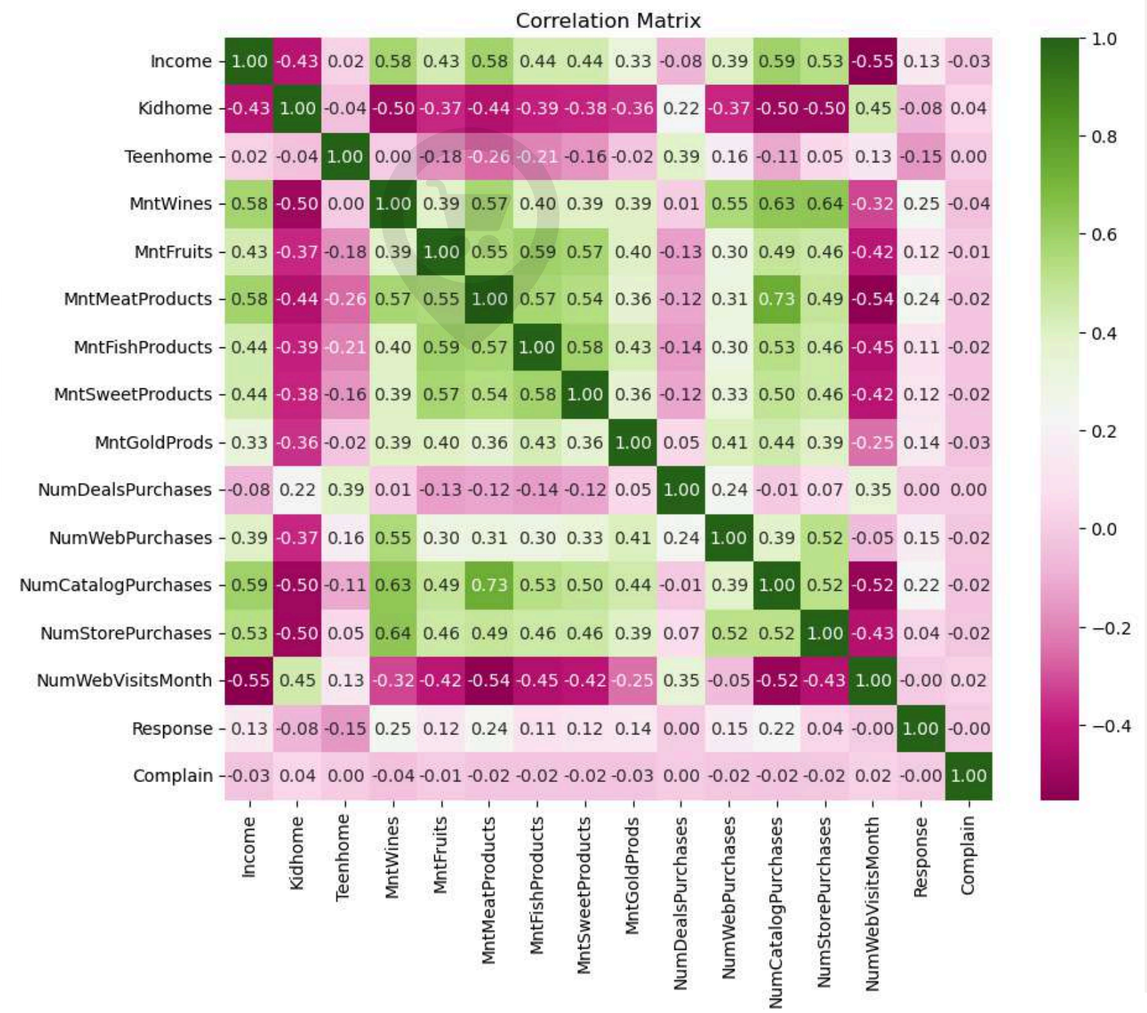
	Income	Kidhome	Teenhome	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts	MntGoldProds	NumDealsPurchases	NumWebPurchases	NumCatalogPurchases	NumStorePurchases	NumWebVisitsMonth	Response	Complain
count	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000	2216.000000
mean	52247.251354	0.441787	0.505415	305.091606	26.356047	166.995939	37.637635	27.028881	43.965253	2.323556	4.085289	2.671029	5.800993	5.319043	0.150271	0.009477
std	25173.076661	0.536896	0.544181	337.327920	39.793917	224.283273	54.752082	41.072046	51.815414	1.923716	2.740951	2.926734	3.250785	2.425359	0.357417	0.096907
min	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	35303.000000	0.000000	0.000000	24.000000	2.000000	16.000000	3.000000	1.000000	9.000000	1.000000	2.000000	0.000000	3.000000	3.000000	0.000000	0.000000
50%	51381.500000	0.000000	0.000000	174.500000	8.000000	68.000000	12.000000	8.000000	24.500000	2.000000	4.000000	2.000000	5.000000	6.000000	0.000000	0.000000
75%	68522.000000	1.000000	1.000000	505.000000	33.000000	232.250000	50.000000	33.000000	56.000000	3.000000	6.000000	4.000000	8.000000	7.000000	0.000000	0.000000
max	666666.000000	2.000000	2.000000	1493.000000	199.000000	1725.000000	259.000000	262.000000	321.000000	15.000000	27.000000	28.000000	13.000000	20.000000	1.000000	1.000000

- Income: Average income is $52,247 with a wide range.
- Household Composition: On average, households have 0.44 children and 0.51 teenagers.
- Spending on Products: Varied spending on wines, fruits, meat, fish, and sweet products.
- Other Purchasing Behavior: Customers make purchases across various channels.
- Web Visits and Response: Average of 5.32 web visits per month; 15.03% responded to a campaign.
- Complaints: Very few customers (0.95%) filed complaints.

Correlation Matrix

```
# Examine Correlations between numeric variables for Multicollinearity
# select_dtypes(include=['number']) selects only numeric variables, corr() display correlation
df_corr = superstore.select_dtypes(include=['number']).corr()

# Plotting the heatmap for correlation
plt.figure(figsize=(10,8))
sns.heatmap(df_corr, annot=True, fmt=".2f", cmap='PiYG')
plt.title('Correlation Matrix')
plt.show()
```



- Positive correlation with MntMeatProducts (0.58), MntWines (0.58), and MntFruits (0.43), suggesting higher-income individuals spend more on these categories.
- Positive correlations with web (0.39), catalog (0.59), and store purchases (0.53), indicating higher-income customers are more active across all platforms.
- Negative correlation with NumWebVisitsMonth (-0.55), indicating wealthier customers visit the website less frequently but make larger purchases per visit.
- Negative correlation with Kidhome (-0.23) and positive with Teenhome (0.35), reflecting typical family life stages.
- Slight negative correlation with NumDealsPurchases (-0.08), suggesting higher-income individuals are less influenced by deals and promotions.

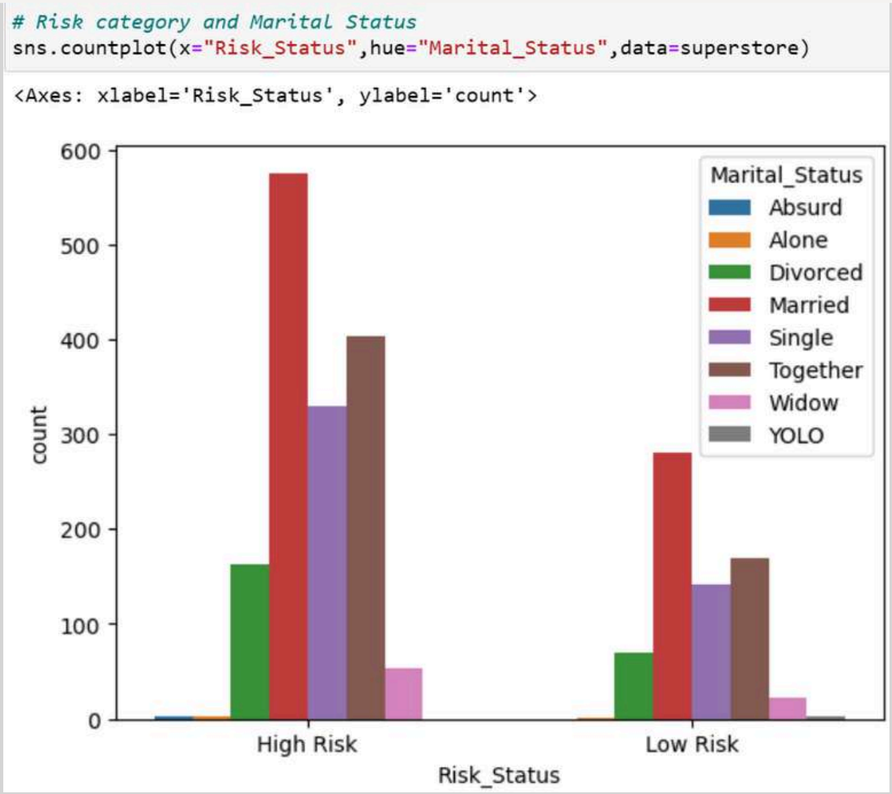
Dropping Highly Correlated Variable based on the Correlation Matrix-

```
superstore = superstore.drop(['MntMeatProducts'],axis=1)
```

Graphical Display: Differences in distributions of data among the low and high-risk customers

```
# Risk category and Marital Status
sns.countplot(x="Risk_Status",hue="Marital_Status",data=superstore)
```

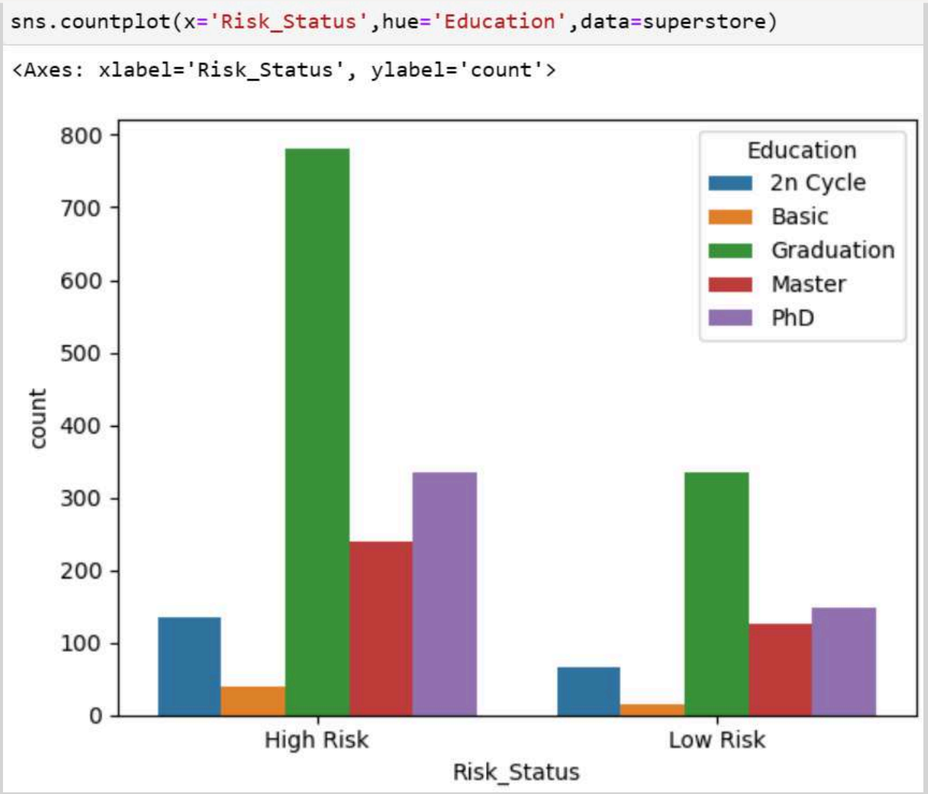
```
<Axes: xlabel='Risk_Status', ylabel='count'>
```



"Married" individuals mostly high-risk

```
sns.countplot(x='Risk_Status',hue='Education',data=superstore)
```

```
<Axes: xlabel='Risk_Status', ylabel='count'>
```

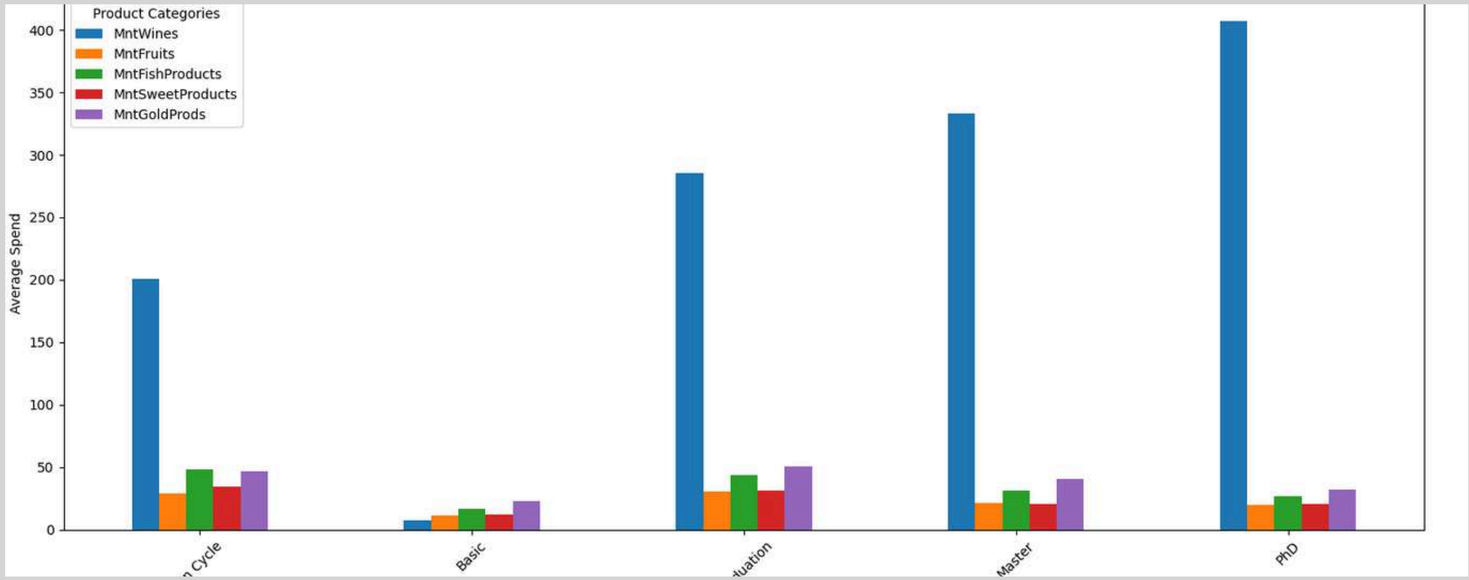


"Graduation" level dominates high-risk group
"PhD" and "Graduation" prevail in low-risk.

```
grouped_data = superstore.groupby('Education')[['MntWines', 'MntFruits','MntFishProducts', 'MntSweetProducts', 'MntGoldProds']].mean()

# Now, plot the data
grouped_data.plot(kind='bar', figsize=(15, 7), stacked=False)

plt.title('Average Spend on Products by Education Level')
plt.ylabel('Average Spend')
plt.xlabel('Education Level')
plt.xticks(rotation=45)
plt.legend(title='Product Categories')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without overlapping
plt.show()
```



High spending on wines by "PhD" and "Masters"; no significant spending variation across education levels in other categories.

Conclusion: Little variation in churn risk distribution across variables; consider dropping them from further analysis.

Dropping Variables that Dont Contribute to the High/Low Risk Distributuion

```
# Now, plot the data
grouped_data.plot(kind='bar', figsize=(15, 7), stacked=False)

plt.title('Average Spend on Products by Education Level')
plt.ylabel('Average Spend')
plt.xlabel('Education Level')
plt.xticks(rotation=45)
plt.legend(title='Product Categories')
plt.tight_layout()  # Adjusts the plot to ensure everything fits without overlapping
plt.show()
```

Based on the charts, I will drop Education and Marital Status because they don't really show any differences in distribution among the High and Low Risk of Churn.

```
df = superstore.drop(['Education','Marital_Status','Risk_Status'],axis=1)
```

Model Building-

#4. Model Building

1. Dummy Coding Variables for categorical variables (if applicable).
2. Define x & y
3. Splitting into training/testing
4. Fit the model
5. Performance Assessment with Confusion Matrix
6. Find the optimal number of splits and leaves
7. Refit the model
8. Performance Assessment

```
# Define X and Y
y =df.is_high_risk
x=df.drop(['is_high_risk'], axis=1)
```

```
# Splitting data into testing and training set
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2, random_state=23823)
```

Decision Tree Model-

```
# Check for balance in target variable in training & testing set
print("Training:", ytrain.value_counts(normalize=True))
print("Testing:", ytest.value_counts(normalize=True))
```

```
Training: 1    0.690745
0    0.309255
Name: is_high_risk, dtype: float64
Testing: 1    0.689189
0    0.310811
Name: is_high_risk, dtype: float64
```

```
# Decision Tree Model

tree1 = DecisionTreeClassifier(criterion="entropy",max_depth=3, min_samples_split=50, min_samples_leaf=25)
tree1.fit(xtrain, ytrain)

# Predict the target in the test data and display accuracy rate
pred1 = tree1.predict(xtest)
```

Decision Tree Model Graph-

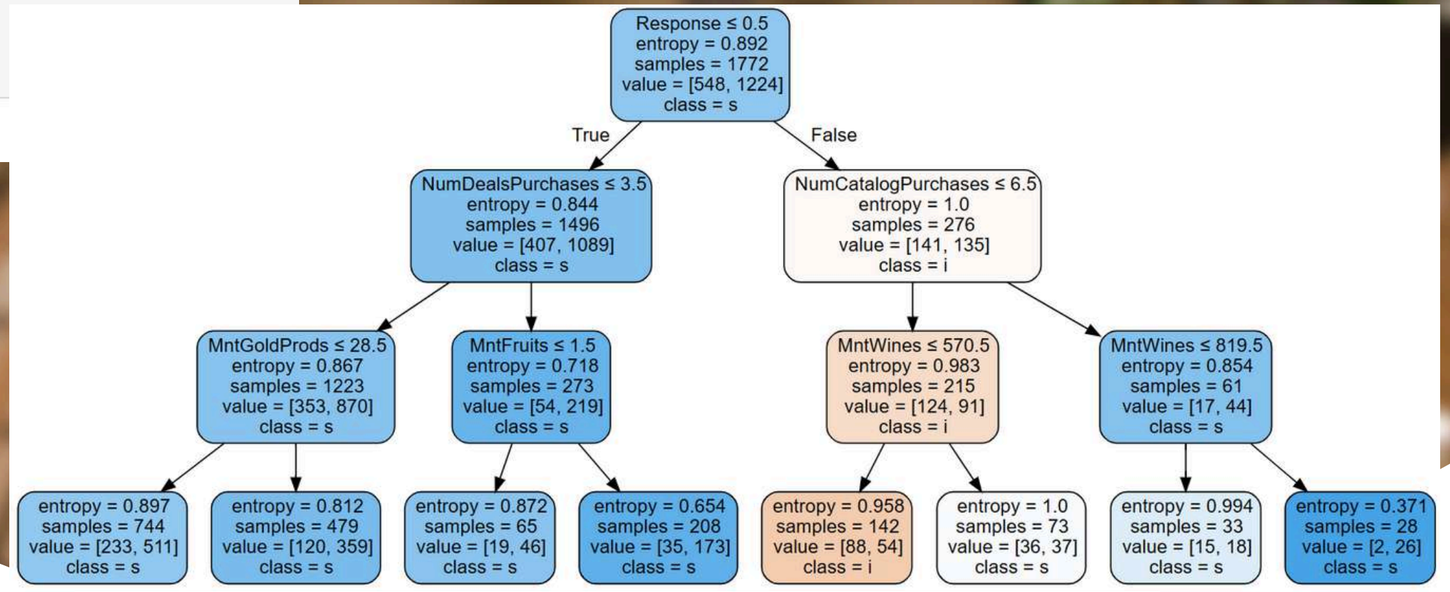
```
# Create a visualization of the decision tree
dot_data = export_graphviz(tree1, out_file=None,
                           feature_names=x.columns,
                           class_names=y.name,
                           filled=True, rounded=True,
                           special_characters=True
                           )

# Modify the dot data to increase font size and adjust graph size
dot_data = dot_data.replace('fontsize=10', 'fontsize=12')
dot_data = dot_data.replace('nodesep=0.1', 'nodesep=0.5')  # Increase node separation
dot_data = dot_data.replace('ranksep=0.1', 'ranksep=0.5')  # Increase rank separation

# Create a graph from the modified dot data
graph = graphviz.Source(dot_data)
graph.format = 'png'
graph.render('decision_tree_visualization', cleanup=True)


graph
```

```
<graphviz.sources.Source at 0x1c644946f10>
```



Performance Evaluation Report-

```
# Performance Evaluation

print("Accuracy for tree:",accuracy_score(ytest, pred1))
print("Classification Report:", "\n", classification_report(ytest,pred1))
```

```
Accuracy for tree: 0.6914414414414415
Classification Report:
                precision    recall  f1-score   support

           0       0.52      0.11      0.18       138
           1       0.70      0.95      0.81       306

    accuracy                           0.69       444
   macro avg       0.61      0.53      0.49       444
weighted avg       0.65      0.69      0.61       444
```

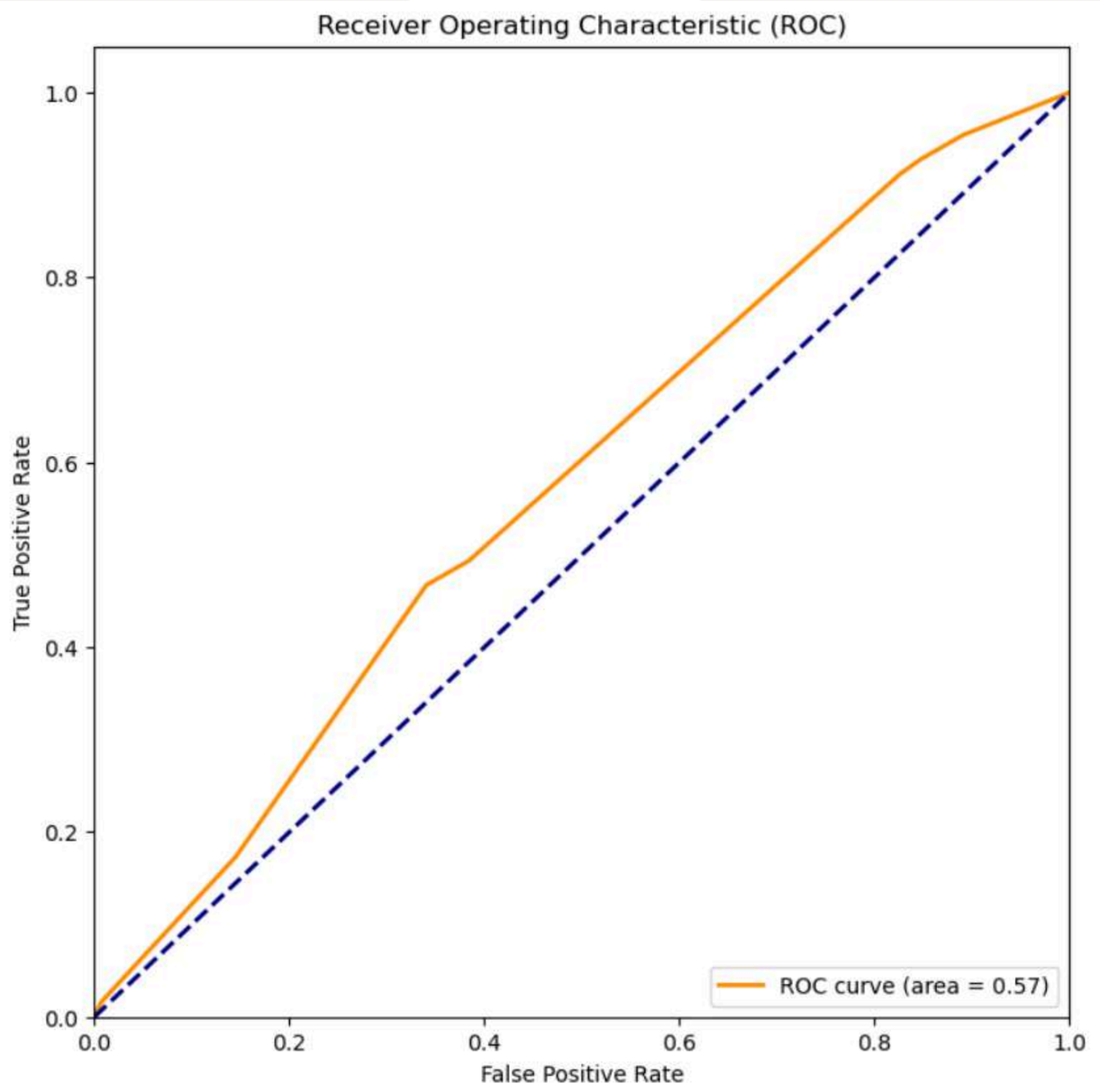
Plotting AUC & ROC Curve

```
# Instead of using predict, use predict_proba to get the probabilities
probas_ = tree1.predict_proba(xtest)[:, 1]  # We are interested in the probabilities for the positive class

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(ytest, probas_)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

ROC Curve-


Receiver Operating Characteristic (ROC)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.57)

Parameter Grid to Fit the Model

```
# Hyperparameter tuning for decision tree
# Create the parameter grid and fit the model

tree_param = {'max_depth': range(5,16,5), 'min_samples_leaf': range(50,151,30),
         'min_samples_split': range(50,151,30), 'criterion': ["entropy"]}
tree2 = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator=tree2, param_grid=tree_param, cv=5, verbose = 1)
grid_search.fit(xtrain, ytrain)
```

```
Fitting 5 folds for each of 48 candidates, totalling 240 fits

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['entropy'], 'max_depth': range(5, 16, 5),
                         'min_samples_leaf': range(50, 151, 30),
                         'min_samples_split': range(50, 151, 30)},
             verbose=1)
```

Accuracy Score-

```
# CV results
result = pd.DataFrame(grid_search.cv_results_)
result.head(2)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	param_min_samples_leaf	param_min_samples_split
0	0.021019	0.010768	0.011293	0.018175	entropy	5	50	50
1	0.015327	0.004918	0.004629	0.006038	entropy	5	50	80

```
# print the optimal accuracy score and hyperparameters

print("best accuracy", grid_search.best_score_)
print(grid_search.best_estimator_)
```

```
best accuracy 0.7065441234980504
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=80,
                       min_samples_split=50)
```

Parameter Grid to Fit the Model

```
# Refit the decision tree model with optimal hyperparameters
tree2 = DecisionTreeClassifier(criterion="entropy", max_depth=5, min_samples_leaf=80,
                               min_samples_split=50, random_state=123)
tree2.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=80,
                       min_samples_split=50, random_state=123)
```

Decision Tree Graph

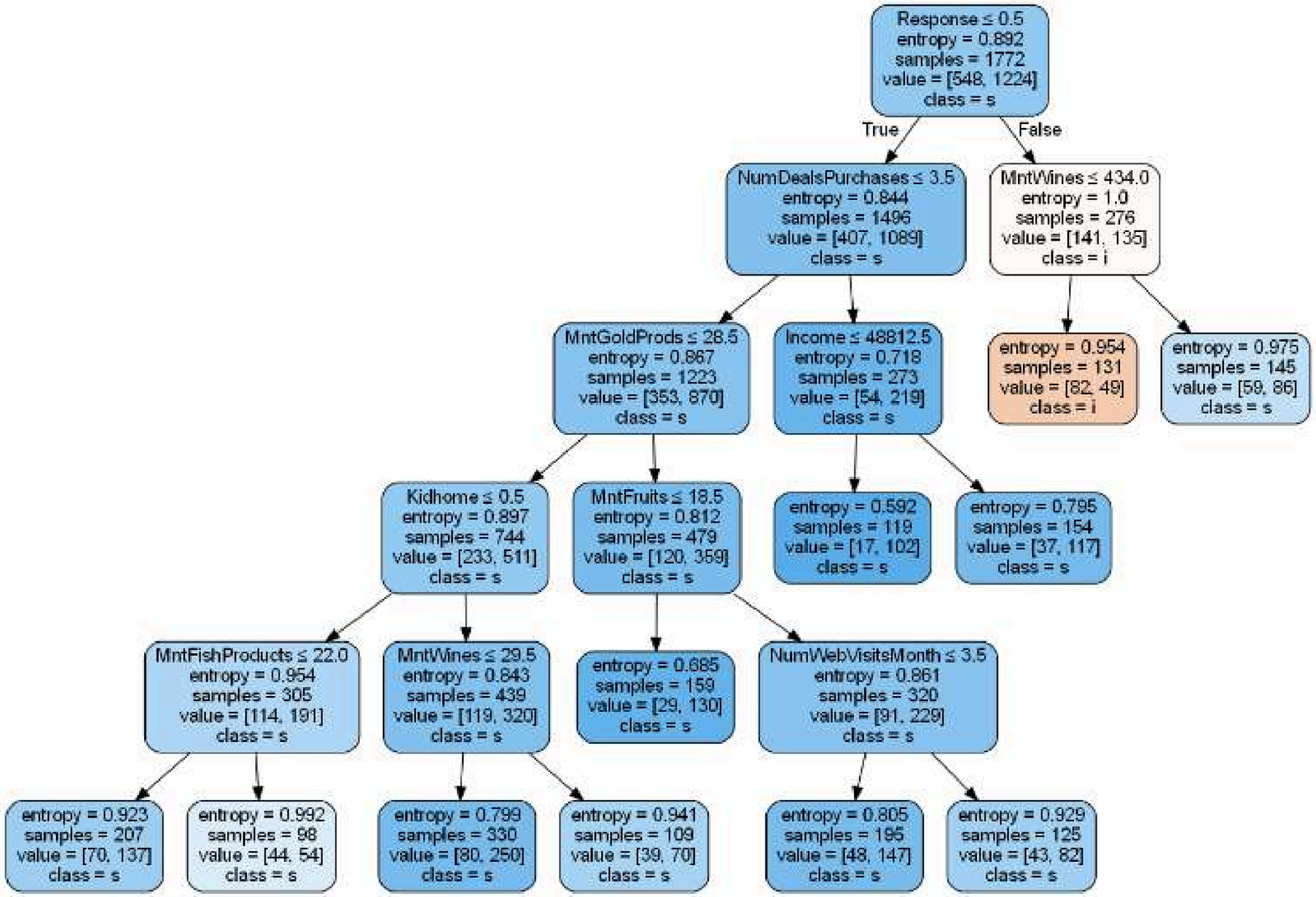
```
# Create a visualization of the decision tree
dot_data = export_graphviz(tree2, out_file=None,
                           feature_names=x.columns,
                           class_names=y.name,
                           filled=True, rounded=True,
                           special_characters=True
                           )

# Modify the dot data to increase font size and adjust graph size
dot_data = dot_data.replace('fontsize=10', 'fontsize=12')
dot_data = dot_data.replace('nodesep=0.1', 'nodesep=0.5')  # Increase node separation
dot_data = dot_data.replace('ranksep=0.1', 'ranksep=0.5')  # Increase rank separation

# Create a graph from the modified dot data
graph = graphviz.Source(dot_data)
graph.format = 'png'
graph.render('decision_tree_visualization', cleanup=True)


graph
```

Decision Tree Graph



```
# Predict the target in the test data and display accuracy rate
pred2 = tree2.predict(xtest)
```

```
# Performance Evaluation

print("Accuracy for tree:",accuracy_score(ytest, pred2))
print("Classification Report:", "\n", classification_report(ytest,pred2))
```

```
Accuracy for tree: 0.6959459459459459
Classification Report:
               precision    recall  f1-score   support

           0       0.55      0.13      0.21       138
           1       0.71      0.95      0.81       306

    accuracy                           0.70       444
   macro avg       0.63      0.54      0.51       444
weighted avg       0.66      0.70      0.62       444
```

Classification Report

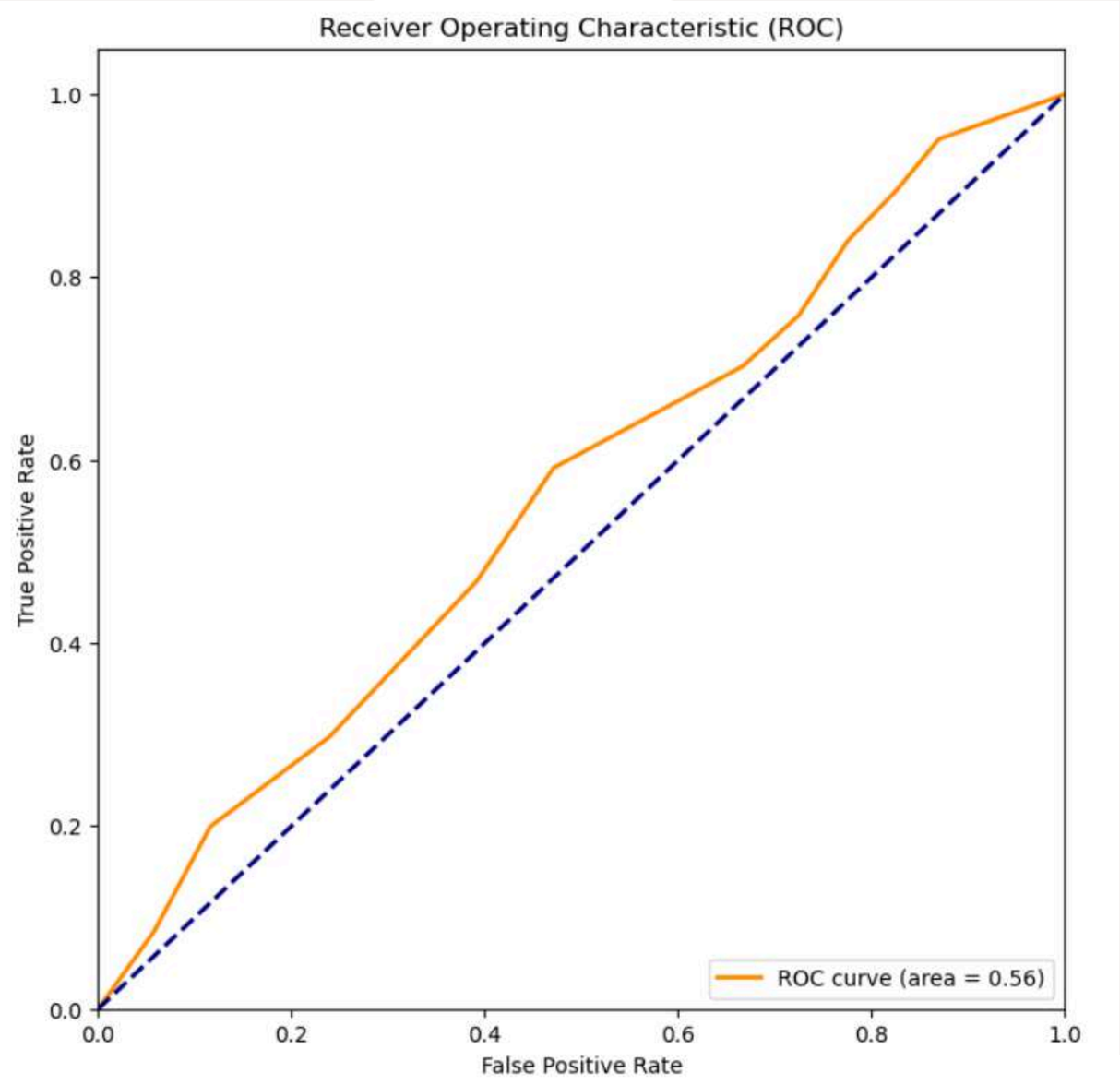
Graph Plot

```
# Instead of using predict, use predict_proba to get the probabilities
probas_ = tree2.predict_proba(xtest)[:, 1]  # We are interested in the probabilities for the positive class

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(ytest, probas_)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

ROC Curve-


Receiver Operating Characteristic (ROC)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.56)

```
#Random Forest
rf1 = RandomForestClassifier(max_features=5, random_state=20)
rf1.fit(xtrain, ytrain)
pred1 = rf1.predict(xtest)
```

```
# Performance Evaluation for Random Forest without Tuning

print("Accuracy for Random Forest:",accuracy_score(ytest, pred1))
print("Classification Report:", "\n", classification_report(ytest,pred1))
```

```
Accuracy for Random Forest: 0.6914414414414415
Classification Report:
               precision    recall  f1-score   support

           0       0.51      0.20      0.29       138
           1       0.72      0.91      0.80       306

    accuracy                           0.69       444
   macro avg       0.61      0.56      0.55       444
weighted avg       0.65      0.69      0.64       444
```

Classification Report

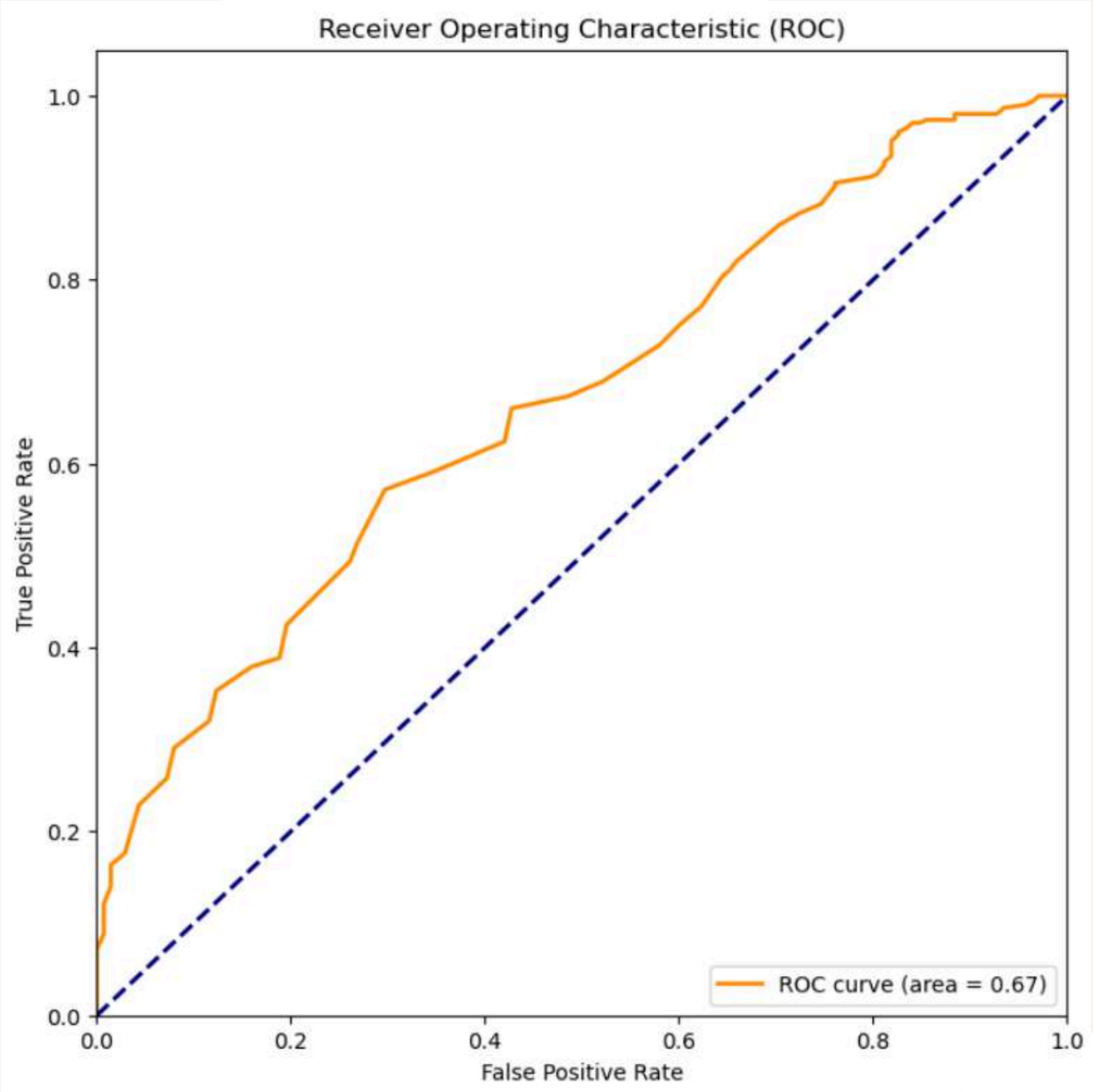
Graph Plot

```
# Instead of using predict, use predict_proba to get the probabilities
probas_ = rf1.predict_proba(xtest)[:, 1]  # We are interested in the probabilities for the positive class

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(ytest, probas_)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

ROC Curve–


Receiver Operating Characteristic (ROC)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.67)

Checking Best Model-

```
# Use random search to find the best hyperparameters (It will take some time)

rf2 = RandomForestClassifier()
rf_param = {'n_estimators': randint(50,200), 'max_depth': randint(5,20)}
search=RandomizedSearchCV(rf2, param_distributions = rf_param, n_iter=5, cv=10).fit(xtrain, ytrain)
```

```
# Create a variable for the best model
rf_best = search.best_estimator_
print('Best hyperparameters for Random Forest:',  search.best_params_)
```

```
Best hyperparameters for Random Forest: {'max_depth': 18, 'n_estimators': 128}
```

```
var_importance = pd.DataFrame({'importance': rf_best.feature_importances_}, index=x.columns)
var_importance.sort_values(by='importance', ascending=False)
```

	importance
Income	0.129254
MntWines	0.120350
MntGoldProds	0.116290
MntFishProducts	0.099591
MntFruits	0.092835
MntSweetProducts	0.089629
NumWebVisitsMonth	0.059180
NumStorePurchases	0.057968
NumWebPurchases	0.057117
NumCatalogPurchases	0.049096
NumDealsPurchases	0.048571
Response	0.041082
Teenhome	0.019407
Kidhome	0.017335

Classification Report-

```
pred2 = rf_best.predict(xtest)
```

```
# Performance Evaluation for Random Forest with Hyperparameter Tuning

print("Accuracy for Random Forest:",accuracy_score(ytest, pred2))
print("Classification Report:", "\n", classification_report(ytest,pred2))
```

```
Accuracy for Random Forest: 0.7162162162162162
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.22      0.33       138
           1       0.73      0.94      0.82       306

    accuracy                           0.72       444
   macro avg       0.67      0.58      0.57       444
weighted avg       0.69      0.72      0.67       444
```

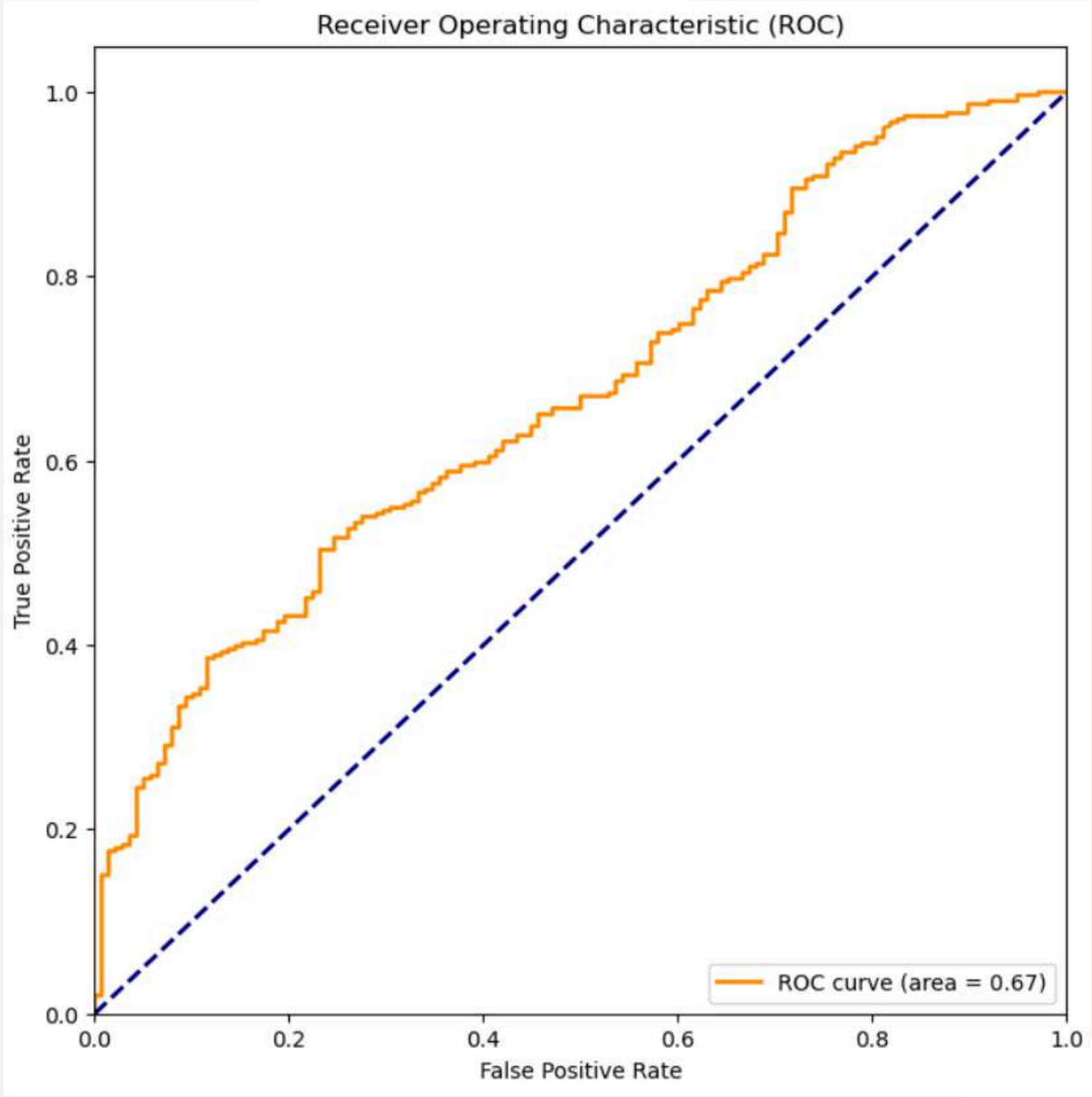
Plotting ROC Curve & AUC-

```
# Instead of using predict, use predict_proba to get the probabilities
probas_ = rf_best.predict_proba(xtest)[:, 1]  # We are interested in the probabilities for the positive class

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(ytest, probas_)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 8))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

ROC Curve–


Receiver Operating Characteristic (ROC)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.67)

Results...

Managerial Implication and Action Plan:

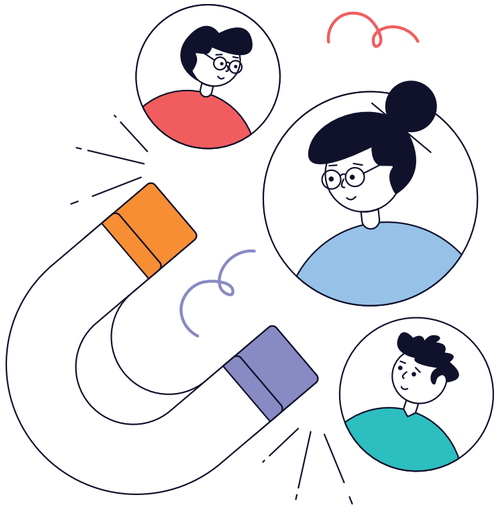
Customer Retention Strategy

Marketing efficiency

Operational Adjustments

Recommended Actions

Targeted Intervention Program



Enhance Customer Retention

Data-Driven Decision Making

Staff/Personnel Training


Thank you