```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score, cross_v

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_m

import warnings
warnings.filterwarnings('ignore')
```

```python
import pandas as pd

# URL of the dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00468/online_

# Load the dataset directly from the URL
data = pd.read_csv(url)
```

Data Pre-Processing

In this section we will make our data ready for model training. This will include:

- Encode Categorical features using dummy encoding
- Encode Boolean variables using label encoder
- Split Data into train and test set
- Scale train set using the standard scaler

```
# Encode categorical features (Month, Visitor Type) using dummy encoding

categorical = ['Month', 'VisitorType']

encoded_features = pd.get_dummies(data[categorical])
encoded_features.head(3)
```

	Month_Aug	Month_Dec	Month_Feb	Month_Jul	Month_June	Month_Mar	Month_May
0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0

Next steps: Generate code with encoded_features View recommended plots

```
# Check shape
data.shape
```

(12330, 18)

```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Administrative           12330 non-null  int64  
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64  
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64  
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object 
 11  OperatingSystems         12330 non-null  int64  
 12  Browser                  12330 non-null  int64  
 13  Region                   12330 non-null  int64  
 14  TrafficType              12330 non-null  int64  
 15  VisitorType              12330 non-null  object 
 16  Weekend                  12330 non-null  bool   
 17  Revenue                  12330 non-null  bool   
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

There are 2 Boolean, 2 Categorical and 14 Numeric Variables (7 Integers and 7 Float) in the dataset.

```
1 data.isna().sum()
```

```
Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

Double-click (or enter) to edit

There are no missing values in the dataset

```
1 data.describe()
```

	Administrative	Administrative_Duration	Informational	Informational_Du
count	12330.000000	12330.000000	12330.000000	12330
mean	2.315166	80.818611	0.503569	34
std	3.321784	176.779107	1.270156	140
min	0.000000	0.000000	0.000000	0
25%	0.000000	0.000000	0.000000	0
50%	1.000000	7.500000	0.000000	0
75%	4.000000	93.256250	0.000000	0
max	27.000000	3398.750000	24.000000	2549

- ***On average, users visit 2 Administrative pages and 31 Product Related Pages. However, there is very little to no engagement with the Informational pages.***

```
data.describe(include=['object', 'bool'])
```

	Month	VisitorType	Weekend	Revenue
count	12330	12330	12330	12330
unique	10	3	2	2
top	May	Returning_Visitor	False	False
freq	3364	10551	9462	10422

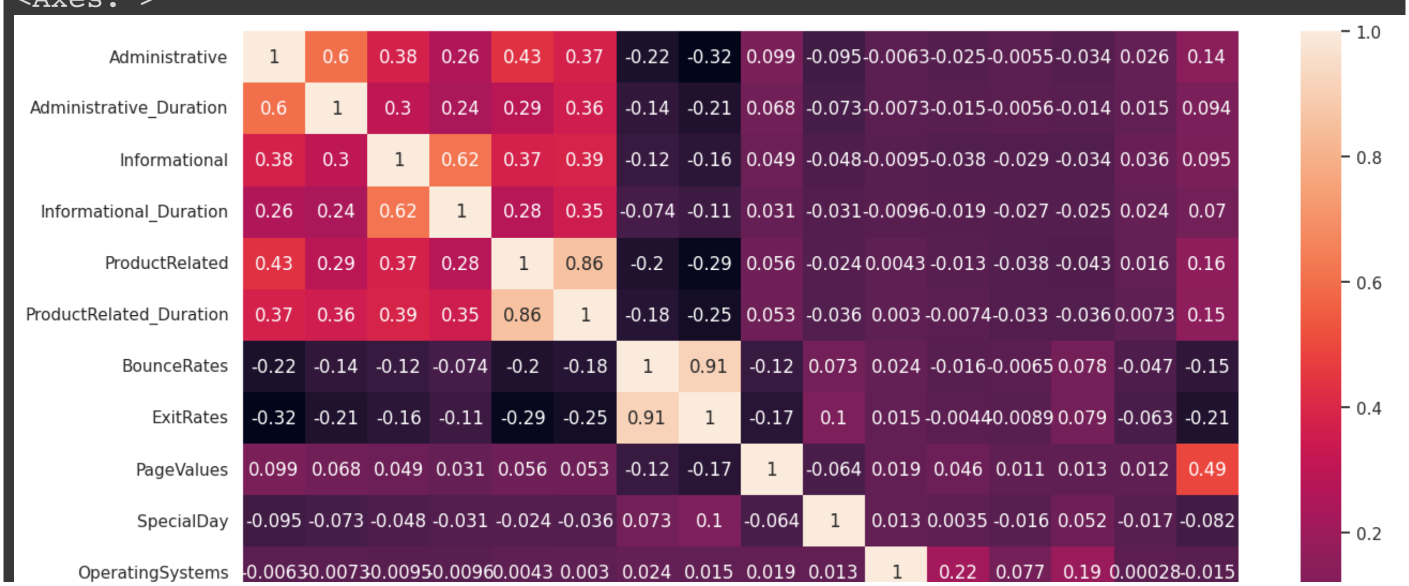
- ***Dataset contains records of 10 unique months; May occurs most frequently.***
- ***There are 3 unique Visitor Types with returning visitor being the most common type; occuring 10,551 instances.***

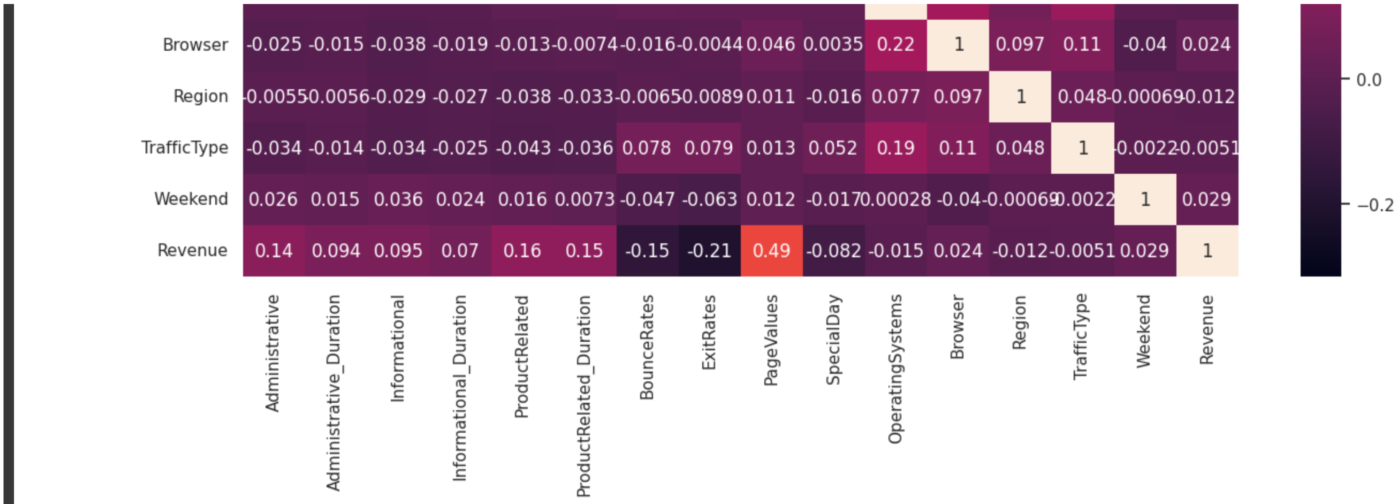
Exploratory Data Analysis

```
### Correlation Analysis
```

```
plt.figure(figsize=(15,10))
sns.heatmap(data.corr(),annot=True)
```

```
<Axes: >
```




Browser
-0.025 -0.015 -0.038 -0.019 -0.013 -0.0074 -0.016 -0.0044 0.046 0.0035 0.22 1 0.097 0.11 -0.04 0.024
Region
-0.0055 -0.0056 -0.029 -0.027 -0.038 -0.033 -0.0065 -0.0089 0.011 -0.016 0.077 0.097 1 0.048 -0.00069 -0.012
TrafficType
-0.034 -0.014 -0.034 -0.025 -0.043 -0.036 0.078 0.079 0.013 0.052 0.19 0.11 0.048 1 -0.0022 -0.0051
Weekend
0.026 0.015 0.036 0.024 0.016 0.0073 -0.047 -0.063 0.012 -0.017 0.00028 -0.04 -0.00069 -0.0022 1 0.029
Revenue
0.14 0.094 0.095 0.07 0.16 0.15 -0.15 -0.21 0.49 -0.082 -0.015 0.024 -0.012 -0.0051 0.029 1
Administrative
Administrative_Duration
Informational
Informational_Duration
ProductRelated
ProductRelated_Duration
BounceRates
ExitRates
PageValues
SpecialDay
OperatingSystems
Browser
Region
TrafficType
Weekend
Revenue
0.0
−0.2

The Heatmap shows there is little correlation among the different features with the exception of the following:

- High correlation between:

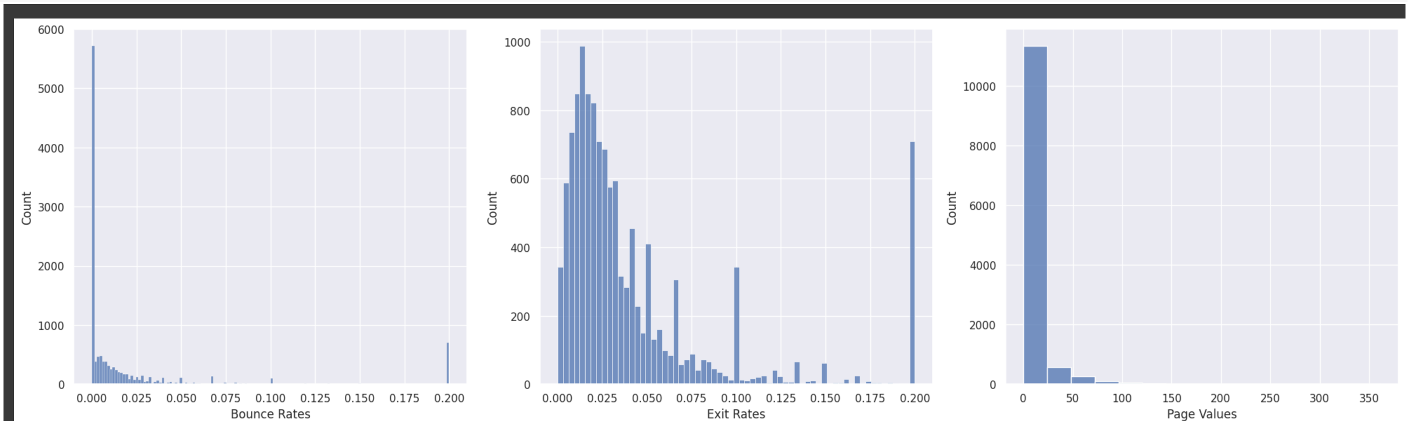
```
- BounceRates & ExitRates (0.91).
- ProductRelated & ProductRelated_Duration (0.86).
```

- Moderate Correlations:

```
- Administrative & Administrative DUration (0.6)
- Informational and Informational Duration (0.62)
- Page Values and Revenue (0.49)
```

Page Matrix analysis:

```python
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 3, figsize=(20, 6))

sns.histplot(data['BounceRates'], ax=axes[0])
axes[0].set_xlabel('Bounce Rates')

sns.histplot(data['ExitRates'], ax=axes[1])
axes[1].set_xlabel('Exit Rates')

sns.histplot(data['PageValues'], ax=axes[2])
axes[2].set_xlabel('Page Values')

plt.tight_layout()
plt.show()

```



The above distribution plots of Page Metrics show the following:

- All 3 features have distributions that are right skewed with a lot of outliers.
- The average bounce rate of most of our data points is low. This is a positive observation as high rates would identicate that visitors are not engaging with the website.
- Exit rates are higher in values than bounce rates. This is expected as we can assume that transaction confirmation pages will cause the average exit rate to increase.

Revenue Analysis

```
data.Revenue.value_counts()
```

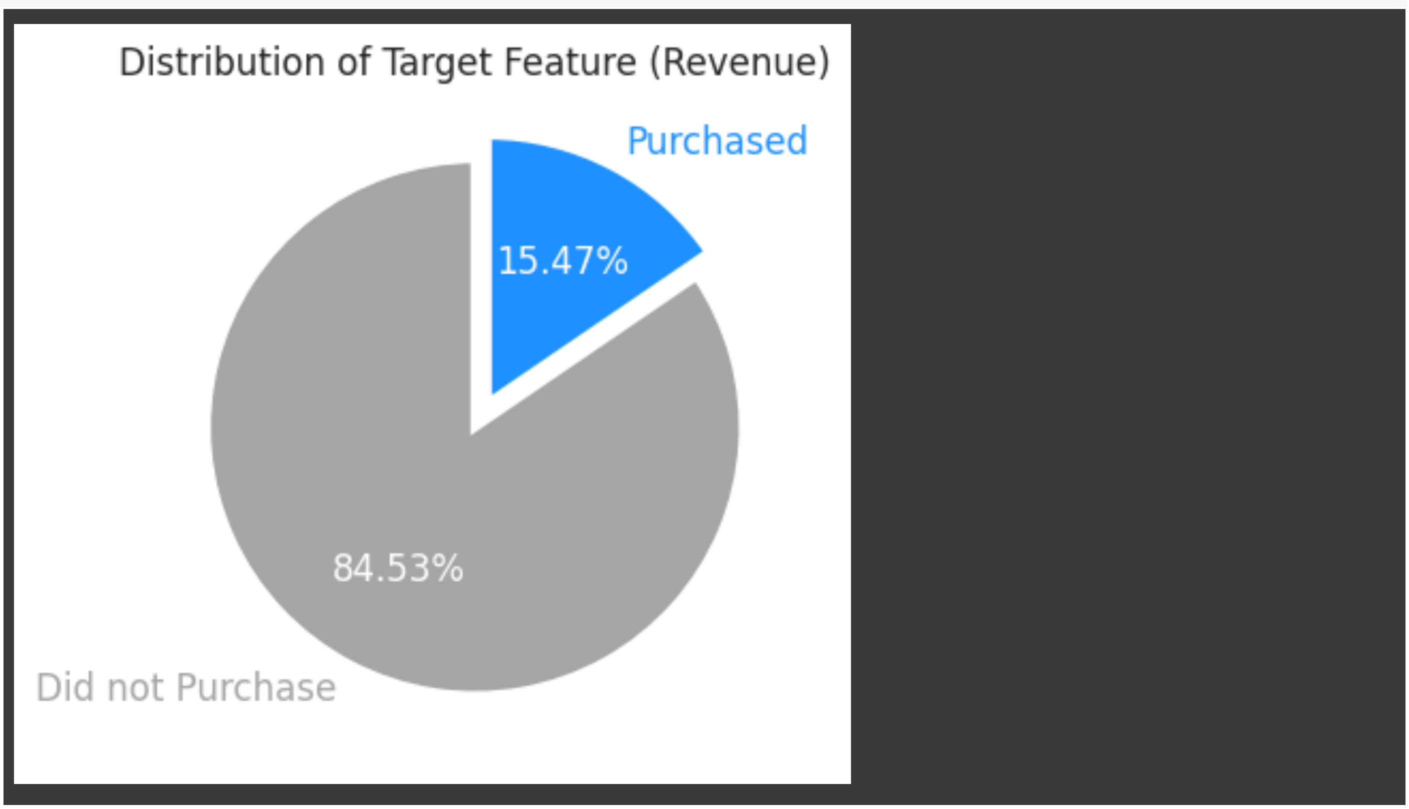
```
False    10422
True      1908
Name: Revenue, dtype: int64
```

0 represents False and 1 represents True

```
revenue_ratio = data.Revenue.value_counts(normalize=True)
revenue_ratio
```

```
False    0.845255
True     0.154745
Name: Revenue, dtype: float64
```

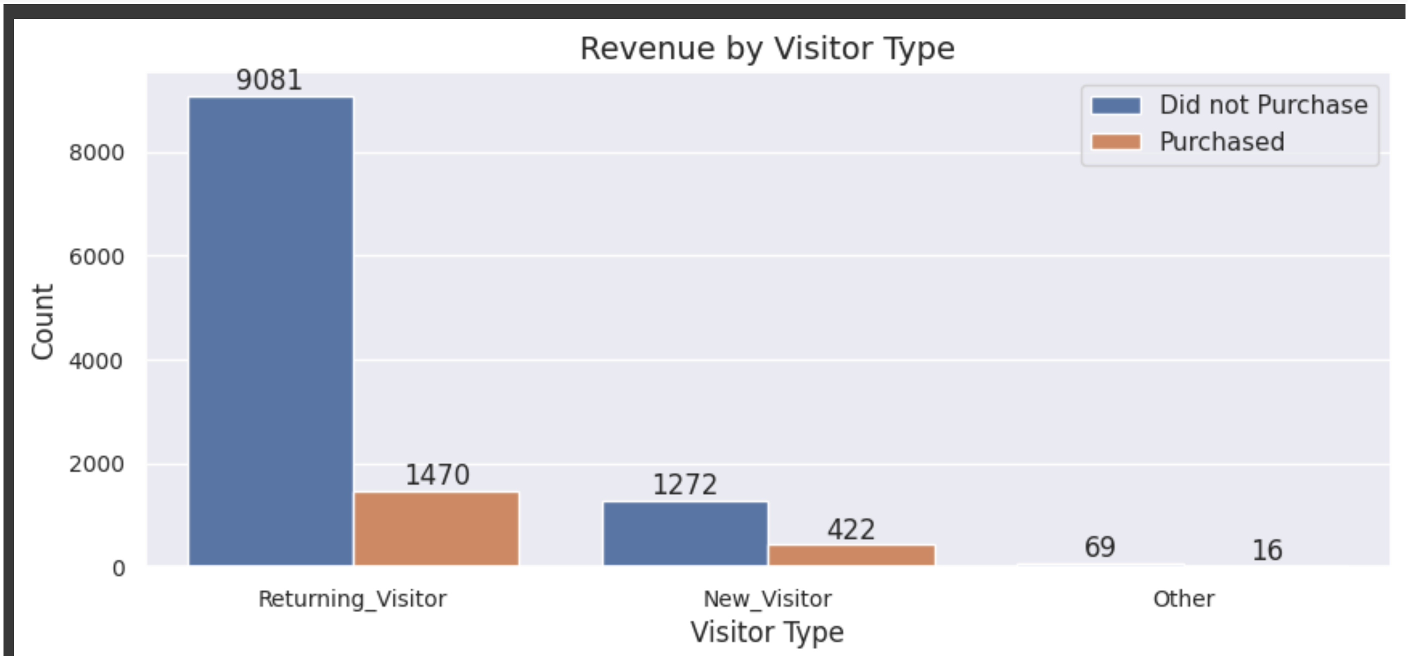
```python
fig, ax = plt.subplots(figsize=(4, 4))

x=revenue_ratio

cmap = plt.get_cmap('Greys')
colors = list(cmap(np.linspace(0.45, len(x))))

colors[1]='dodgerblue'
labels = ['Did not Purchase','Purchased']

patches, texts, pcts = ax.pie(
    x, labels=labels, autopct='%.2f%%',
    wedgeprops={'linewidth': 3.0, 'edgecolor': 'white'},
    textprops={'size': 'medium'},
    startangle=90,
    colors=colors,
    explode=(0, 0.1))

for i, patch in enumerate(patches):
   texts[i].set_color(patch.get_facecolor())
plt.setp(pcts, color='white')
plt.setp(texts, fontweight=300)
ax.set_title('Distribution of Target Feature (Revenue)', fontsize=12)
plt.tight_layout()
```



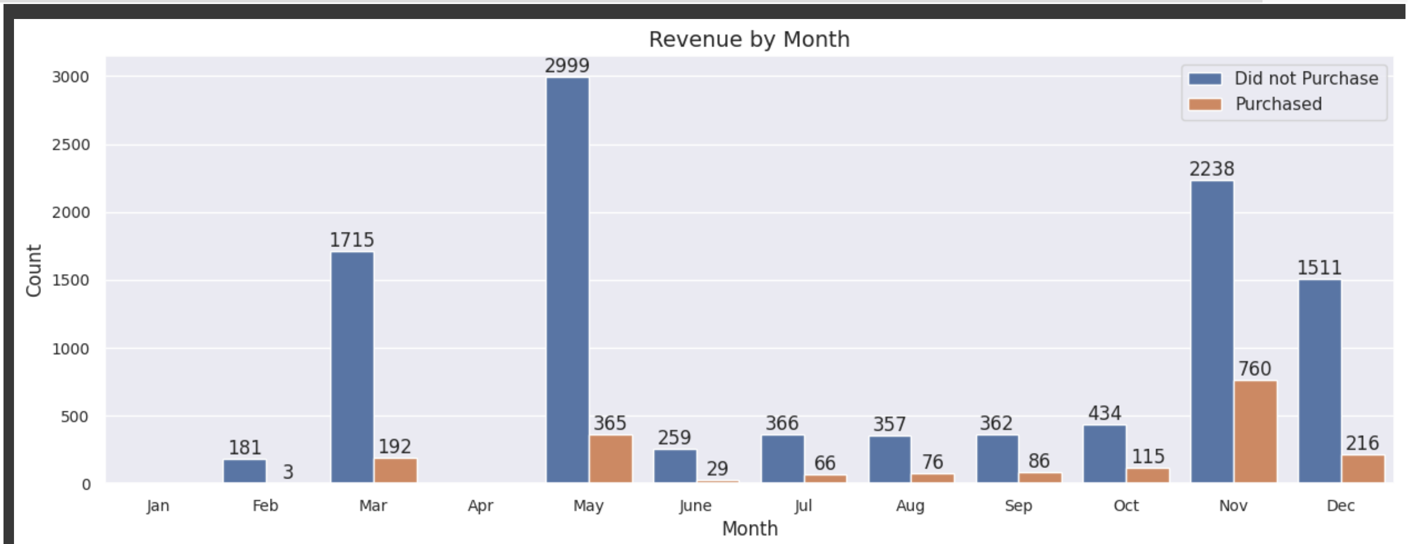
Imbalance in the output variable, where 84.53% didnot purchase.

Revenue by visitor type"

```
plt.figure(figsize=(10,4))
plt.title("Revenue by Visitor Type", fontsize=14)
ax = sns.countplot(x='VisitorType', data=data, hue = 'Revenue')
ax.legend(labels=['Did not Purchase','Purchased'])
for i in ax.containers:
    ax.bar_label(i)
plt.xlabel("Visitor Type", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



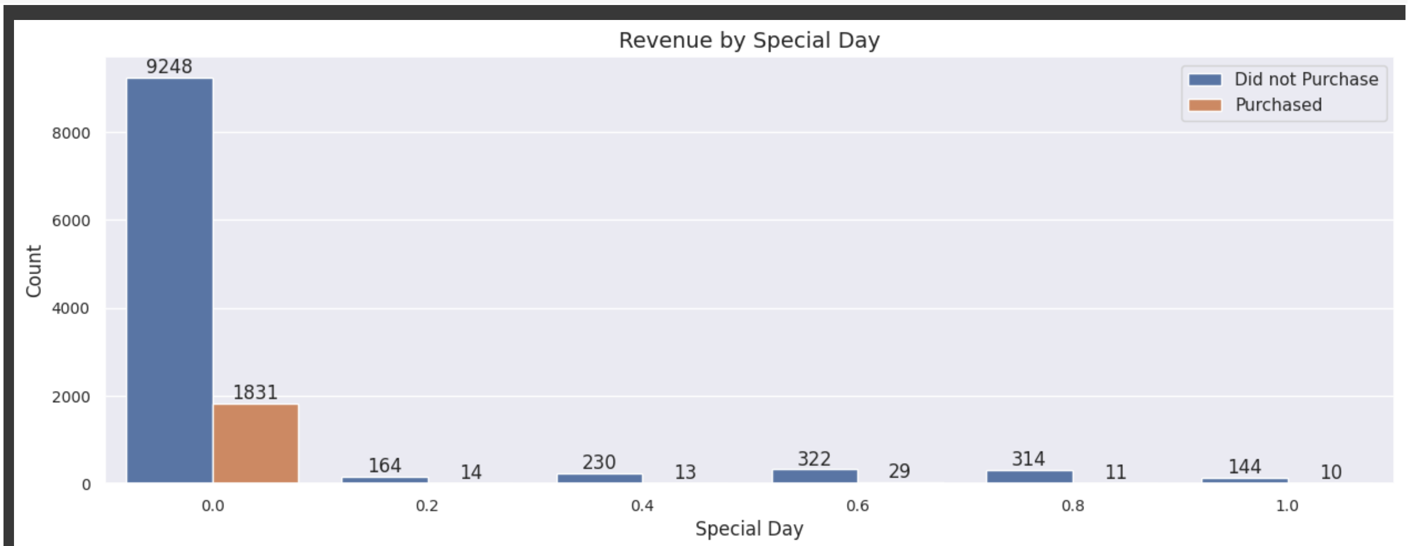
```
plt.figure(figsize=(15,5))
plt.title("Revenue by Month", fontsize=14)

orderlist = ['Jan','Feb','Mar','Apr','May','June','Jul','Aug','Sep','Oct','Nov

ax = sns.countplot(x='Month', data=data, hue = 'Revenue', order=orderlist)
ax.legend(labels=['Did not Purchase','Purchased'])
for i in ax.containers:
    ax.bar_label(i)
plt.xlabel("Month", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



- No data found for January and April
- Lot of the transaction happned at the end of the year

Revenue by Special Day:

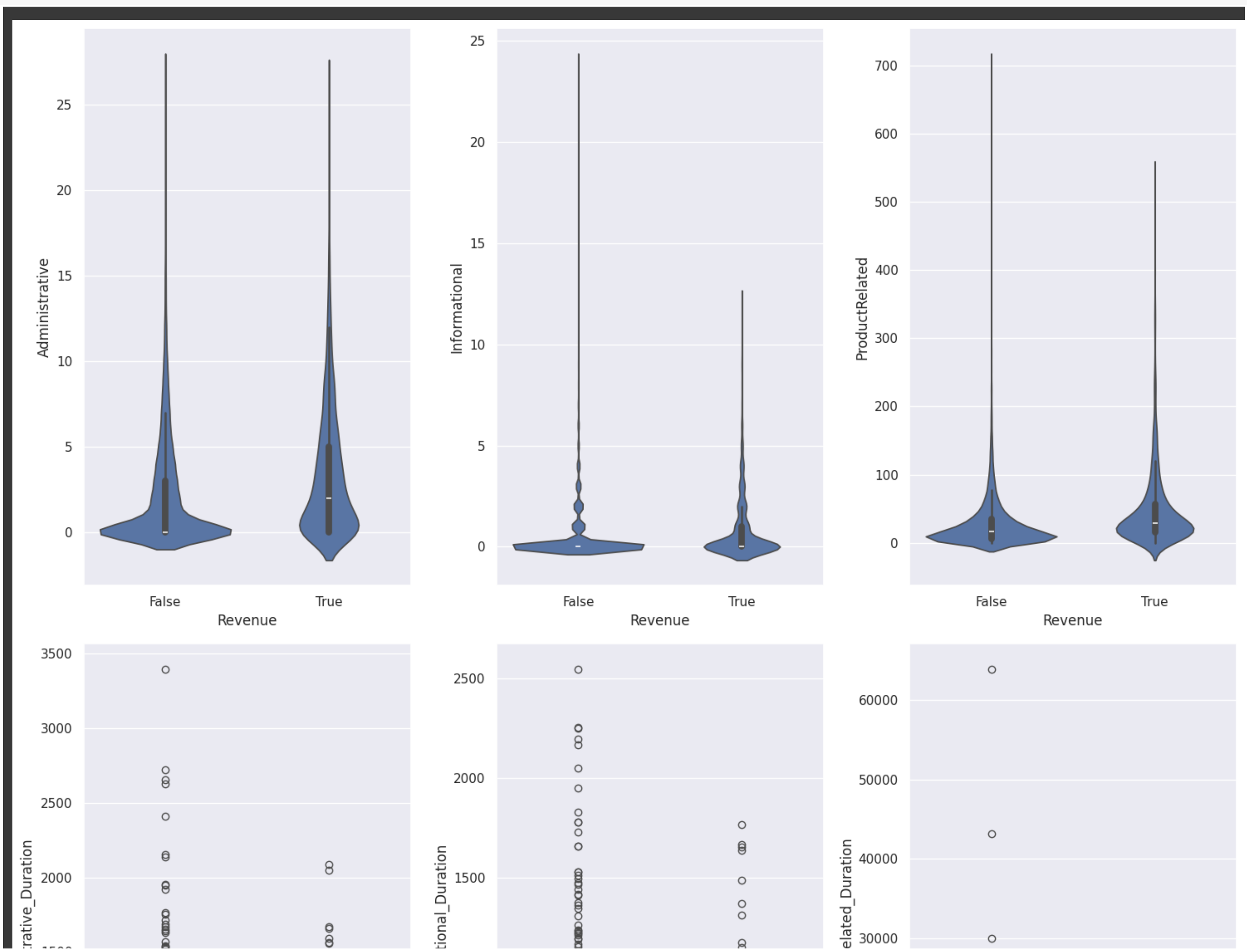
```
plt.figure(figsize=(15,5))
plt.title("Revenue by Special Day", fontsize=14)

ax = sns.countplot(x='SpecialDay', data=data, hue = 'Revenue')
ax.legend(labels=['Did not Purchase','Purchased'])
for i in ax.containers:
    ax.bar_label(i)
plt.xlabel("Special Day", fontsize=12)
plt.ylabel("Count", fontsize=12)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```

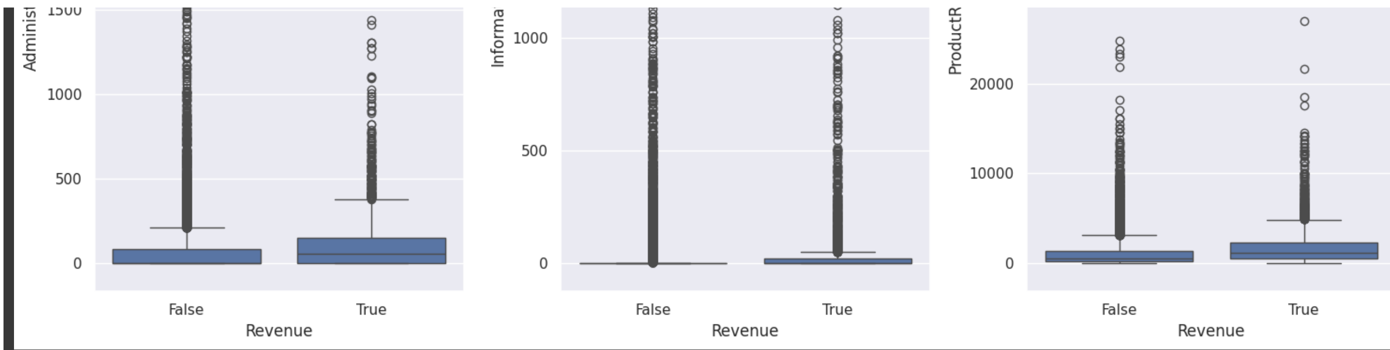


- ***There were significantly more website visitors and revenue generated (Completed purchases) on Special Day 0.0 in comparison to the other special days.***

Revenue by page type/ Duration

```
 fig = plt.figure(figsize=(15, 15))

ax1 = fig.add_subplot(2, 3, 1)
ax2 = fig.add_subplot(2, 3, 2)
ax3 = fig.add_subplot(2, 3, 3)
ax4 = fig.add_subplot(2, 3, 4)
ax5 = fig.add_subplot(2, 3, 5)
ax6 = fig.add_subplot(2, 3, 6)

sns.violinplot(data=data, x = 'Revenue', y = 'Administrative', ax=ax1)
sns.violinplot(data=data, x = 'Revenue', y = 'Informational', ax=ax2)
sns.violinplot(data=data, x = 'Revenue', y = 'ProductRelated', ax=ax3)
sns.boxplot(data=data, x = 'Revenue', y = 'Administrative_Duration', ax=ax4)
sns.boxplot(data=data, x = 'Revenue', y = 'Informational_Duration', ax=ax5)
sns.boxplot(data=data, x = 'Revenue', y = 'ProductRelated_Duration', ax=ax6)

plt.tight_layout()
plt.show()
```




Administ
1500
1000
500
0
False
True
Revenue
Informa
1000
500
0
False
True
Revenue
ProductR
20000
10000
0
False
True
Revenue

- ***Visitors tend to visit less pages, and spend less time, if they are not going to make a purchase.***
- ***The number of product related pages visited and time spent on them is more than that for account related or informational pages.***

Data Pre-Processing

In this section we will make our data ready for model training. This will include:

- Encode Categorical features using dummy encoding
- Encode Boolean variables using label encoder
- Split Data into train and test set
- Scale train set using the standard scaler

```
# Encode categorical features (Month, Visitor Type) using dummy encoding

categorical = ['Month', 'VisitorType']

encoded_features = pd.get_dummies(data[categorical])
encoded_features.head(3)
```

	Month_Aug	Month_Dec	Month_Feb	Month_Jul	Month_June	Month_Mar	Month_May
0	0	0	1	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0

Next steps: Generate code with encoded_features View recommended plots

```
#Concactenante encoded features to dataset and drop non-encoded variables

data = pd.concat([data, encoded_features], axis=1)

data.drop(categorical, axis=1, inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 29 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Administrative                 12330 non-null  int64  
 1   Administrative_Duration        12330 non-null  float64
 2   Informational                  12330 non-null  int64  
 3   Informational_Duration         12330 non-null  float64
 4   ProductRelated                 12330 non-null  int64  
 5   ProductRelated_Duration        12330 non-null  float64
 6   BounceRates                    12330 non-null  float64
 7   ExitRates                      12330 non-null  float64
 8   PageValues                     12330 non-null  float64
 9   SpecialDay                     12330 non-null  float64
 10  OperatingSystems               12330 non-null  int64  
 11  Browser                        12330 non-null  int64  
 12  Region                         12330 non-null  int64  
 13  TrafficType                    12330 non-null  int64  
 14  Weekend                        12330 non-null  bool   
 15  Revenue                        12330 non-null  bool   
 16  Month_Aug                      12330 non-null  uint8  
 17  Month_Dec                      12330 non-null  uint8  
 18  Month_Feb                      12330 non-null  uint8  
 19  Month_Jul                      12330 non-null  uint8  
 20  Month_June                     12330 non-null  uint8  
 21  Month_Mar                      12330 non-null  uint8  
 22  Month_May                      12330 non-null  uint8  
 23  Month_Nov                      12330 non-null  uint8  
 24  Month_Oct                      12330 non-null  uint8  
 25  Month_Sep                      12330 non-null  uint8  
 26  VisitorType_New_Visitor        12330 non-null  uint8  
 27  VisitorType_Other              12330 non-null  uint8  
 28  VisitorType_Returning_Visitor  12330 non-null  uint8  
dtypes: bool(2), float64(7), int64(7), uint8(13)
memory usage: 1.5 MB
```

```
# Encode Boolean variables using label Encoder

le = LabelEncoder()

data['Revenue'] = le.fit_transform(data['Revenue'])
data['Weekend'] = le.fit_transform(data['Weekend'])

print(data.Revenue.value_counts())
print(data.Weekend.value_counts())
```

```
0    10422
1     1908
Name: Revenue, dtype: int64
0    9462
1    2868
Name: Weekend, dtype: int64
```

Select Target and Features

```
y = data['Revenue']
X = data.drop('Revenue', axis=1)
```

```
#Split Dataset into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
```

```
#Scale train set using Standard scaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_train_scaled = pd.DataFrame(X_train_scaled, index=X_train.index, columns = X_

X_test_scaled = scaler.transform(X_test)
X_test_scaled = pd.DataFrame(X_test_scaled, index=X_test.index, columns = X_te
```

```
X_train_scaled.head()
```

	Administrative	Administrative_Duration	Informational	Informational_Du
3688	-0.398768	-0.410011	-0.393761	-0
8981	-0.698312	-0.454118	-0.393761	-0
5794	-0.698312	-0.454118	-0.393761	-0
11051	-0.698312	-0.454118	-0.393761	-0
6356	-0.698312	-0.454118	-0.393761	-0

5 rows × 28 columns

Modelling

- Train and evaluate models. Predictive models that will be used are Logistic Regression, KNeighbors Classifier, SVM, Decision Tree and Random Forest Classifier.
- The Scaled Dataset would be used for :- Logistic Regression, KNN and SVM.
- The Unscaled Dataset would be used for :- Decision Tree and Random Forest Classifier.
- Hyperparameter Tuning for the model with the best performance to try to improve its performance further.
- Inspect Feature importance (Top 10 features)
- Evaluate with Cross Validation.

```
# Initialize models

LR = LogisticRegression()
KN = KNeighborsClassifier()
SV = SVC()
DC = DecisionTreeClassifier()
RF = RandomForestClassifier()
```

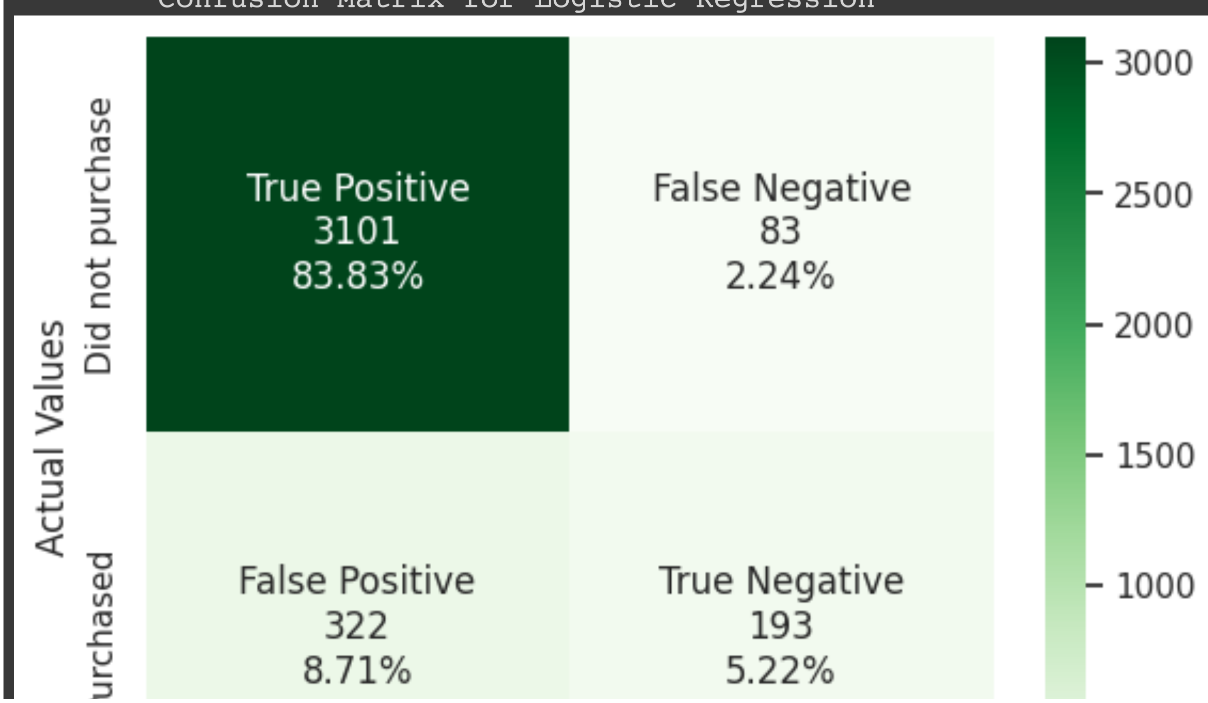
```
def c_matrix_plot(y_test,prediction):

    c_matrix = confusion_matrix(y_test,prediction)
    group_names = ['True Positive', 'False Negative', 'False Positive', 'True 
    group_counts = ["{0:0.0f}".format(value) for value in
                    c_matrix.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         c_matrix.flatten()/np.sum(c_matrix)]
    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
              zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)

    ax = sns.heatmap(c_matrix, annot=labels, fmt='', cmap='Greens')

   # ax.set_title(f'Confusion Matix for {prediction.__class__.__name__}');
    ax.set_xlabel('\nPredicted Values')
    ax.set_ylabel('Actual Values ');

    ax.xaxis.set_ticklabels(['Did not purchase', 'Purchased'])
    ax.yaxis.set_ticklabels(['Did not purchase', 'Purchased'])

    plt.show()
```

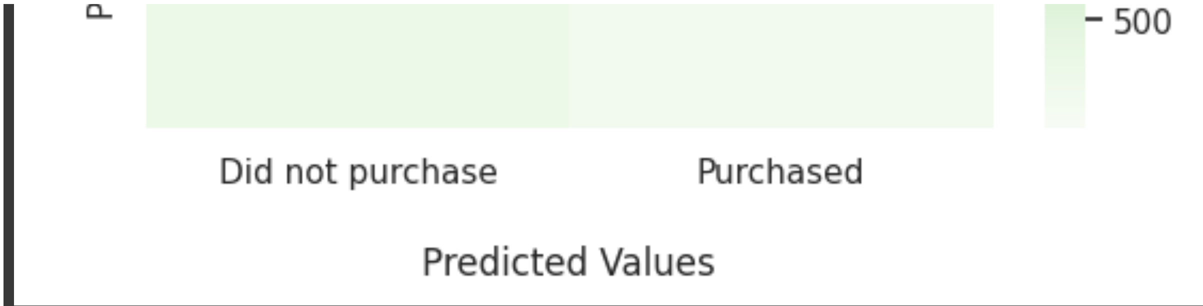
```
from sklearn.preprocessing import StandardScaler

# Instantiate the scaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the testing data
X_test_scaled = scaler.transform(X_test)

```

```
# For Logistic Regression, KNN and SVM, we will use the scaled dataset

LR = LogisticRegression()
LR = LR.fit(X_train_scaled, y_train)
LR_preds = LR.predict(X_test_scaled)
print('\nFor Logistic Regression, Accuracy score is ', accuracy_score(y_test,L
print(classification_report(y_test, LR_preds))
print(confusion_matrix(y_test, LR_preds))
print('\n\tConfusion Matrix for Logistic Regression')
c_matrix_plot(y_test, LR_preds)

```

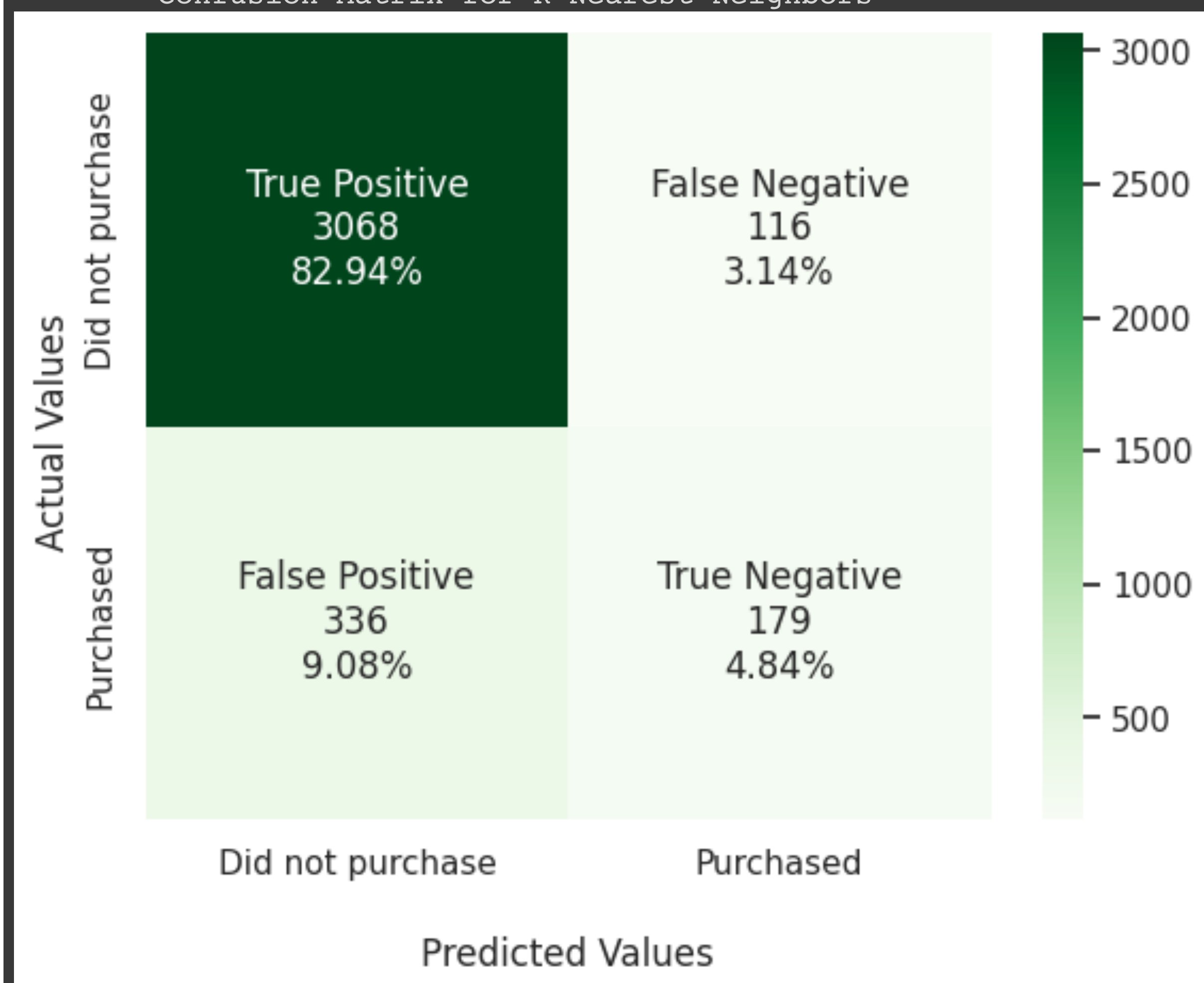
```
KN = KNeighborsClassifier()
KN = KN.fit(X_train_scaled, y_train)
KN_preds = KN.predict(X_test_scaled)
print('\nFor KNeighbors, Accuracy score is ', accuracy_score(y_test,KN_preds))
print(classification_report(y_test, KN_preds))
print(confusion_matrix(y_test, KN_preds))
print('\n\tConfusion Matrix for K-Nearest Neighbors')
c_matrix_plot(y_test, KN_preds)

SV = SVC()
SV = SV.fit(X_train_scaled, y_train)
SV_preds = SV.predict(X_test_scaled)
print('\nFor SVM, Accuracy score is ', accuracy_score(y_test,SV_preds))
print(classification_report(y_test, SV_preds))
print(confusion_matrix(y_test, SV_preds))
print('\n\tConfusion Matrix for SVM')
c_matrix_plot(y_test, SV_preds)
```

```
For Logistic Regression, Accuracy score is  0.8905109489051095
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      3184
           1       0.70      0.37      0.49       515

    accuracy                           0.89      3699
   macro avg       0.80      0.67      0.71      3699
weighted avg       0.88      0.89      0.88      3699

[[3101   83]
 [ 322  193]]

        Confusion Matrix for Logistic Regression
```



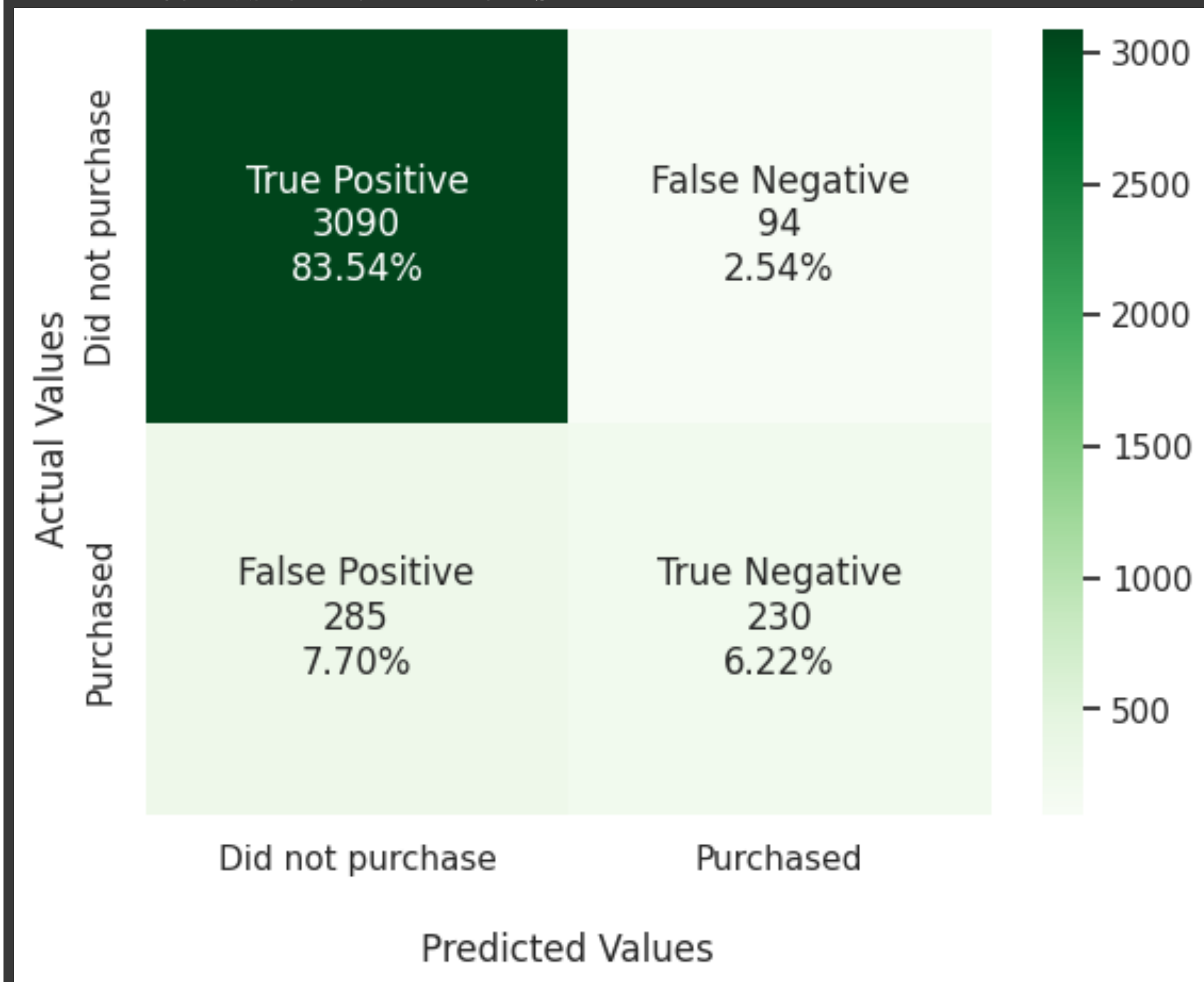



```
For KNeighbors, Accuracy score is  0.8778048121113815
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      3184
           1       0.61      0.35      0.44       515

    accuracy                           0.88      3699
   macro avg       0.75      0.66      0.69      3699
weighted avg       0.86      0.88      0.86      3699

[[3068  116]
 [ 336  179]]

        Confusion Matrix for K-Nearest Neighbors
```




```
For SVM, Accuracy score is  0.8975398756420654
              precision    recall  f1-score   support
```

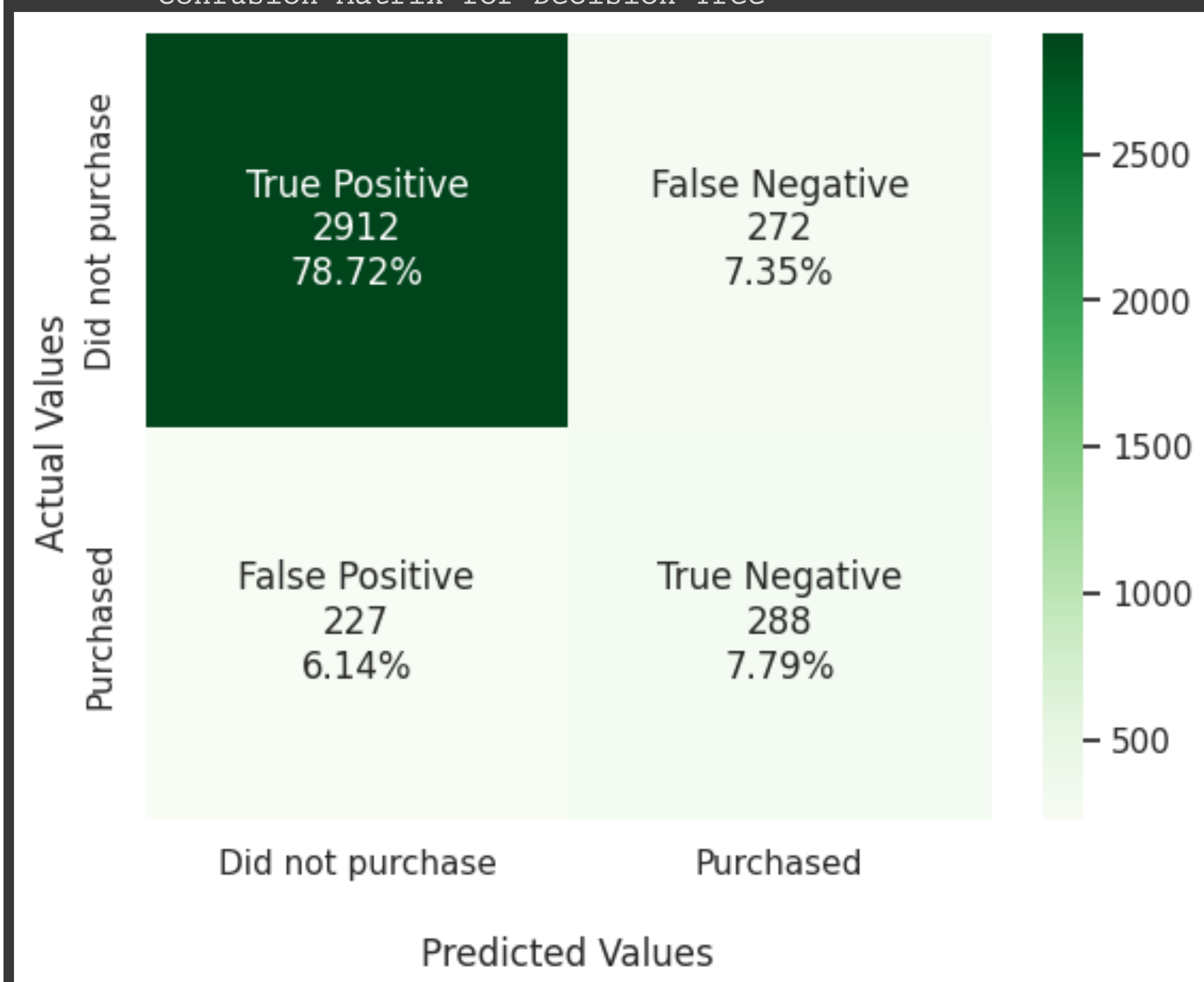
```
           0       0.92      0.97      0.94      3184
           1       0.71      0.45      0.55       515

    accuracy                           0.90      3699
   macro avg       0.81      0.71      0.75      3699
weighted avg       0.89      0.90      0.89      3699

[[3090   94]
 [ 285  230]]

        Confusion Matrix for SVM
```




```
# For Decision Tree Classifier and Random Forest, we will use the unscaled dat

DC = DecisionTreeClassifier()
DC = DC.fit(X_train, y_train)
DC_preds = DC.predict(X_test)
print('\nFor Decision Tree Classifier, Accuracy score is ', accuracy_score(y_t
print(classification_report(y_test,DC_preds))
print(confusion_matrix(y_test,DC_preds))
print('\n\tConfusion Matrix for Decision Tree')
c_matrix_plot(y_test,DC_preds)

```

```
RF = RandomForestClassifier()
RF = RF.fit(X_train, y_train)
RF_preds = RF.predict(X_test)
print('\nFor Random Forest Classifier, Accuracy score is ', accuracy_score(y_t
print(classification_report(y_test,RF_preds))
print(confusion_matrix(y_test,RF_preds))
print('\n\tConfusion Matrix for Random Forest')
c_matrix_plot(y_test,RF_preds)
```

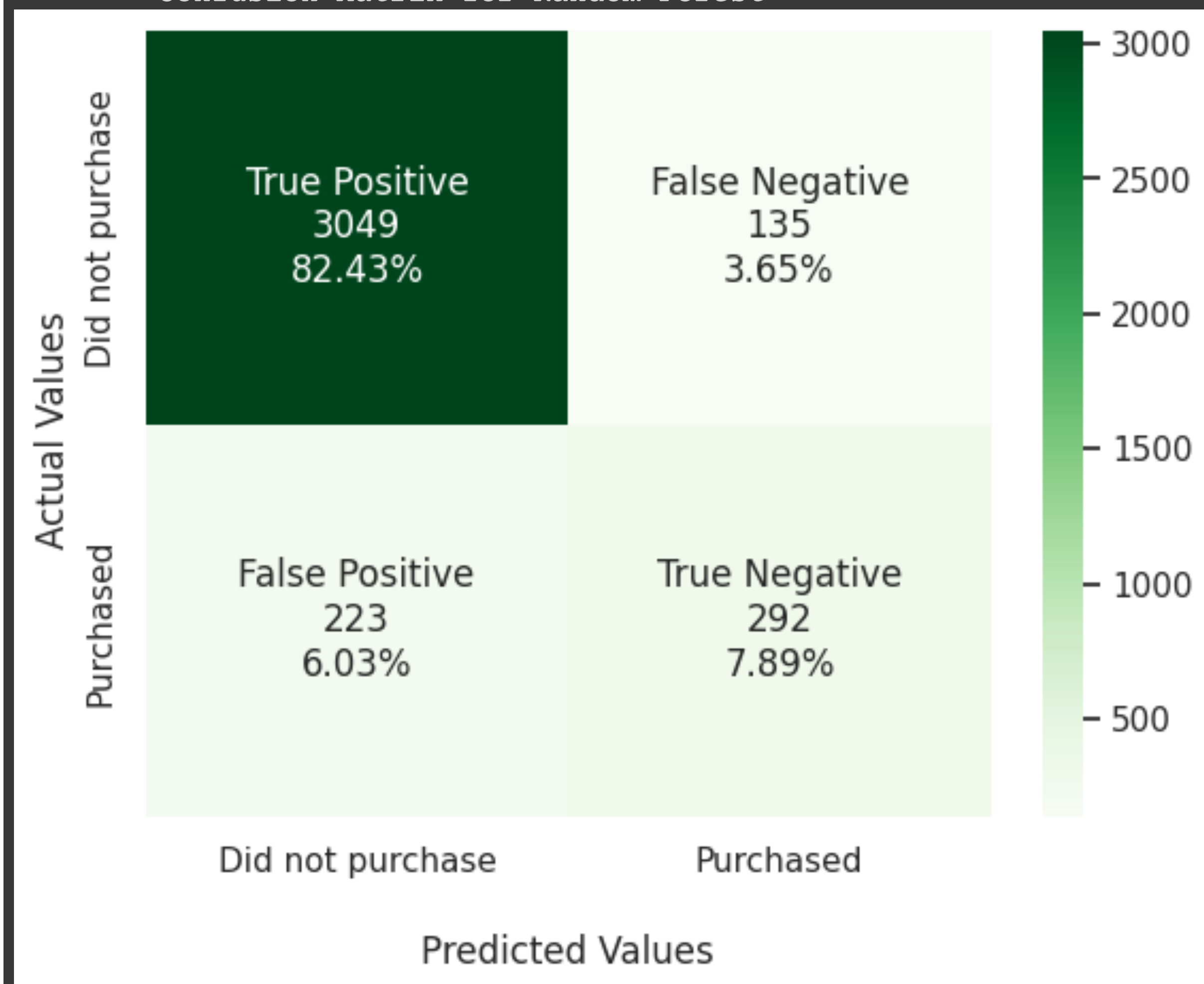
```
For Decision Tree Classifier, Accuracy score is  0.8650986753176534
              precision    recall  f1-score   support

           0       0.93      0.91      0.92      3184
           1       0.51      0.56      0.54       515

    accuracy                           0.87      3699
   macro avg       0.72      0.74      0.73      3699
weighted avg       0.87      0.87      0.87      3699

[[2912  272]
 [ 227  288]]

        Confusion Matrix for Decision Tree
```




```
For Random Forest Classifier, Accuracy score is  0.9032170856988375
```

```
              precision    recall  f1-score   support

           0       0.93      0.96      0.94      3184
           1       0.68      0.57      0.62       515

    accuracy                           0.90      3699
   macro avg       0.81      0.76      0.78      3699
weighted avg       0.90      0.90      0.90      3699

[[3049  135]
 [ 223  292]]
```

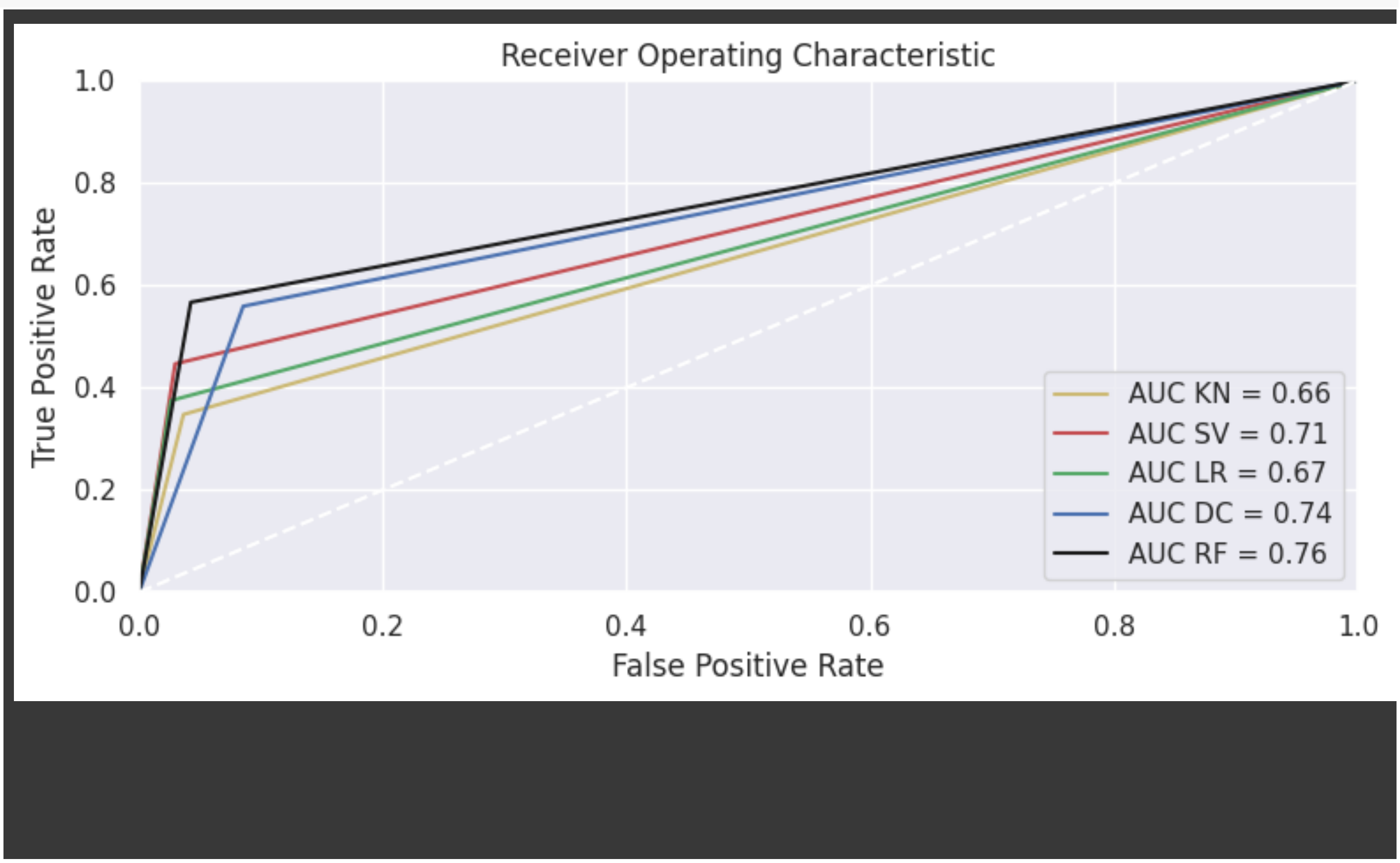
Confusion Matrix for Random Forest



ROC Carve

```
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

fpr_kn, tpr_kn, threshold_kn = metrics.roc_curve(y_test, KN_preds)
roc_auc_kn = metrics.auc(fpr_kn, tpr_kn)
fpr_sv, tpr_sv, threshold_sv = metrics.roc_curve(y_test, SV_preds)
roc_auc_sv = metrics.auc(fpr_sv, tpr_sv)
```

```
fpr_lr, tpr_lr, threshold_lr = metrics.roc_curve(y_test, LR_preds)
roc_auc_lr = metrics.auc(fpr_lr, tpr_lr)
fpr_dc, tpr_dc, threshold_dc = metrics.roc_curve(y_test, DC_preds)
roc_auc_dc = metrics.auc(fpr_dc, tpr_dc)
fpr_rf, tpr_rf, threshold_rf = metrics.roc_curve(y_test, RF_preds)
roc_auc_rf = metrics.auc(fpr_rf, tpr_rf)

fig = plt.figure(figsize=(8, 4))
plt.title('Receiver Operating Characteristic')
plt.plot(fpr_kn, tpr_kn, 'y', label = 'AUC KN = %0.2f' % roc_auc_kn)
plt.plot(fpr_sv, tpr_sv, 'r', label = 'AUC SV = %0.2f' % roc_auc_sv)
plt.plot(fpr_lr, tpr_lr, 'g', label = 'AUC LR = %0.2f' % roc_auc_lr)
plt.plot(fpr_dc, tpr_dc, 'b', label = 'AUC DC = %0.2f' % roc_auc_dc)
plt.plot(fpr_rf, tpr_rf, 'k', label = 'AUC RF = %0.2f' % roc_auc_rf)

plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'w--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.tight_layout()
plt.show()
```



Hyper-Parameter Tuning - Random Forest

```
from pprint import pprint

print('Parameters currently in use:\n')
pprint(RF.get_params())

```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

```python
from sklearn.model_selection import RandomizedSearchCV

n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

```python
# Use the random grid to search for best hyperparameters

# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = RF,
                               param_distributions = random_grid,
                               n_iter = 100,
                               cv = 3,
                               verbose=2,
                               random_state=42,
                               n_jobs = -1)

```

```
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

▸ **RandomizedSearchCV**

▸ **estimator: RandomForestClassifier**

▸ RandomForestClassifier

```
rf_random.best_params_
```

```
{'n_estimators': 1000,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 10,
 'bootstrap': True}
```

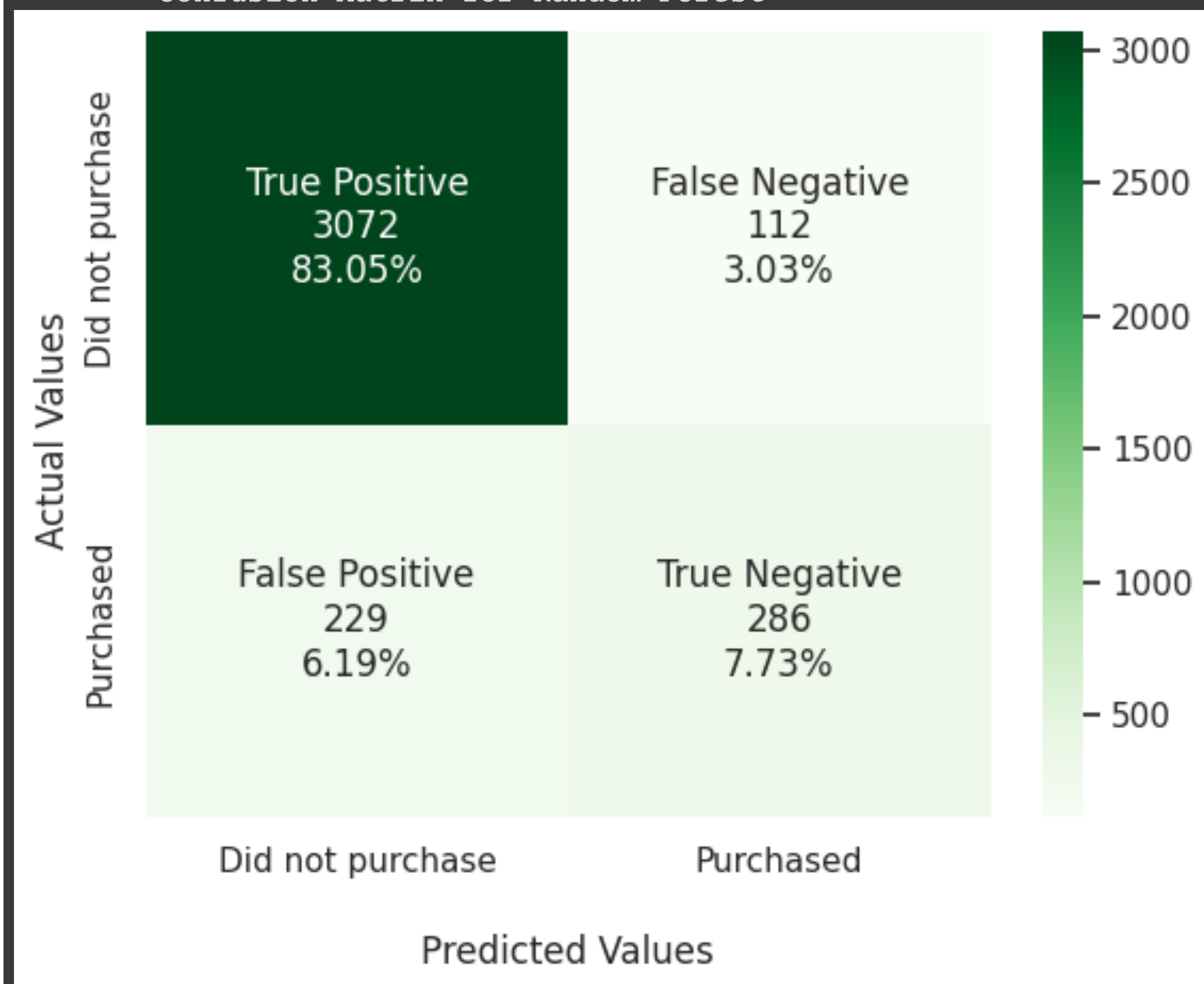
```
rf_random.best_estimator_
```

▾ RandomForestClassifier

```
RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split=5,
                       n_estimators=1000)
```

```
rf_random = RandomForestClassifier(n_estimators=1600,
                                   max_depth=10,
                                   min_samples_split=2,
                                   min_samples_leaf=4,
                                   max_features='sqrt',
                                   bootstrap=True)
rf_random.fit(X_train,y_train)
rf_random_preds = rf_random.predict(X_test)

print('\nFor Random Forest Classifier, Accuracy score is ', accuracy_score(y_tes
print(classification_report(y_test, rf_random_preds))
print(confusion_matrix(y_test, rf_random_preds))
print('\n\tConfusion Matrix for Random Forest')
c_matrix_plot(y_test, rf_random_preds)
```

```
For Random Forest Classifier, Accuracy score is  0.9078129224114626
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      3184
           1       0.72      0.56      0.63       515

    accuracy                           0.91      3699
   macro avg       0.82      0.76      0.79      3699
weighted avg       0.90      0.91      0.90      3699

[[3072  112]
 [ 229  286]]

        Confusion Matrix for Random Forest
```



Inspect Feature Importance

```
#get feature importances
RF_importances = pd.DataFrame(data = rf_random.feature_importances_,index = X_

#plot top 10 feature importances, sorted
RF_importances[:10].sort_values(by='Importance').plot.barh()

plt.title('Feature importances for Random Forest')
plt.show()
```

```
#get these top 10 importances
RF_importances[:10].sort_values(by='Importance').index.values
```

Evaluating with Cross Validation

```
# evaluate your models using k-fold cross-validation
from numpy import mean
from numpy import std
from sklearn.model_selection import KFold, cross_val_score, cross_val_predict

# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=1, shuffle=True)

```

```
#create function to train a model with cross validations and evaluate accuracy
def trainer_with_cv(model,X,y):
    '''Cross validation function. Expects a model,'''
    # evaluate model
    accuracy_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv,
    print('Accuracy:')
    print(accuracy_scores)
    print(model.__class__.__name__,'Mean Accuracy: %.3f' % (mean(accuracy_scor

    precision_scores = cross_val_score(model, X, y, scoring='precision', cv=cv
    print('\nPrecision:')
    print(precision_scores)
    print(model.__class__.__name__,'Mean Precision: %.3f' % (mean(precision_sc

    recall_scores = cross_val_score(model, X, y, scoring='recall', cv=cv, n_jo
    print('\nRecall:')
    print(recall_scores)
    print(model.__class__.__name__,'Mean Recall: %.3f' % (mean(recall_scores))
```

```
1 trainer_with_cv(rf_random,X,y)
```

```
Accuracy:
[0.9107867  0.91159773 0.89213301 0.90348743 0.90024331 0.918897
 0.90673155 0.90429846 0.90105434 0.89699919]
RandomForestClassifier Mean Accuracy: 0.905

Precision:
[0.74647887 0.74193548 0.73913043 0.77536232 0.75555556 0.8030303
 0.7443609  0.78231293 0.7826087  0.78873239]
RandomForestClassifier Mean Precision: 0.766

Recall:
[0.58888889 0.5380117  0.52763819 0.55440415 0.52331606 0.57923497
 0.55307263 0.57425743 0.54       0.53846154]
RandomForestClassifier Mean Recall: 0.552
```

The cross validation result shows that the Random Forest Classifier is able to generalize to new data

Conclusion

- In this project, we trained models that can classify visitors to a store's website, and predict if they are likely to make a purchase on the website or not.
- Five (5) learning classifiers (Logistic Regression, KNN, SVM, Decision Tree and Random Forest) were tested.
- The Random Forest Classifier had the best performance with an accuracy of 90% and F-1 Score of 62%.
- The Page Values Feature was found to be the most important feature in determining the purchase intention of a website visitor. Other important features include the Exit rate, Bounce rate, type of pages visited as well as the duration spent on the pages.
- The cross validation result shows that the Random Forest Classifier is able to generalize to new data