Online Shoppers Behavior Prediction Report

EXECUTIVE SUMMARY

Online shopping is a very popular activity that a lot of people enjoy and find convenient. The E-commerce market has been growing for many years. The popular ecommerce sites such as Amazon, Walmart, Ebay, etc have been providing this convenient service to people and earning profit. At present time, even small businesses have started building their own ecommerce websites and providing customized online shopping facilities to their customers. Therefore, the purpose of this project is to evaluate the intentions and behavior of online shoppers, and find their relation with revenue.

The businessperson or the e-commerce site owner's primary goal is to increase their revenue as much as possible. This can be achieved by making sure that most of the buyers visiting the online store end up buying a product and contribute to the company's revenue. Therefore, for this project, we chose a dataset published by UCI Machine Learning Repository called Online Shoppers Intention. This dataset contains 12,330 observations which are 12,330 online sessions of the unique website visitors over the period of a year. It consists of 18 variables, 10 numerical and 8 categorical, including the target variable i.e. revenue. Revenue is a boolean attribute where, true indicates that the buyer made a purchase at the end of their session, and false means that no purchase was made and thus, no contribution towards the revenue for that particular session.

Out of 12,330 sessions, only 1,908 sessions ended up with purchase which is 15.47% of the total observations in the dataset. Thus, our dataset is an imbalanced dataset. Some of the attributes in our dataset are as follow:

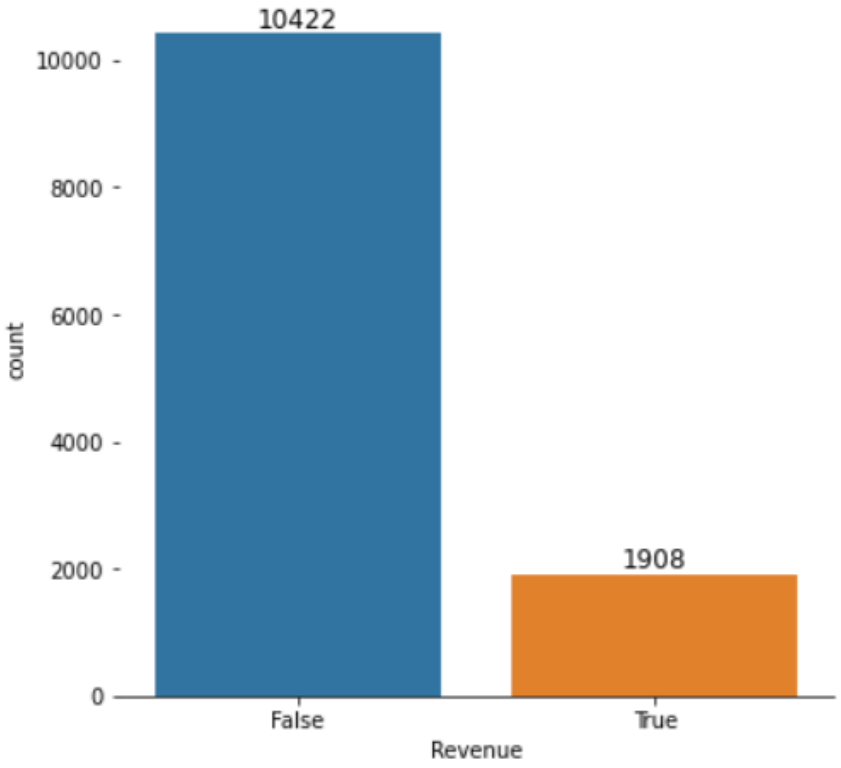
- Administrative : The type of administrative page in the website.
- Administrative_Duration : The amount of time spent by the visitor in administrative related pages.
- Product Related : The type of product related pages in the website.
- ProductRelated_Duration : The amount of time spent by the visitor in the product related pages.
- Page Values : The value of a particular page in the site depending on its contribution to the revenue, i.e. buyer making a purchase.
- SpecialDay : Number representing the closeness of special days like Mother's Day, Christmas, etc from the session day.
- Month : Month of the year when session was recorded
- Operating Systems : Operating System used by the visitor.
- Bounce Rate : The percentage of visitors who enter the website through that page and exit without triggering any additional tasks.
- Visitor Type : New visitor, returning visitor, or other

➢ Weekend : Boolean value representing if the day of the session recorded was a weekend or weekday.

To understand the dataset and its relationship with the target variable, Revenue, we performed Exploratory Data Analysis.

1. Exploratory Data Analysis

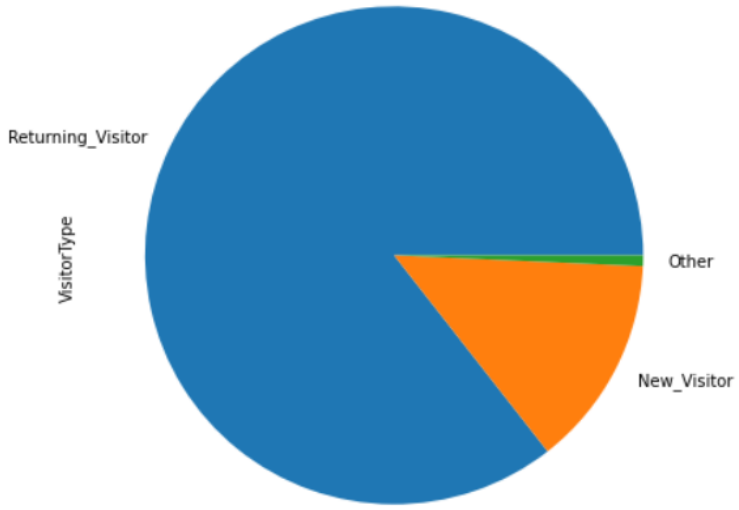
Target Variable: Revenue



Observation:

➔ Only 15.47% of the sessions ended up making a purchase.
➔ 84.5% of the sessions in the dataset did not have Any purchase and thus no contributions toward revenue.

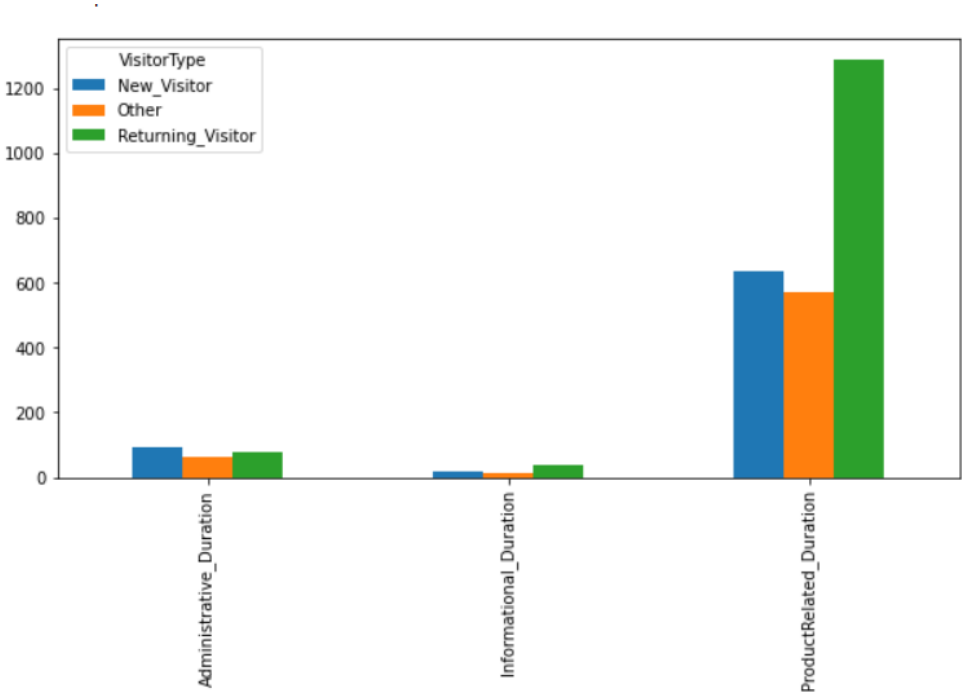
Types of Visitors



Observation:

- ➔ 85% of customers return back to site, which is a good indication, which could mean that the customers are satisfied with what they are getting.
- ➔ 13.74% of customers are new customers.

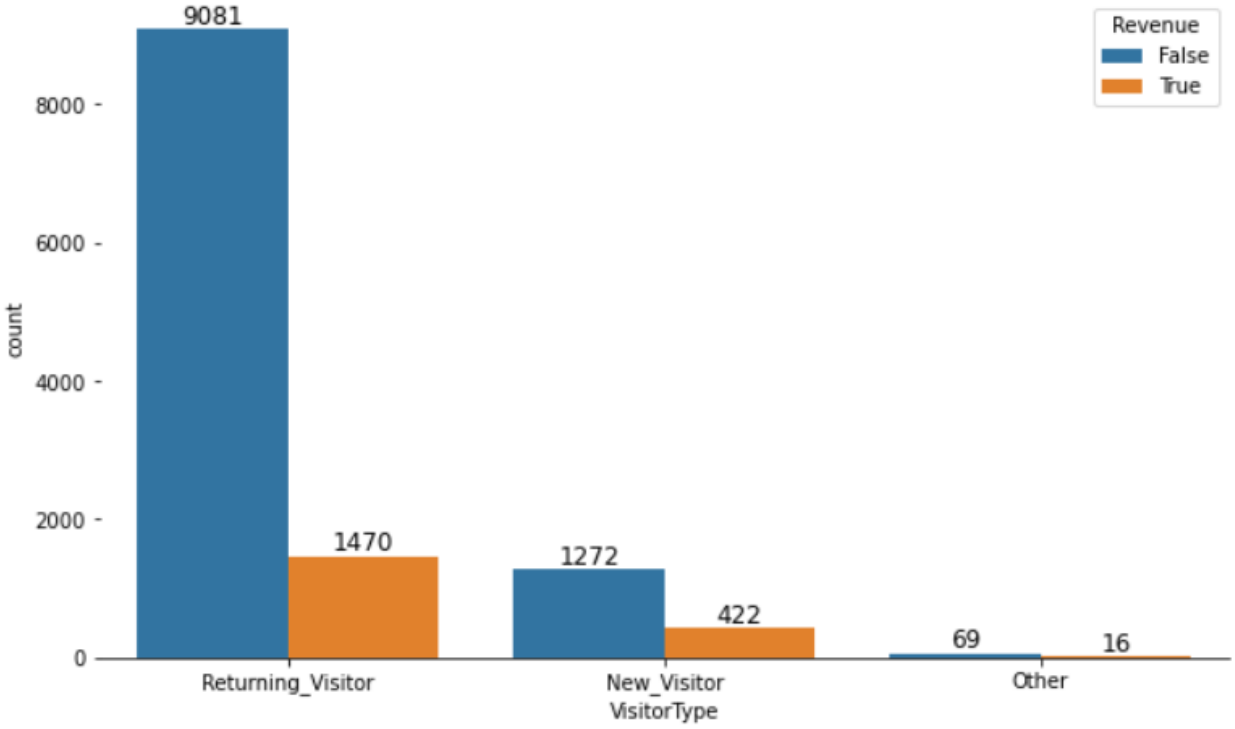
Page Durations



Observation:

- ➔ Most of the customers spend a maximum amount of time in product related pages, therefore, this type of pages should be the main focus for the business owners.
- ➔ New visitors spend more time in Administrative pages rather than returning visitors as they would be involved in performing actions like creating profiles, signing up on the website, etc. Therefore, making sure that this process goes smoothly for the new customers could ensure that the visitors remain longer in the website and explore more pages.

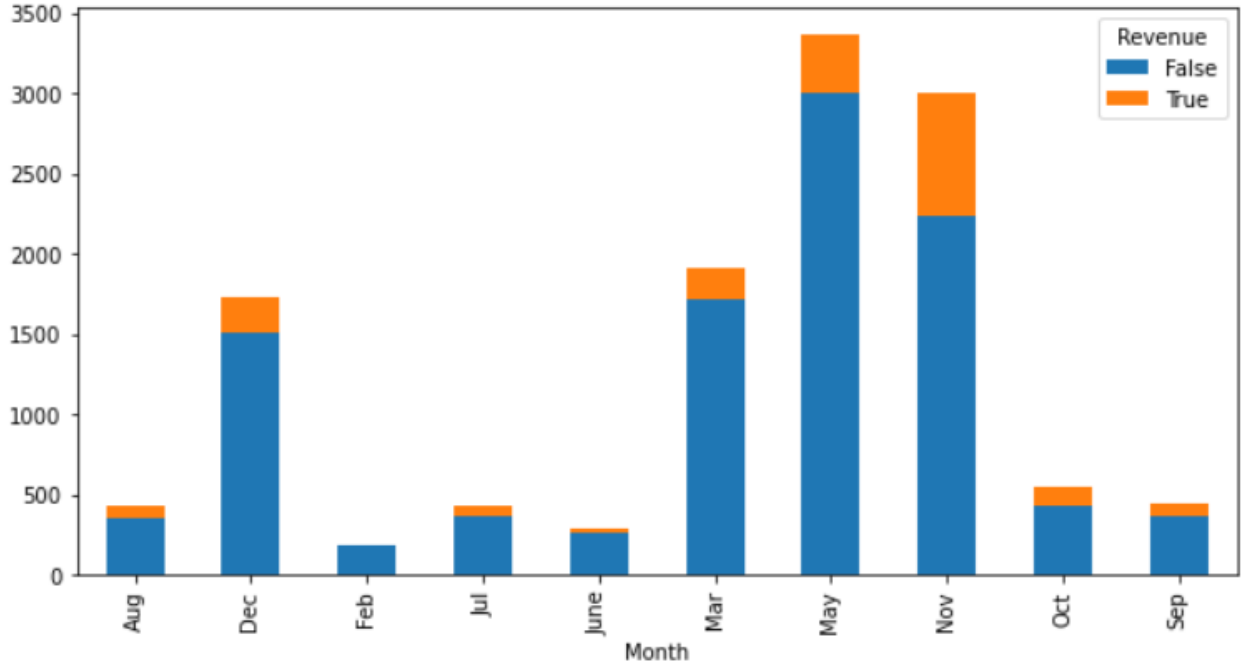
Visitors Contributions to Revenue



Observation:

- ➔ Returning visitors contribute more towards revenue.
- ➔ New visitors' buying percentage could be improved by providing them incentives to try the products on the website.
- ➔ The website owners could provide new and returning visitors with some perks so that they end up buying products at the end.

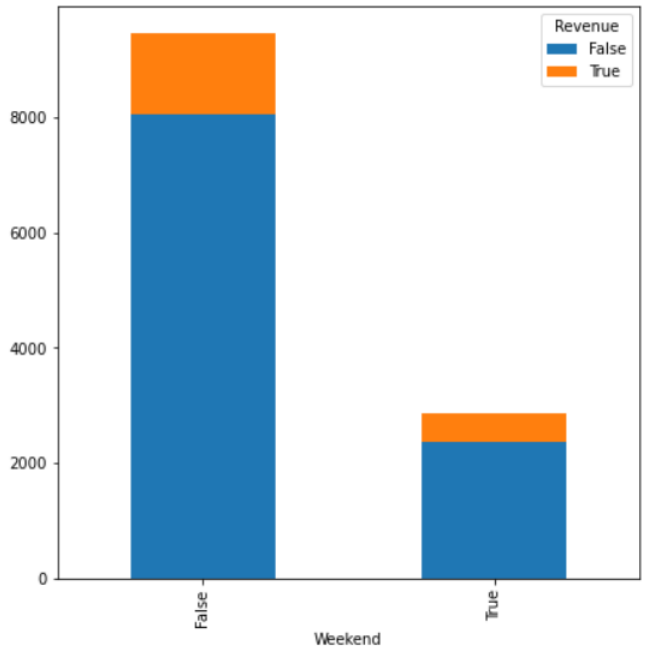
Monthly Revenue



Observation:

- ➔ May has the highest number of visitors on the website.
- ➔ November has the highest revenue.

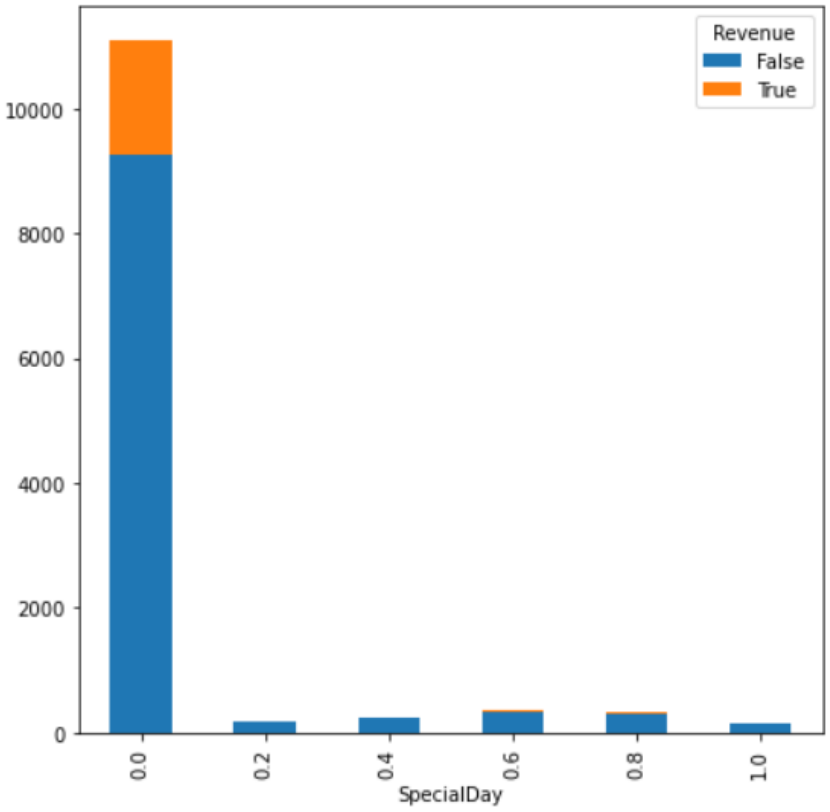
Weekend VS Weekday Revenue



Observation:

➔ There are more site visitors on weekdays for this site rather than weekends.
➔ Weekdays bring in more revenue compared to weekends.

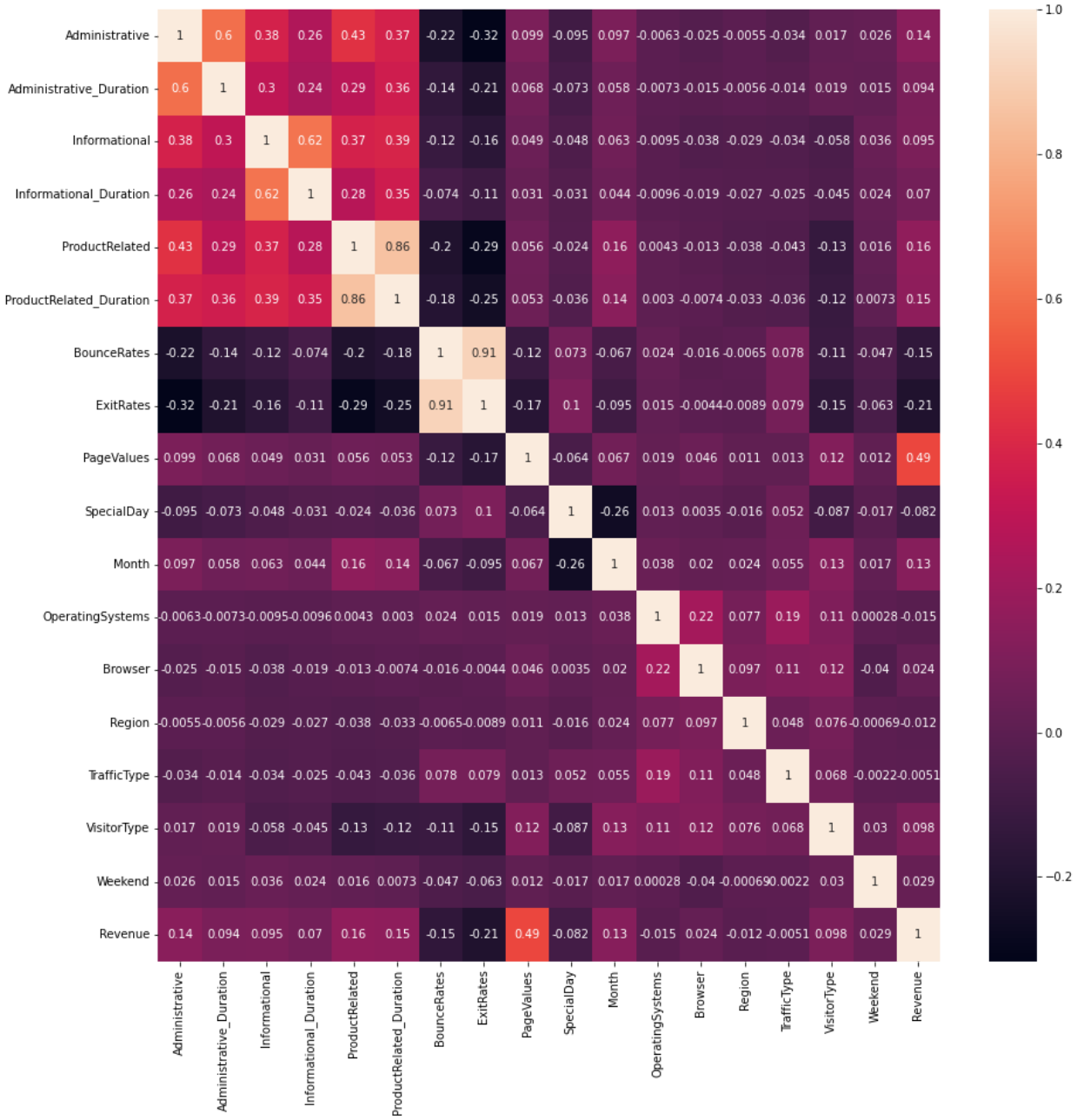
Special Days



Observation:

➔ Special days do not have any effect on revenue.

Correlation Matrix for all the attributes in the dataset



Now that we are familiar with our dataset and the different attributes associated with it, we can prepare the data for training our classification model. Then, we will train our prediction model and apply different Machine Learning algorithms to examine and interpret our data.

1. Logistic Regression

In general, logistic regression is a simple machine learning algorithm that allows us to solve classification problems, where we are trying to predict discrete categories. For example, predicting whether an email is spam or not and so on. We can't use a normal

linear regression model on a binary group, as it's totally unrealistic to represent the probability below 0% in linear regression.. Since the problems we are trying to solve here are associated with a binary group, i.e, whether a customer's intention eventually generates revenue or not, logistic regression fits perfectly here.

Training our data

```
df
features = ['Administrative', 'Administrative_Duration',....., 'TrafficType', 'VisitorType']
X = df[features] #Training data
y = OSI_df['Revenue'] #Prediction label

# Split the data into x_train and y_train data sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=42)

# Scale the numeric values
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

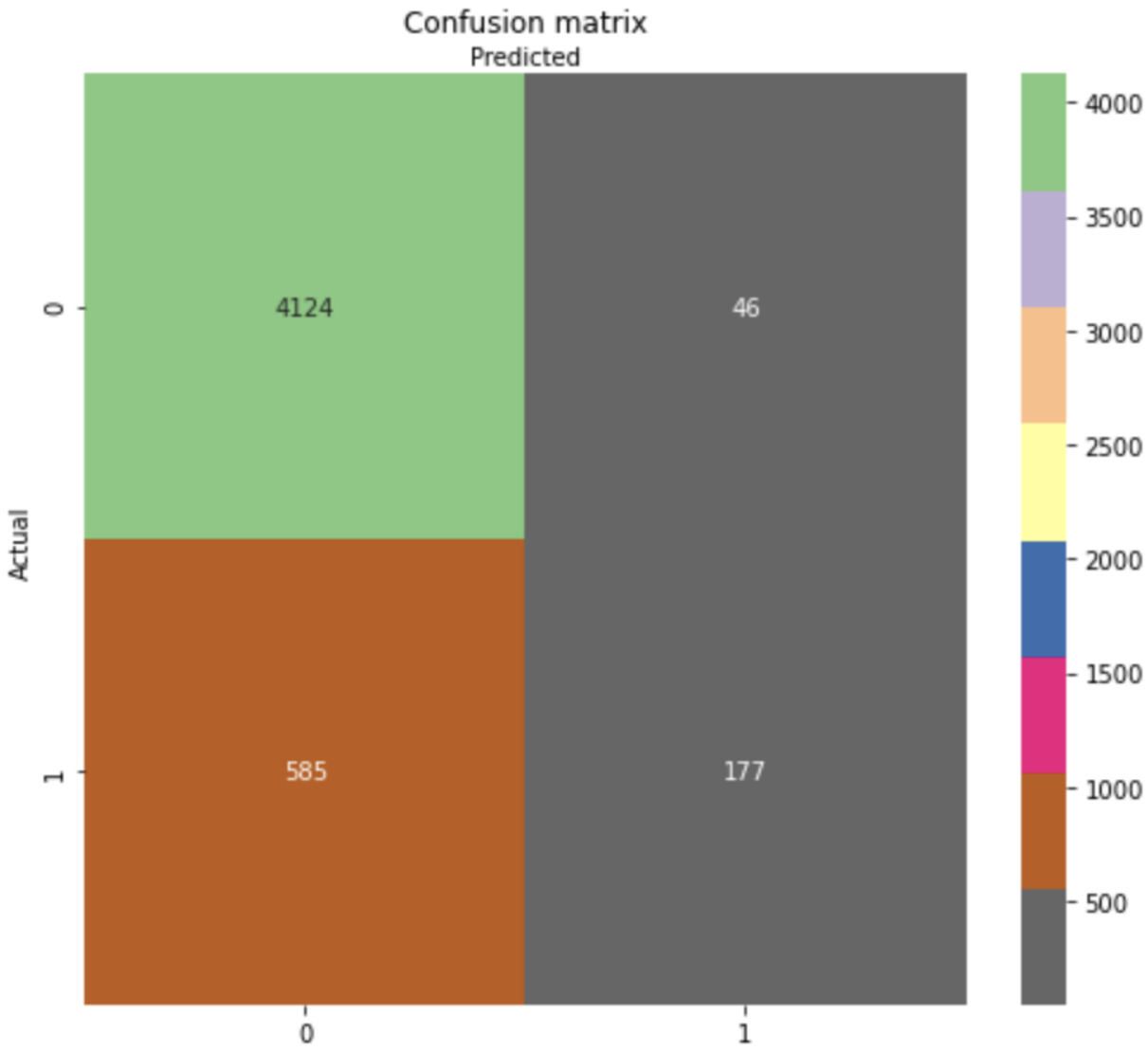
As we can see above, we have first selected only relevant features from our original dataset, and then we have split the data into train and test sets with a ratio of 70% and random state of 42.. While X_train and X_test contain the features, y_train and y_test contain the prediction labels associated with them. Since, the value we obtain from the logistic regression model should be between 0 and 1, we have used the MinMaxScaler to scale numeric features into that range.

Training Prediction Model

```
logmodel = LogisticRegression(solver='newton-cg')
score = logmodel.fit(X_train, y_train)
```

As shown in the code snippet above, we then begin our training process using the fit method available in the logistic regression model. The training score recorded here determines the performance of our model in the dataset.

Confusion matrix



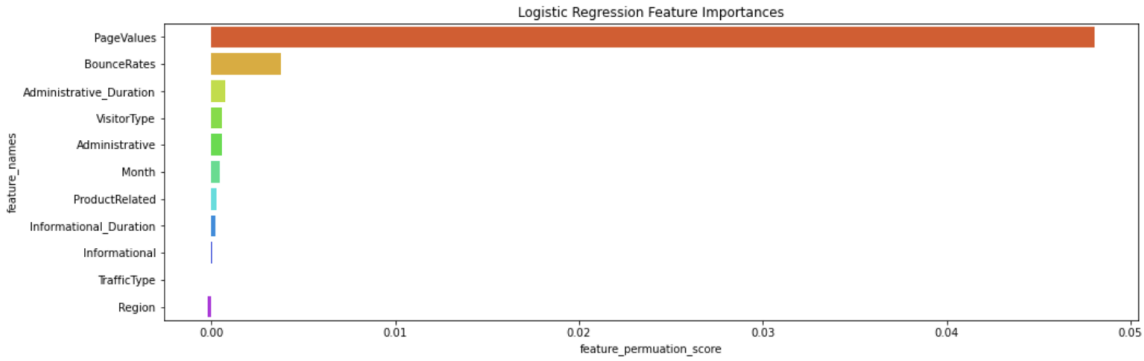
The figure above shows the confusion matrix created and obtained based on our logistic regression model. Note the value 0 represents that no revenue was generated and the value 1 represents that the revenue was generated. The upper left quadrant represents the true negative value of 4124. This means that similar to what our model predicted, 4124 online shopper sessions did not end up as a purchase. Likewise, the upper right quadrant represents the false negative value which is 46. Meaning in 46 different cases, our model predicted there would be a purchase made when in reality, there wasn't. Similarly, the lower left quadrant represents the false negative value of 585. This means that in 585 distinct cases, our model incorrectly predicted that there would be no purchase made when in reality, shoppers had been purchasing at least something. Lastly, the lower right quadrant represents the true positive value which is 177. In short, our model correctly identified 177 online shoppers who would actually make purchases.

Performance Metrics

```
Accuracy: 0.87
Precision: 0.79
Recall: 0.23
F1_score: 0.36
```

The accuracy on the test set shows that 87% of the online shopping sessions were correctly classified. However, accuracy might not be an appropriate metric to measure the performance of our model here. The precision on the test is 79% which is no of much use to us as precision is concerned with true positives and false positives, and it does not say much about the true negatives. Likewise, the recall is 23% and the F1_score is 36% here. Since our dataset is imbalanced, and most of our labels have "False" and only few of them are "True", accuracy and performance metrics can be biased here. If we want to deduce a business model, our main focus would be what contributes to a purchase rather than what doesn't. In conclusion, it is challenging based on our model to determine which performance matrix is important because the value of True Negative, and True positive or False negative and False positive can have different costs in performance metrics if the dataset is imbalanced.

Insights based on the Permutation Feature Importance



Permutation Feature Importance is a technique that measures the influence of features on the prediction of our model. Since the feature with high positive or high negative can have a substantial impact on predicting labels, it helps to find out which features are more relevant than others. Therefore, as we can see on the figure, the three features with the highest impact are PageValues, BounceRates and Administration_Duration. The higher the value of the page, the higher the chance that a customer makes a purchase. Likewise, the average bounce rate that the customer visits also contributes greatly to the purchase decision.

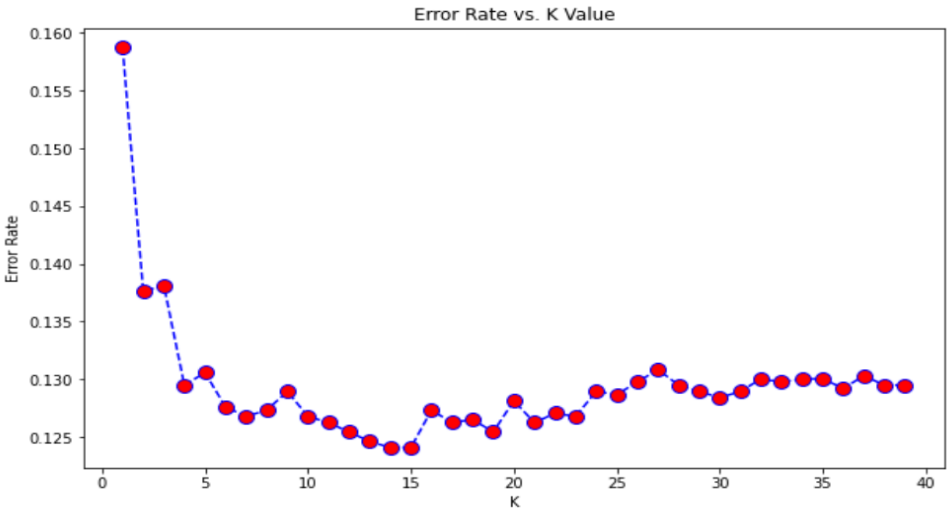
2. <u>K Nearest Neighbors</u>

K nearest neighbors is another classification algorithm that is used to determine how the data set can be classified. Initially for the data set, a training set and test set was created at the ratio of 70:30. This data set was used and a model was created in order to get the values for the KNN model. As the best K value(number of neighbors to be considered for the voting process) was not yet determined, the initial K value given to the model was 1. With this the model gave the following outcome:

```
Accuracy:  0.8413084617464179
Precision:  0.4947589098532495
Recall:  0.4054982817869416
F1:  0.44570349386213404
```

The accuracy achieved from the model was 0.84 which was high but it did not really matter since our dataset is imbalanced. The precision , recall and F1 value of the model were all less than 0.5 so this could be improved.

One of the best ways to improve the model was by finding the best value of the K to be used with the model. In order to do that, a graph of error rate is considered. For instance, error rate for range of values from 1 to 40 is calculated and plotted as shown below:



From the graph above we can see that values in the range 1-5 are not really suitable for the model as they have a high error rate. While considering the K value we needed to consider two things. Firstly, if the k value was too small it would lead to Low Bias, High Variance and Overfitting. On the other hand, if the K value was too big it would lead to

High Bias, Low Variance, Underfitting. So K value that was stable enough to not create overfitting and underfitting issues is chosen .i.e K=10.

After choosing the new value of K, a new model is created which produces the following results.

```
Accuracy:  0.87320897539875 64
Precision:  0.7383966244725738
Recall:  0.3006872852233677
F1:  0.42735042735042733
```

This newer model has better performance when compared in terms of precision compared to previous model, but recall and F1 score of this model decreases even more as the K value was increased. The outcomes of this final KNN model will be compared with other models created in the later part.

3. **Decision Trees**

In general, decision trees are fast, simple and easy to implement. They require little to no data preparation. One of the methods used in this analysis is Scikit-Learn's CART (Classification and Regression Tree) algorithm to train Decision Trees. The goal here was to create a model that predicts the value of the target variable (Revenue/no Revenue) by learning simple decision rules inferred from the data features.

The decision tree works as follows:

- It starts at the root node, where a particular feature is used to split the observations.
- On either side, we continue to split observations based on features to get the best split in class possible.

Given a node, there are following properties that are addressed:

- Node's sample: counts how many training instances it applies to
- Node's value: counts how many training instances of each class this node applies to
- Node's gini: measures the impurity of a node; given a node, it is either equally distributed across the classes, or it's biased towards a particular class. A lower gini score means the node is pure.

Building a Preliminary Tree

We're starting with a preliminary tree that has a maximum depth specified to 10. We test our model under this classifier to get a general idea of the feature interaction. It is no surprise that this is overfitting, and definitely not the best model. However, we can now start changing multiple parameters of the classifier to prune the tree and avoid some overfitting.

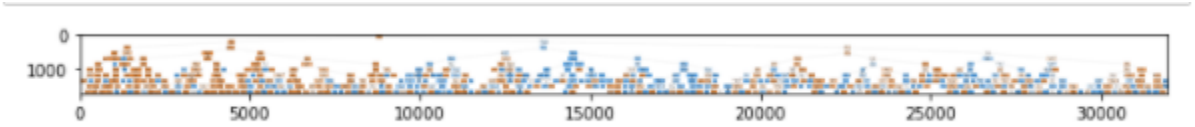


Fig 3.1: preliminary decision tree with max_depth = 10

It definitely looks like we could prune the tree further to increase performance metrics.

```
Accuracy:  0.8796972154636388
Precision:  0.6623931623931624
Recall:  0.5192629815745393
```

Fig 3.2: Metrics for tree with depth 10

Pruning the Tree

We try pruning the tree by manipulating multiple features such as max_depth. Experimenting with different values for max_depth: 10, 3, 4, we conclude on using the value 4.

Then, to further prune the tree, we manipulate the cost-complexity parameter. The CART algorithm uses a cost-complexity function that balances tree size (complexity) and misclassification error (cost) in order to choose tree size. ccp_alpha, or α is the cost-complexity parameter in the penalty term (tree size). When α = 0, the tree grows fully and overfit the data; when α =1, the tree is just a single node and hence underfits the data. The process is that starting from α = 0, we slowly increase it to 1 to fit trees at each stage. For each tree and its corresponding α, we use cross-validation to evaluate the tree, then compute the average errors across all different splits. We choose the α complexity parameter for which the CV error is minimal. And use the α and all the data we have to grow a new tree.

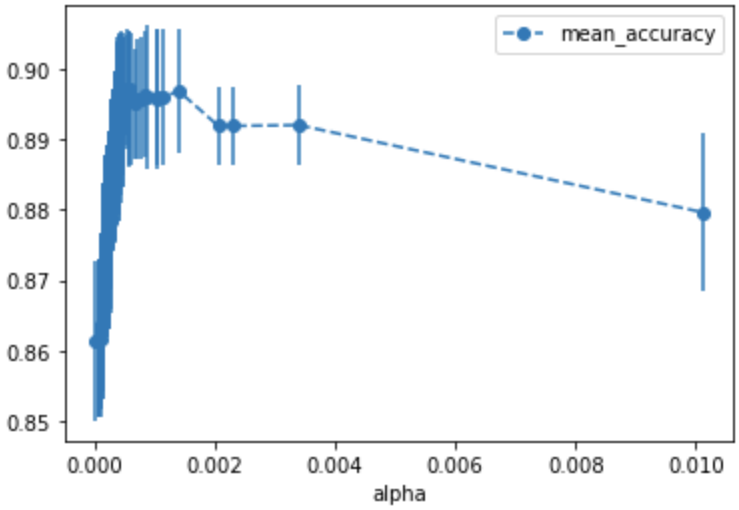


Fig 3.3: mean accuracy for 5 fold cross validation with multiple α values

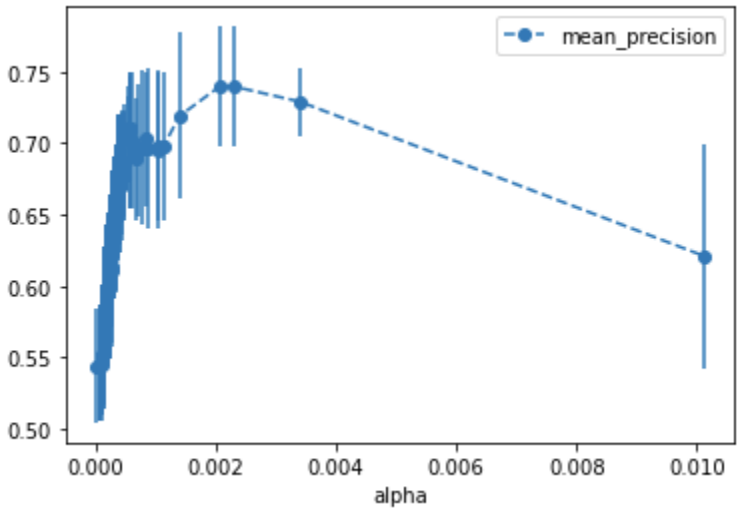


Fig 3.4: mean precision for 5 fold cross validation with multiple α values

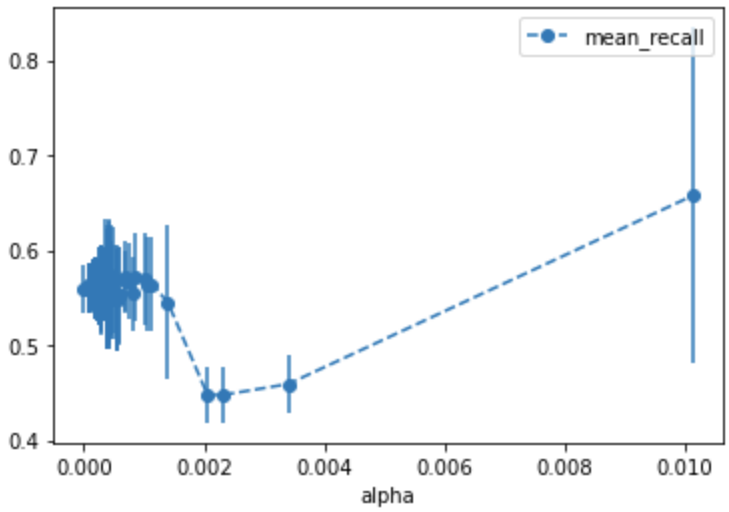


Fig 3.5: mean recall for 5 fold cross validation with multiple α values

As we see in the above figures, the majority of the ccp_alpha values are clustered in between 0.000 and 0.002.

Fig 3.3, shows that accuracy is highest for α = 0.0005.
Fig 3.4 shows that precision is highest at α = 0.002.
Fig 3.5 shows that recall is highest at α = 0.001.

Since it is an unbalanced dataset, accuracy values are not much reliable for prediction and efficiency. We take α = 0.00015 that maximizes recall and precision.

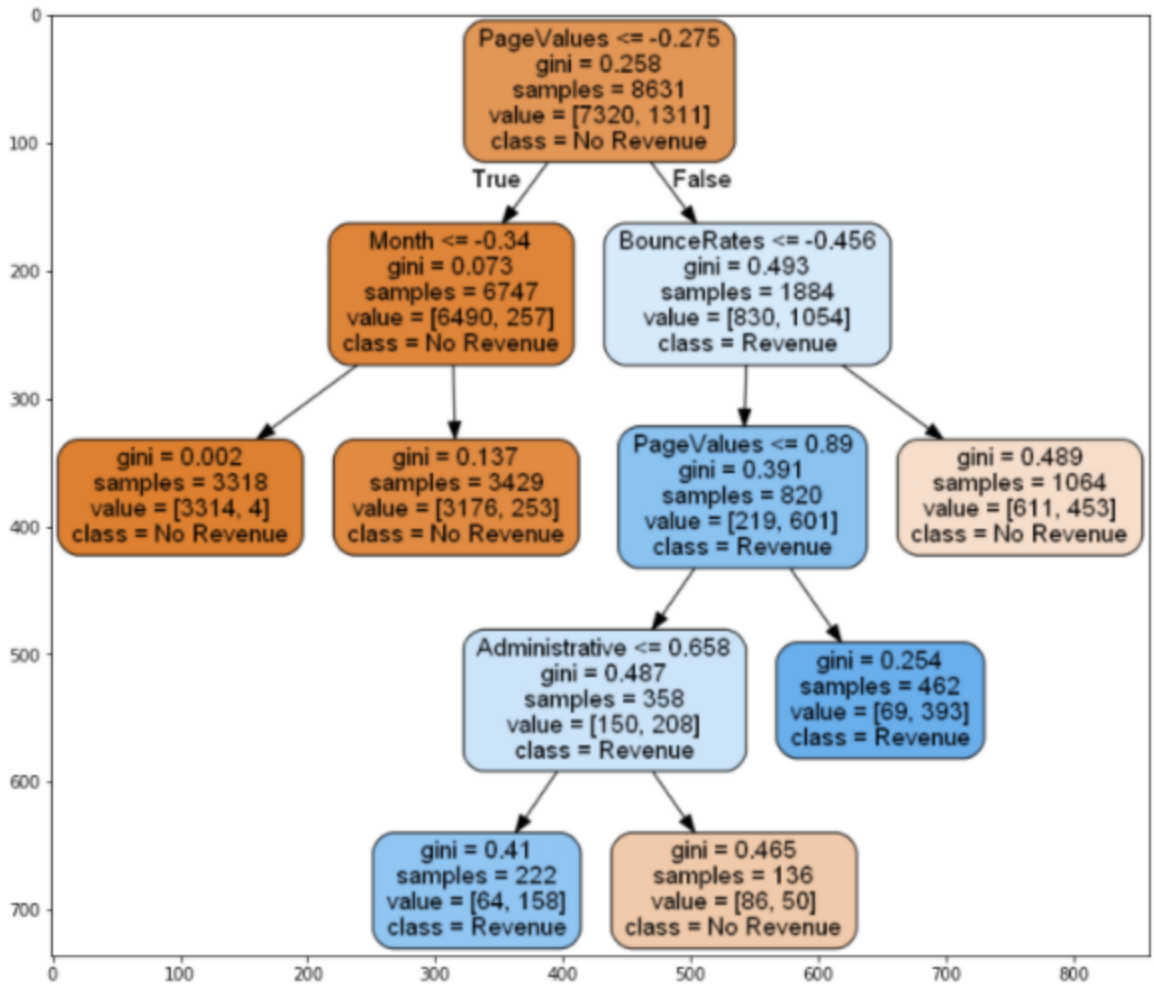


Fig 3.6: Final Decision tree

Interpreting the Tree

The orange nodes represent the class where no revenue is made at the end of a session.
The blue nodes represent the class where a revenue is finally made at the end of a session.
The darker the color of a node, the lower the Gini impurity; hence the better the split in classes.

Evaluating the Metrics

```
Accuracy:  0.8891592322249257
Precision:  0.8065573770491803
Recall:  0.4120603015075377
F1-Score:  0.5454545454545454
```

Fig 3.8: Metric Scores

The precision has dramatically increased from 52% to 80%.

Decision trees also help us get the importance of each feature based on the model to predict the class. Below is a feature importance table constructed using the final decision tree algorithm.

	Feature Name	Importance
8	PageValues	0.868991
6	BounceRates	0.091503
0	Administrative	0.020906
10	Month	0.018600
3	Informational_Duration	0.000000
4	ProductRelated	0.000000
5	ProductRelated_Duration	0.000000
2	Informational	0.000000
7	ExitRates	0.000000
1	Administrative_Duration	0.000000
9	SpecialDay	0.000000
11	OperatingSystems	0.000000
12	Browser	0.000000
13	Region	0.000000
14	TrafficType	0.000000
15	VisitorType	0.000000
16	Weekend	0.000000

Fig 3.9: Feature Importances derived from decision tree

The above table highlights how page value has a significant effect on whether or not revenue is generated at the end of a session. Some of the significant features are discussed below:

- Page Values: gives an idea of which page in the site contributed the most to a site's revenue
- Product Related: number of pages related to a product that the user was looking at/purchased
- Bounce Rates: the percentage of visitors who enter the website through the page and exit without triggering any additional tasks.

Decision Tree vs. Ensemble Learning (Random Forest)

	DecisionTreeClassifier	RandomForestClassifier
Accuracy	0.889159	0.902676
Precision	0.806557	0.758170
Recall	0.412060	0.582915
F1-Scores	0.545455	0.659091

Fig 3.10: Scores for final decision tree and random forest

Fig 3.10 shows the metrics from the tree and the random forest that employs an ensemble of 500 such tree classifiers.

CONCLUSION

Metrics	Logistic Regression	KNN Classifier	Decision Tree	Random Forest
Accuracy	0.87	0.87	0.89	0.89
Precision	0.79	0.73	0.80	0.76
Recall	0.23	0.30	0.41	0.58
F1-Score	0.36	0.43	0.55	0.66

Table 4.1: Comparison metrics

In general, linear regression and clustering techniques did not observe significant performance improvement, largely accounting due to the class imbalance present in the dataset (only ~18% of the observations in the sample have true values for revenue generation) due to which accuracy is not a performance indicator in this case.

As we know, high precision indicates a low number of false positives. In this case, the single Decision Tree classifier presents a higher precision score, which means this classifier is more exact than the other three.

Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.

If the intention was to improve on revenue drivers, recall rate suggests improvement. This is in line with the F1 score which takes both precision and recall into account.

Hence, these methods offered a first insight into the nature of the attributes and drivers towards the Revenue attribute. However better models were required to account for the class imbalance.

Decision Trees and Random Forest, on the other hand, have a higher recall score than the other classifiers. The intention was to capture those attributes that contribute the most to revenue growth. We can also note that the metrics for decision trees and random classifiers vary, with the ensemble having a greater score with a 17% difference. The F-1 Score for random forest is the highest, 0.66, which means that among the models discussed, it is more appropriate to be employed for revenue generation prediction.

Apart from the final prediction, along the way, our models have also shed light on some of the steps that the business could take to improve the online revenue generation by examining each variable in close light with respect to the target variable.

Here are some of the recommendations and revelations as a result of the analysis:

- The significant impact on PageValue as depicted by the linear regression and decision tree suggests that customers look at considerably different products and its recommendations. Hence, a significant improvement on recommendation engines and bundle packages would bring in more revenue.
- This also suggests that including more products exploiting the long tail effect in e-commerce will also bring in more revenue drivers. This would mean selling low volumes of hard to find products to many customers instead of only selling large volumes of common products found in the market.
- In our analysis, we also saw that product relation had a significant impact on revenue generation. This means, users are more likely to browse through similar products when given the option. So, employing ad mechanisms to display similar products is also another strategy that can be employed to influence purchasing behaviour.
- Lastly, the lower the bounce rate, the greater the chance of revenue being generated. Once the user lands a page, the business' strategy would be to indulge the user on the page, and browse through multiple products on the website rather than simply exiting the session. So, in order to retain the user's interest, an effective easy-to-use UI would go a long way. But also, employing search engine optimization, so that the best results show up for the user is also very effective on whether the user stays and browses or simply moves on to a different website.

REFERENCES

https://www.kaggle.com/annettecatherinepaul/online-shoppers-behavior-prediction#1.-Abstract-and-business-objective

APPENDIX

Note: Applicable feedback from the presentation and project proposals have been included in this report:

- Inclusion of an ensemble method to compare values against the single tree classifier
- PageRank doesn't seem applicable for this analysis since the page value simply is a numerical value that already indicates the importance of a particular page for revenue generation within a website

The code and results are attached in the pages below.

ML_projects-1

December 11, 2021

[290]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.figure as figure
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import pydotplus
import sklearn.metrics as metrics
```

0.0.1 Reading the CSV file

[177]:
```
OSI_df = pd.read_csv('online_shoppers_intention.csv')
```

[178]:
```
OSI_df.head()
```

[178]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	OperatingSystems
0	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0	0.0	Feb	1
1	0	0.0	0	0.0	2	64.000000	0.00	0.10	0.0	0.0	Feb	2
2	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0	0.0	Feb	4
3	0	0.0	0	0.0	2	2.666667	0.05	0.14	0.0	0.0	Feb	3
4	0	0.0	0	0.0	10	627.500000	0.02	0.05	0.0	0.0	Feb	3

	Browser	Region	TrafficType	VisitorType	Weekend	Revenue
0	1	1	1	Returning_Visitor	False	False
1	2	1	2	Returning_Visitor	False	False
2	1	9	3	Returning_Visitor	False	False
3	2	2	4	Returning_Visitor	False	False
4	3	1	4	Returning_Visitor	True	False

[179]:
```
OSI_df.shape
```

[179]: (12330, 18)

[180]:
```
OSI_df.describe()
```

[180]:

	Administrative	Administrative_Duration	Informational	\
count	12330.000000	12330.000000	12330.000000	
mean	2.315166	80.818611	0.503569	
std	3.321784	176.779107	1.270156	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	
50%	1.000000	7.500000	0.000000	
75%	4.000000	93.256250	0.000000	
max	27.000000	3398.750000	24.000000	

	Informational_Duration	ProductRelated	ProductRelated_Duration	\
count	12330.000000	12330.000000	12330.000000	
mean	34.472398	31.731468	1194.746220	
std	140.749294	44.475503	1913.669288	
min	0.000000	0.000000	0.000000	
25%	0.000000	7.000000	184.137500	
50%	0.000000	18.000000	598.936905	
75%	0.000000	38.000000	1464.157213	
max	2549.375000	705.000000	63973.522230	

	BounceRates	ExitRates	PageValues	SpecialDay	\
count	12330.000000	12330.000000	12330.000000	12330.000000	
mean	0.022191	0.043073	5.889258	0.061427	
std	0.048488	0.048597	18.568437	0.198917	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.014286	0.000000	0.000000	
50%	0.003112	0.025156	0.000000	0.000000	
75%	0.016813	0.050000	0.000000	0.000000	
max	0.200000	0.200000	361.763742	1.000000	

	OperatingSystems	Browser	Region	TrafficType
count	12330.000000	12330.000000	12330.000000	12330.000000
mean	2.124006	2.357097	3.147364	4.069586

```
std            0.911325      1.717277      2.401591      4.025169
min            1.000000      1.000000      1.000000      1.000000
25%            2.000000      2.000000      1.000000      2.000000
50%            2.000000      2.000000      3.000000      2.000000
75%            3.000000      2.000000      4.000000      4.000000
max            8.000000     13.000000      9.000000     20.000000
```

[181]:
```
OSI_df.nunique()
```

[181]:
```
Administrative              27
Administrative_Duration   3335
Informational               17
Informational_Duration    1258
ProductRelated             311
ProductRelated_Duration   9551
BounceRates               1872
ExitRates                 4777
PageValues                2704
SpecialDay                   6
Month                       10
OperatingSystems             8
Browser                     13
Region                       9
TrafficType                 20
VisitorType                  3
Weekend                      2
Revenue                      2
dtype: int64
```

[182]:
```
OSI_df['VisitorType'].unique()
OSI_df['Month'].unique()
```

[182]:
```
array(['Feb', 'Mar', 'May', 'Oct', 'June', 'Jul', 'Aug', 'Nov', 'Sep',
       'Dec'], dtype=object)
```

0.0.2 Unique Values

Months: 10 months ('Feb', 'Mar', 'May', 'Oct', 'June', 'Jul', 'Aug', 'Nov', 'Sep', 'Dec')

OS: 8

Browser: 13

Region: 9

VisitorType: 3 ('Returning_Visitor', 'New_Visitor', 'Other')

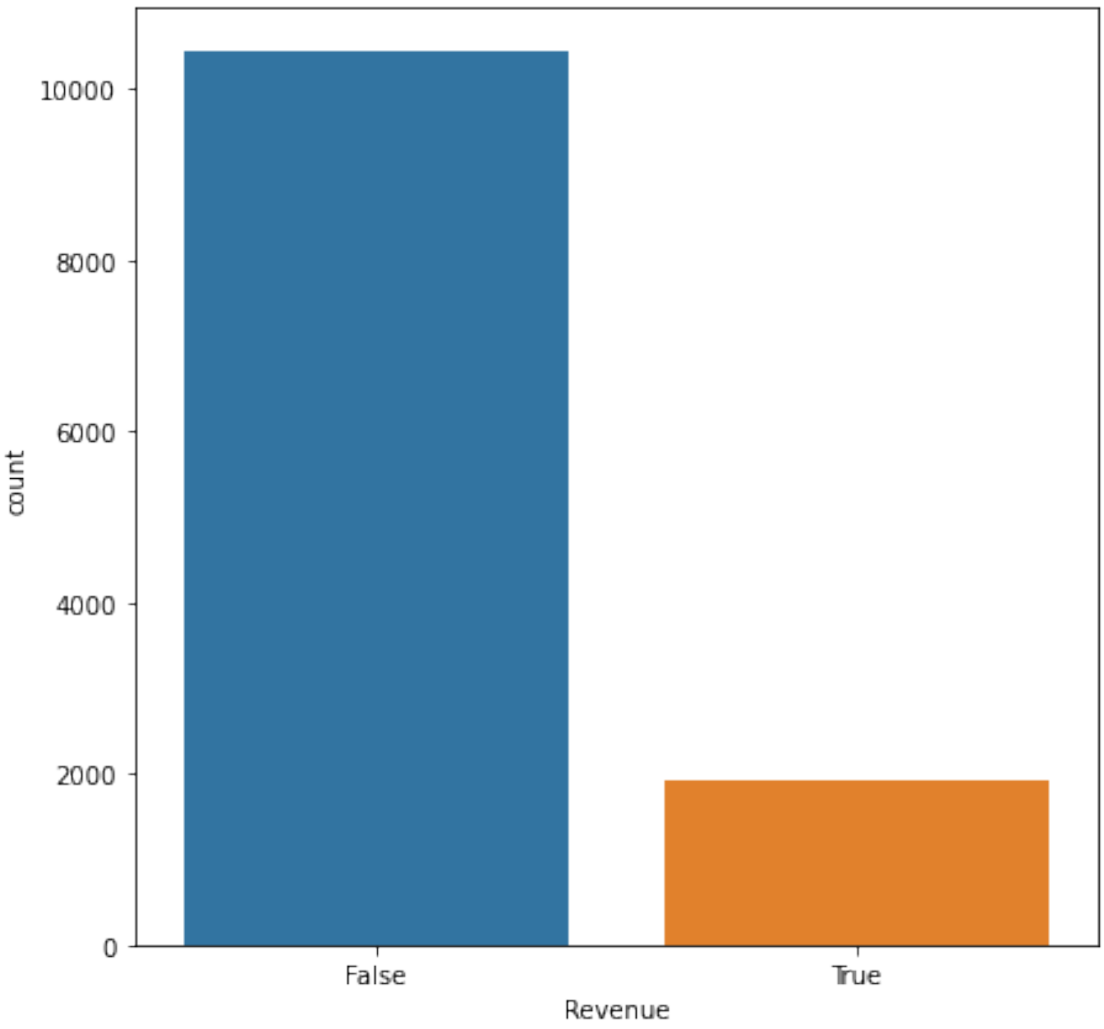
[183]:
```
#Checking for null values
OSI_df.isnull().sum()
```

[183]:
```
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
```

Dataset has no null values

0.0.3 Target variable: Revenue

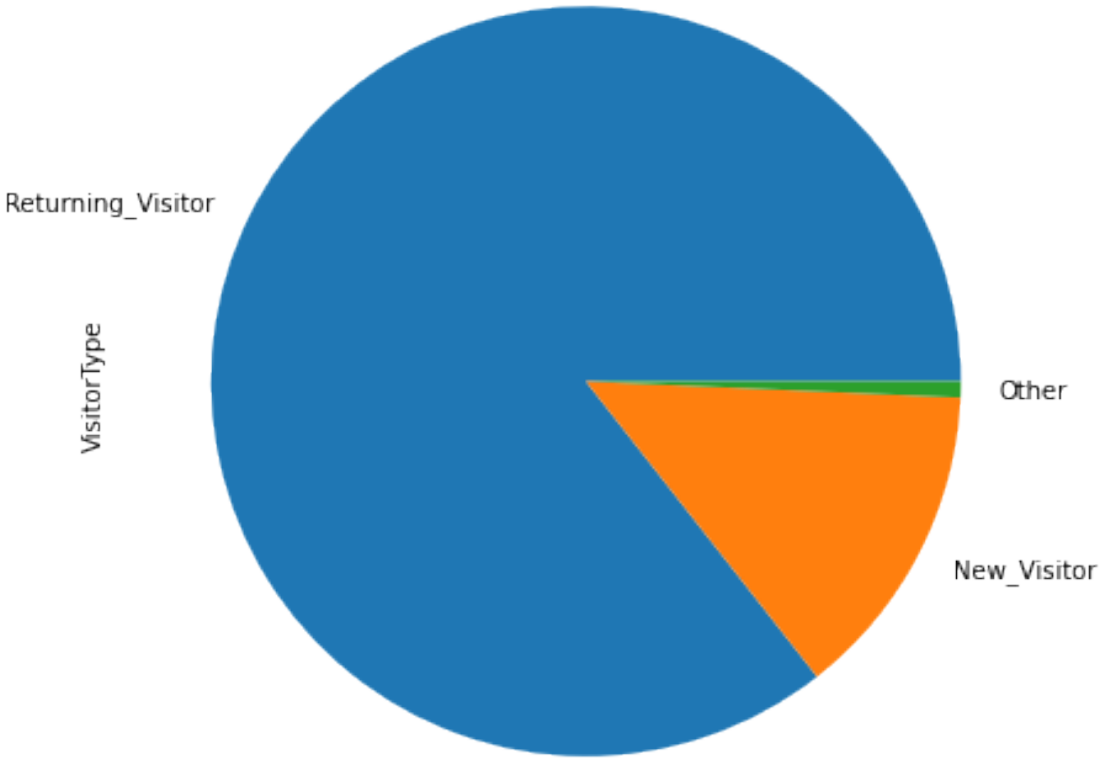
[184]:
```
plt.figure(figsize=(7, 7))
sns.countplot(data=OSI_df, x='Revenue')
plt.show()
```



0.0.4 Pie chart representing the types of site visitors

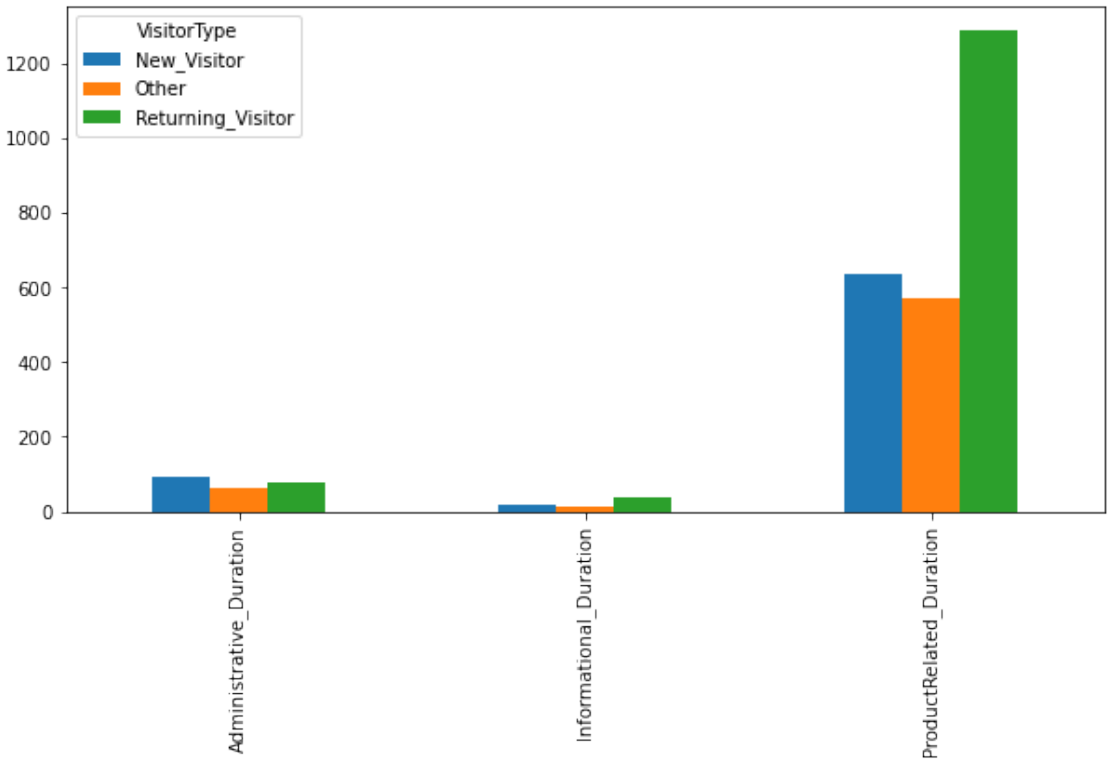
[185]:
```
OSI_df['VisitorType'].value_counts().plot.pie(y='VisitorType', figsize=(7, 7))
```

[185]:
```
<AxesSubplot:ylabel='VisitorType'>
```



[186]:
```
df_comb=OSI_df[['Administrative_Duration','Informational_Duration','ProductRelated_Duration','
pd.pivot_table(df_comb,␣
↪values=['Administrative_Duration','Informational_Duration','ProductRelated_Duration'],colum
↪aggfunc='mean').plot(kind='bar', figsize=(10, 5))
```

[186]: <AxesSubplot:>



0.0.5 The amount of time different visitors spend in different types of pages in the site

New visitors spend more time on administrative page

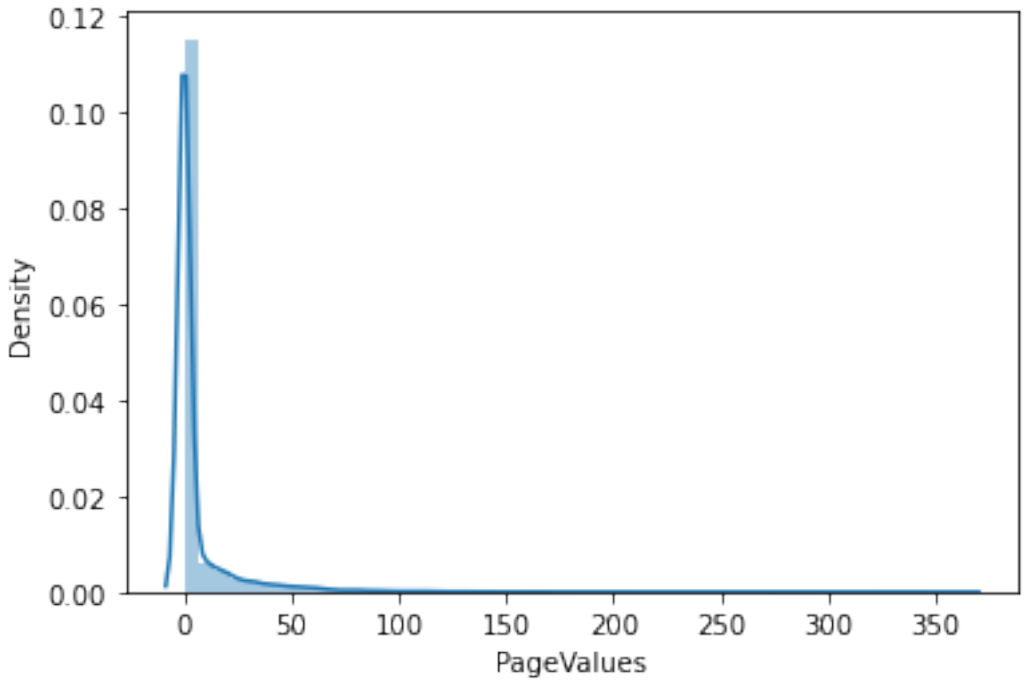
returning visitors spend more time on informational page

returning visitors spend more time on product related page

[187]:
```
sns.distplot(OSI_df['PageValues'])
```

```
C:\Users\shree\anaconda3\envs\cmps530\lib\site-
packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a
deprecated function and will be removed in a future version. Please adapt your
code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

[187]:
```
<AxesSubplot:xlabel='PageValues', ylabel='Density'>
```

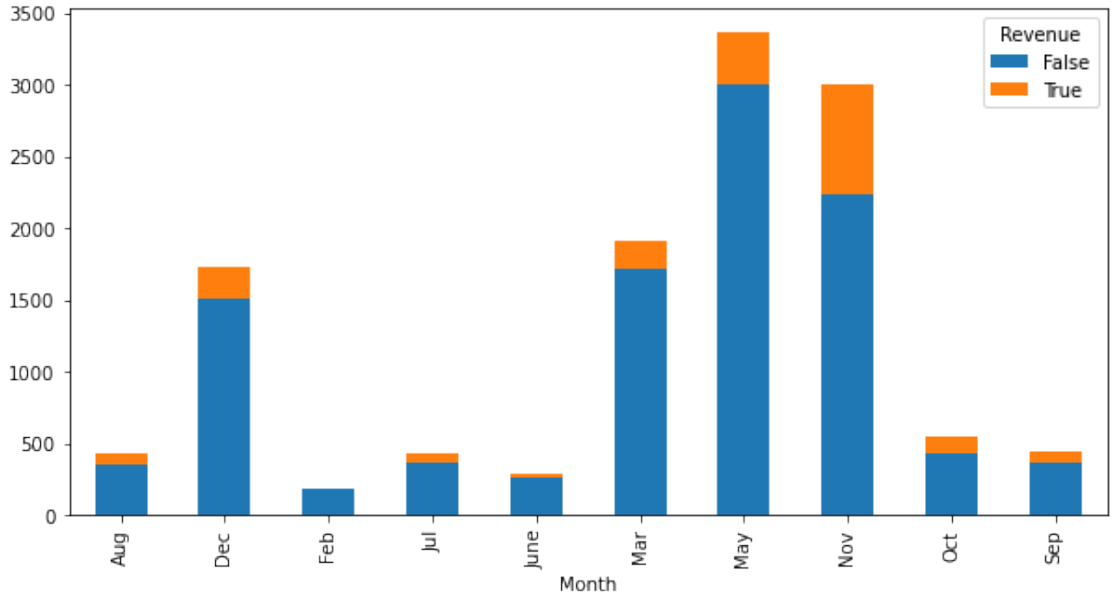


Most pageValues value is in the range of 0 to 5 page value: Page Value is the average value for a page that a user visited before landing on the goal page or completing an Ecommerce transaction (or both). This value is intended to give you an idea of which page in your site contributed more to your site's revenue.

0.0.6 More Data visualization

[188]:
```
OSI_df.groupby('Month')['Revenue'].value_counts().unstack('Revenue').
↪plot(kind='bar', stacked=True, figsize=(10, 5))
```

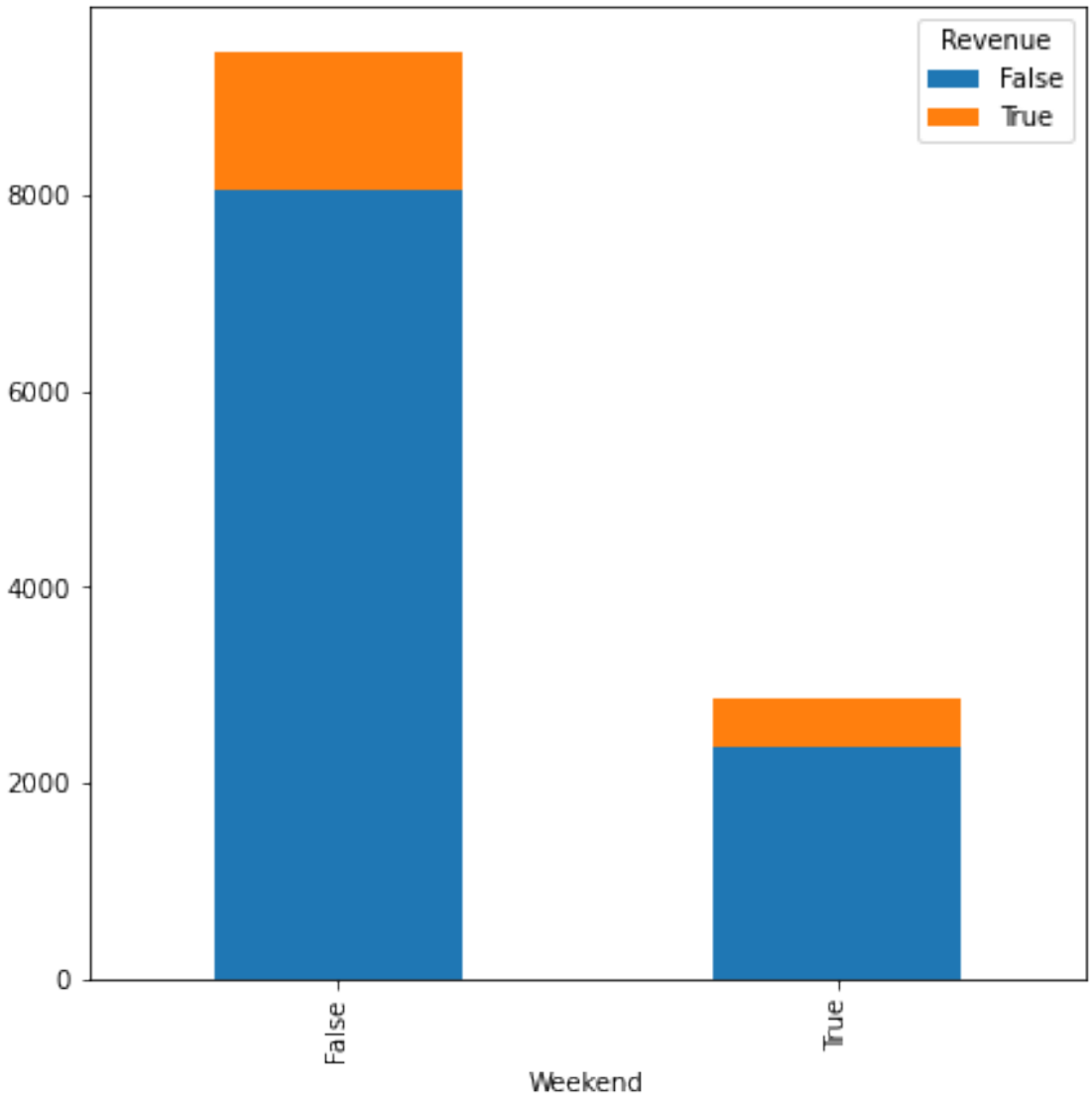
[188]: <AxesSubplot:xlabel='Month'>



Month november has the highest revenue

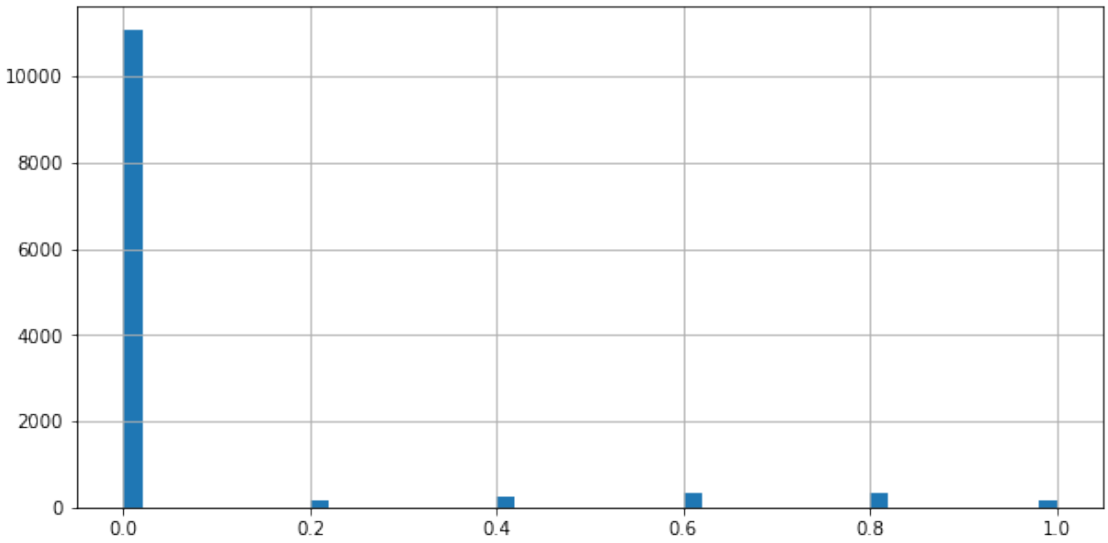
[189]:
```
OSI_df.groupby('Weekend')['Revenue'].value_counts().unstack('Revenue').
↪plot(kind='bar', stacked=True, figsize=(7, 7))
```

[189]: <AxesSubplot:xlabel='Weekend'>



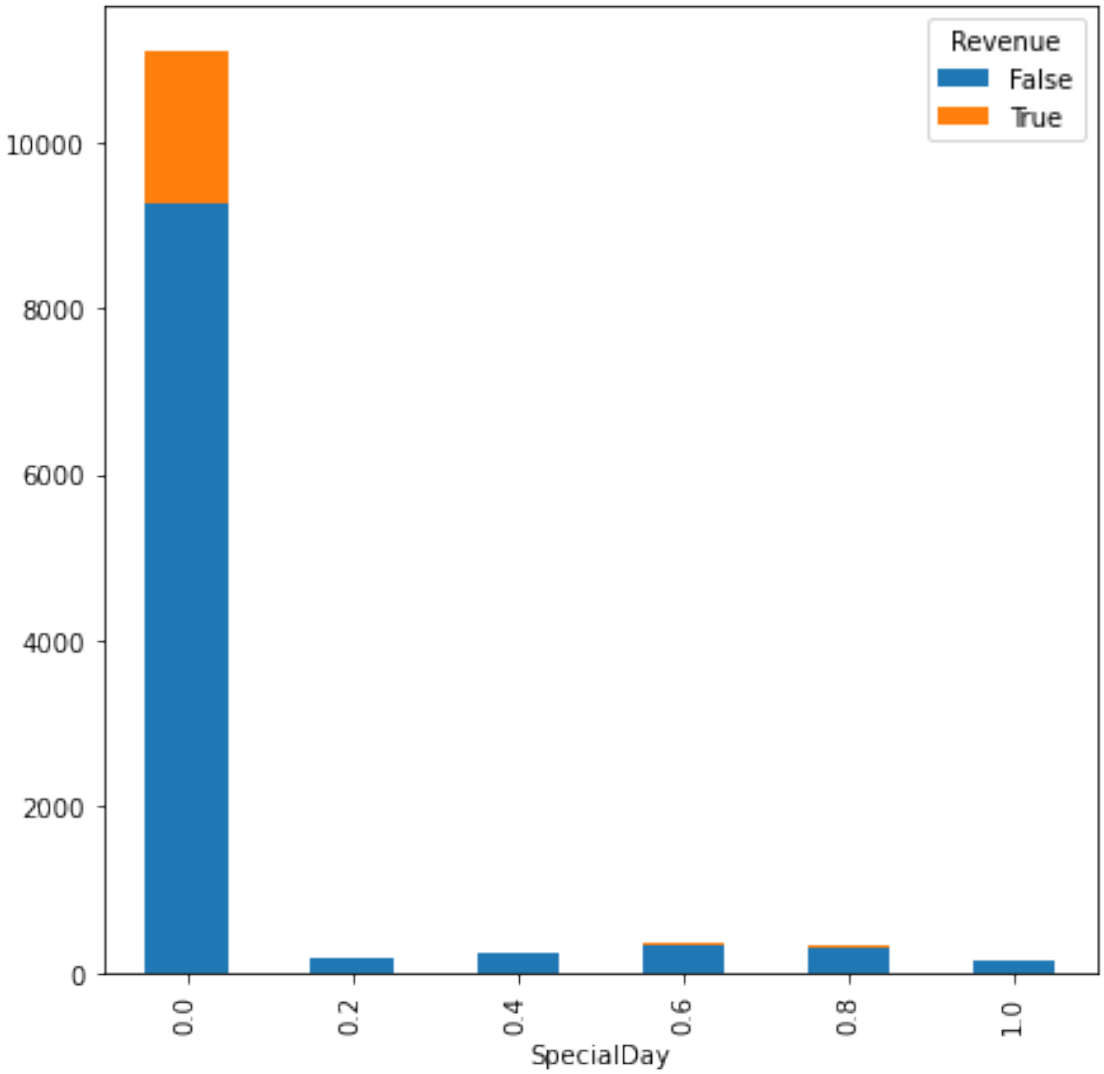
Weekend has less revenue than week days

[190]:
```
OSI_df['SpecialDay'].hist(bins=50, figsize=(10,5))
plt.show()
```



[191]:
```
OSI_df.groupby('SpecialDay')['Revenue'].value_counts().unstack('Revenue').
↪plot(kind='bar', stacked=True, figsize=(7, 7))
```

[191]: <AxesSubplot:xlabel='SpecialDay'>

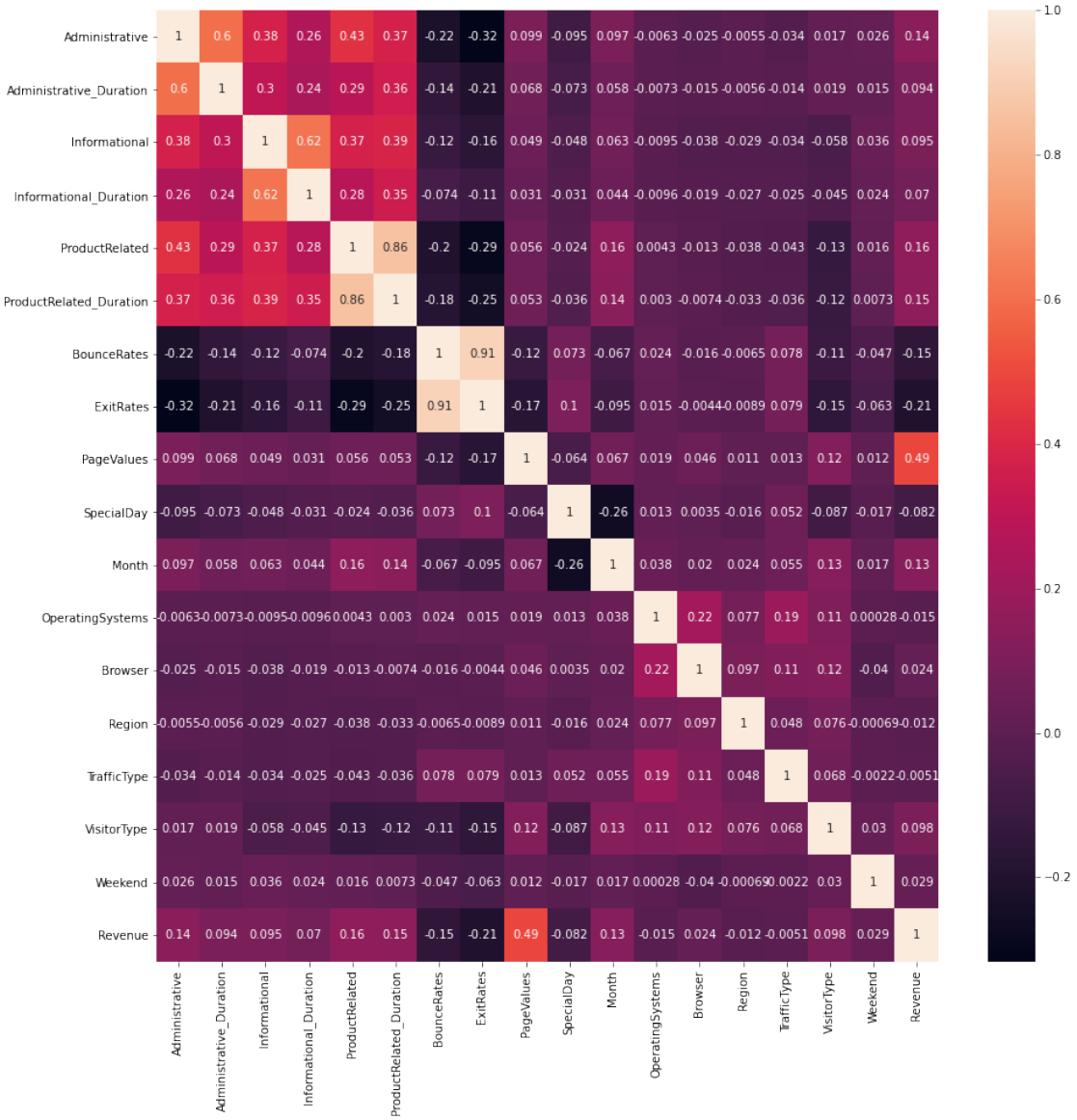


Special day does not have a huge impact on revenue for this dataset in contrast to our normal expectation.

[192]:
```
#This row should not be run more than once or else months values will be␣
 ↪already mapped and month will be empty.
#Data cleaning process
#Before splitting data, data needs to be cleaned as it contains string values␣
 ↪for month and other columns
monthtonumber={'Feb':2, 'Mar':3, 'May':5, 'June':6, 'Jul':7, 'Aug':8, 'Sep':
 ↪9,'Oct':10,'Nov':11, 'Dec':12}
OSI_df['Month']=OSI_df['Month'].map(monthtonumber)

#converting boolean value to numerical value
Isweekend={True:1,False:0}
```

```
OSI_df['Weekend']=OSI_df['Weekend'].map(Isweekend)
OSI_df['Revenue']=OSI_df['Revenue'].map(Isweekend)

#Visitor type were also string which needs converted to numerical value
VisitorTypetonumber={'Returning_Visitor':1, 'New_Visitor':2, 'Other':3}
OSI_df['VisitorType']=OSI_df['VisitorType'].map(VisitorTypetonumber)


Var_Corr = OSI_df.corr()
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(Var_Corr, xticklabels=Var_Corr.columns, yticklabels=Var_Corr.
 ↪columns, annot=True)
```

[192]: <AxesSubplot:>



[193]:
```
scaler = StandardScaler()

scaler.fit(OSI_df.drop('Revenue', axis = 1))
scaled_df = scaler.transform(OSI_df.drop('Revenue', axis = 1))

#converting to panda dataframe
df = pd.DataFrame(scaled_df, columns = OSI_df.columns[:-1])
df.head()
```

[193]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Month	OperatingSystems	Browser	Region	TrafficType	VisitorType	Weekend
0	-0.696993	-0.457191	-0.396478	-0.244931	-0.691003	-0.624348	3.667189	3.229316	-0.317178	-0.308821	-1.665924	-1.233426	-0.790293	-0.894178	-0.762629	-0.401025	-0.550552
1	-0.696993	-0.457191	-0.396478	-0.244931	-0.668518	-0.590903	-0.457683	1.171473	-0.317178	-0.308821	-1.665924	-0.136078	-0.207952	-0.894178	-0.514182	-0.401025	-0.550552
2	-0.696993	-0.457191	-0.396478	-0.244931	-0.691003	-0.624348	3.667189	3.229316	-0.317178	-0.308821	-1.665924	2.058618	-0.790293	2.437081	-0.265735	-0.401025	-0.550552
3	-0.696993	-0.457191	-0.396478	-0.244931	-0.668518	-0.622954	0.573535	1.994610	-0.317178	-0.308821	-1.665924	0.961270	-0.207952	-0.477771	-0.017289	-0.401025	-0.550552
4	-0.696993	-0.457191	-0.396478	-0.244931	-0.488636	-0.296430	-0.045196	0.142551	-0.317178	-0.308821	-1.665924	0.961270	0.374389	-0.894178	-0.017289	-0.401025	1.816360

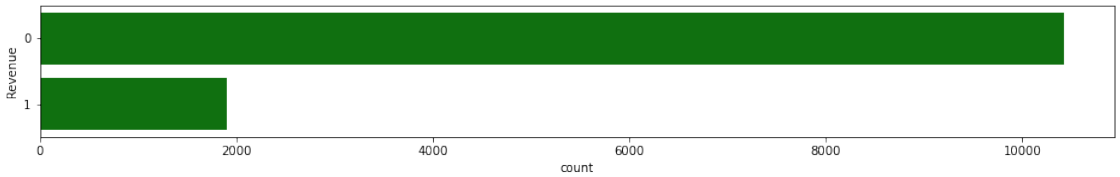
[194]:
```
predictor_names = df.columns
len(df)
```

[194]: 12330

From the bar diagram below, we can see the imbalance in our label. It's apparent that more cases in our data has a prediction "false". This is because maximum number of visitors do not buy anything. This specifies that it is an imbalanced dataset We have to take this into the consideration while

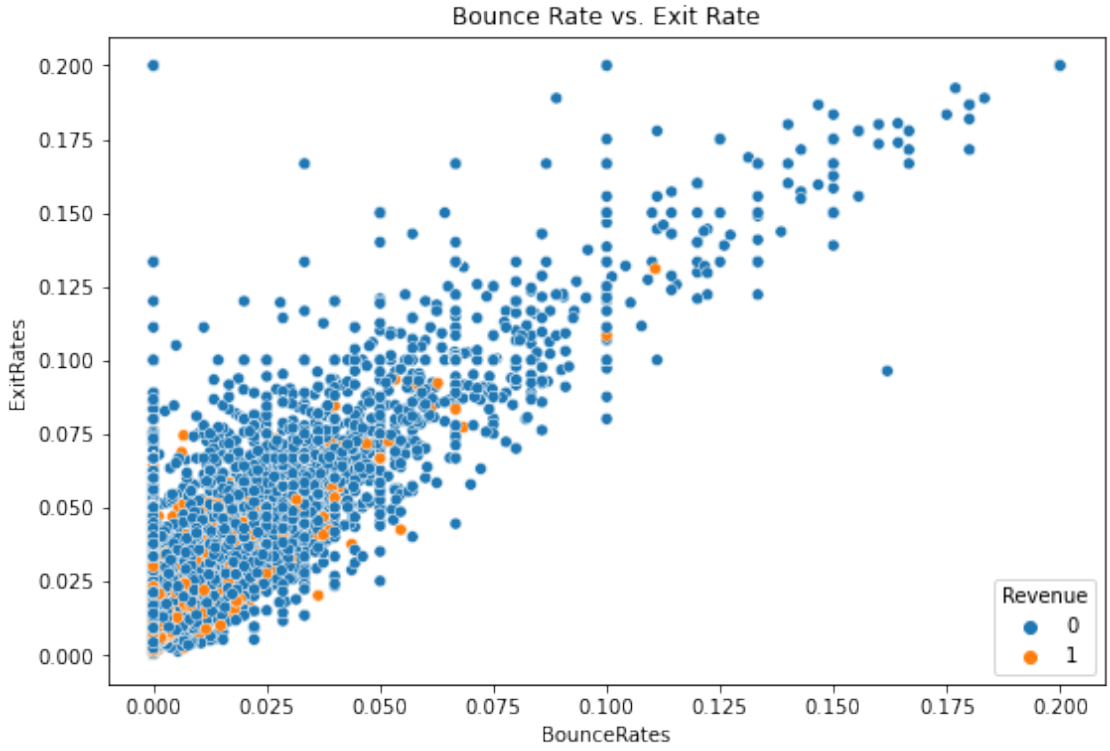
choosing our evaluation metrics.

[195]:
```
plt.figure(figsize=(16,2))
fig = sns.countplot(y="Revenue", data=OSI_df, color='green')
plt.show()
```

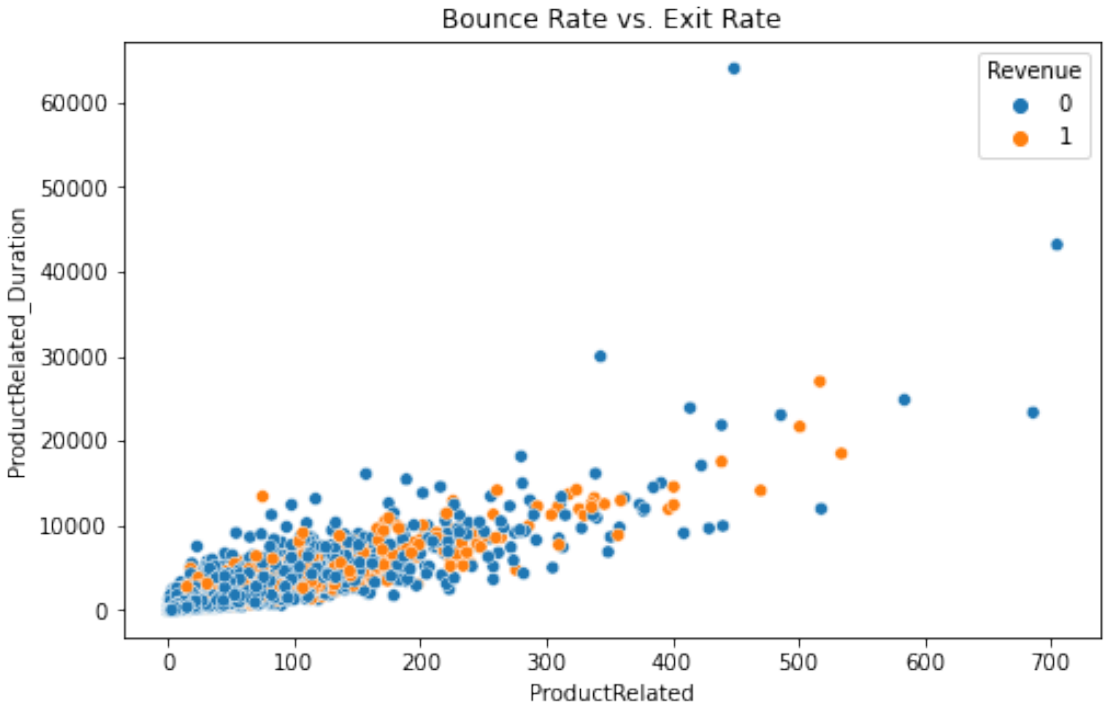


From the correlation plot we got above, we can see that features like ProductRelated and ProductRelated_duration, BounceRates and ExitRates are highly correlated.

[196]:
```
plt.figure(figsize=(9,6))
sns.scatterplot(x= 'BounceRates',y='ExitRates',data=OSI_df, hue='Revenue')
plt.title('Bounce Rate vs. Exit Rate')
plt.show()
```



[197]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(x=␣
↪'ProductRelated',y='ProductRelated_Duration',data=OSI_df,hue='Revenue')
plt.title('Bounce Rate vs. Exit Rate')
plt.show()
```



0.0.7 Logistic Regression

Data Preprocessing

[198]:
```
#### Separating labels from training data
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler

features = ['Administrative', 'Administrative_Duration', 'Informational',
            'Informational_Duration', 'ProductRelated', 'BounceRates',␣
↪'PageValues',
            'Month', 'Region', 'TrafficType', 'VisitorType']

#Training data
X = OSI_df[features]

#Prediction label
y = OSI_df['Revenue']
```

```
# Split the data into x_train and y_train data sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6,␣
↪random_state=42)

# Scale the numeric values
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Training a prediction model using Logistic Regression

[199]:
```
logmodel = LogisticRegression()
score = logmodel.fit(X_train, y_train)
```

Confusion Matrix

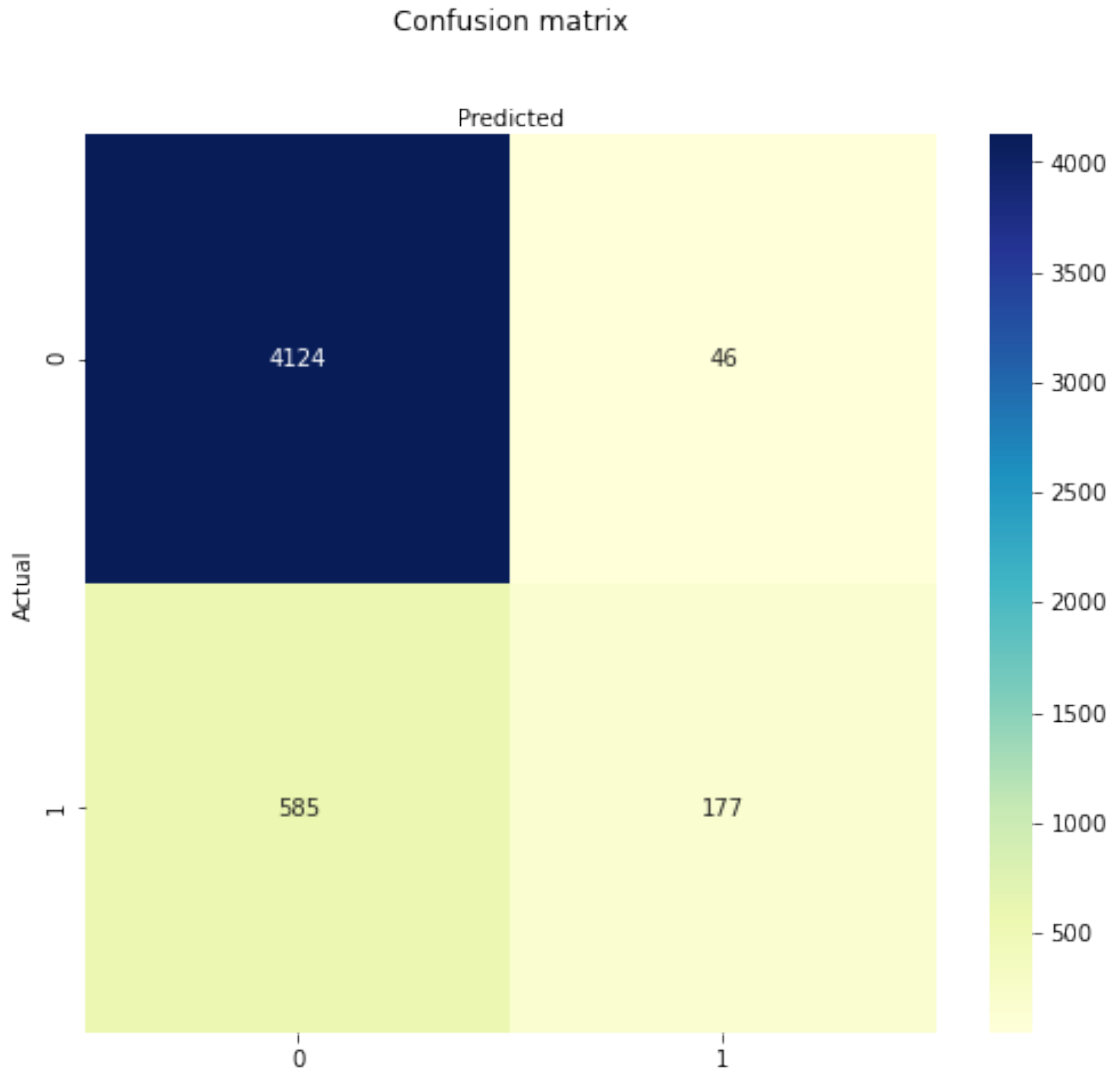
[200]:
```
#importing modules
from sklearn.metrics import confusion_matrix, roc_curve, auc, roc_auc_score
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
```

[201]:
```
y_pred = logmodel.predict(X_test)
cnf_matrix = confusion_matrix(y_test, y_pred)
conf_mat_df = pd.DataFrame(cnf_matrix)
```

Evaluation

[202]:
```
%matplotlib inline
class_names = [False, True]
fig, ax = plt.subplots(figsize=(7, 6))
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)


sns.heatmap(conf_mat_df, annot=True, cmap='YlGnBu', fmt='.0f')
ax.xaxis.set_label_position("top")
plt.tight_layout()

plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual')
plt.xlabel('Predicted')
```

[202]:
```
Text(0.5, 425.19999999999993, 'Predicted')
```



[203]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.99      0.93      4170
           1       0.79      0.23      0.36       762

    accuracy                           0.87      4932
   macro avg       0.83      0.61      0.64      4932
weighted avg       0.86      0.87      0.84      4932
```

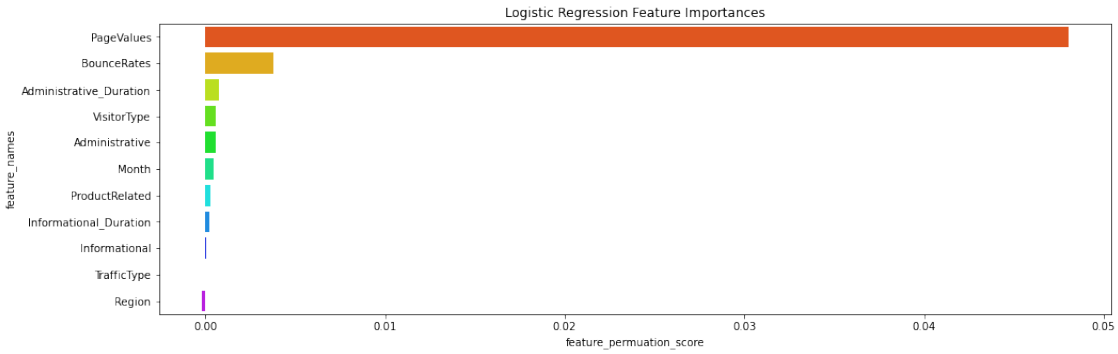
[204]:
```
print('Accuracy: {:.2f}'.format(metrics.accuracy_score(y_test, y_pred)))
print('Precision: {:.2f}'.format(metrics.precision_score(y_test, y_pred)))
```

```
print('Recall: {:.2f}'.format(metrics.recall_score(y_test, y_pred)))
print('f1_score: {:.2f}'.format(metrics.f1_score(y_test, y_pred)))
```

```
Accuracy: 0.87
Precision: 0.79
Recall: 0.23
f1_score: 0.36
```

[205]:
```
from sklearn.inspection import permutation_importance

r = permutation_importance(logmodel, X_test, y_test, n_repeats=30,␣
 ↪random_state=42)

# Plot the barchart
data_im = pd.DataFrame(r.importances_mean, columns=['feature_permuation_score'])
data_im['feature_names'] = X.columns
data_im = data_im.sort_values('feature_permuation_score', ascending=False)

fig, ax = plt.subplots(figsize=(16, 5))

sns.barplot(y=data_im['feature_names'], x="feature_permuation_score",␣
 ↪data=data_im, palette='gist_rainbow')
ax.set_title("Logistic Regression Feature Importances")
```

[205]:
```
Text(0.5, 1.0, 'Logistic Regression Feature Importances')
```



1 Decision Trees

[236]:
```
# data pre-processing
X_train, X_test, y_train, y_test = train_test_split(
      scaled_df, OSI_df['Revenue'], test_size = 0.30)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
```

```
print(y_test.shape)
```

```
(8631, 17)
(3699, 17)
(8631,)
(3699,)
```

1.0.1 Depth 10: Preliminary Tree

We're starting with a preliminary tree that has a maximum depth specified to 10. We test our model under this classifier to get a general idea of the feature interaction. It is no surprise that this is overfitting, and definitely not the best model. However, we can now start changing multiple parameters of the classifier to prune the tree and avoid some overfitting.

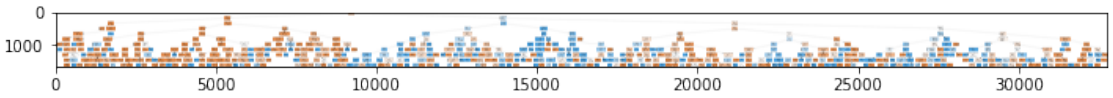
[ ]:
```
# Install a conda package in the current Jupyter kernel
# import sys
# !conda install --yes --prefix {sys.prefix} python-graphviz
```

[237]:
```
# Decision tree modules
import graphviz
from sklearn import tree
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import precision_score, recall_score, accuracy_score,␣
 ↪f1_score

# print(clf.classes_)
class_names = ["No Revenue","Revenue"]
```

[238]:
```
clf_tree_10 = tree.DecisionTreeClassifier(random_state = 0, max_depth = 10)
clf_tree_10 = clf_tree_10.fit(X_train, y_train)

import io
import matplotlib.image as mpimg

from six import StringIO
from sklearn.tree import export_graphviz

dot_data = io.StringIO()

export_graphviz(
        clf_tree_10,
        out_file=dot_data,
        feature_names=predictor_names,
        class_names=class_names,
        rounded=True,
        filled=True
    )
```

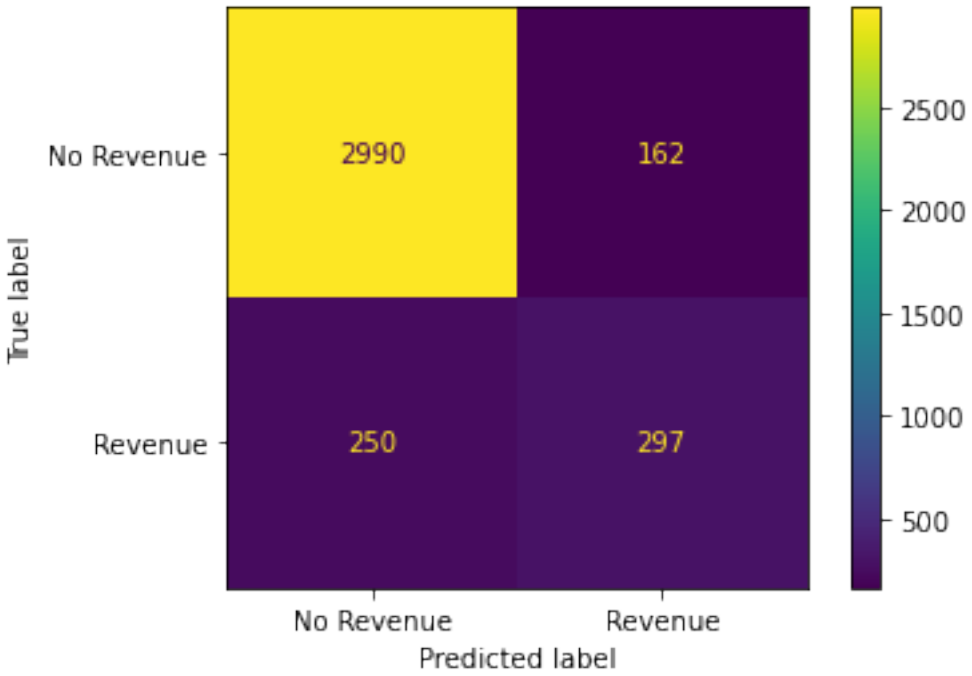
```
filename = "tree_10.png"
pydotplus.graph_from_dot_data(dot_data.getvalue()).write_png(filename)

plt.figure(figsize=(12,12))
img = mpimg.imread(filename)
imgplot = plt.imshow(img)

plt.show()
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.990688 to fit
```



[239]:
```
#plot_confusion_matrix: runs the test data down the tree and draws a confusion␣
↪matrix
plot_confusion_matrix(clf_tree_10, X_test, y_test, display_labels = class_names)
```

[239]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2178052efa0>
```



[240]:
```
y_pred_10 = clf_tree_10.predict(X_test)
print("Accuracy: ", accuracy_score(y_test, y_pred_10))
print("Precision: ", precision_score(y_test, y_pred_10))
print("Recall: ", recall_score(y_test, y_pred_10))
```

```
Accuracy:  0.8886185455528521
Precision:  0.6470588235294118
Recall:  0.5429616087751371
```

Based on the above confusion matrix, we see the following observations for the 3699 rows of test data: - Of the 547 sessions where revenue was generated, 297 or ~54% were correctly classified. - Of the 3152 sessions where revenue was not generated, 2935 or ~94% were correctly classified. It definitely looks like we could prune the tree further to increase performance metrics.

1.0.2 Depth 3: Intermediate Tree

[241]:
```
clf_tree_3 = tree.DecisionTreeClassifier(max_depth = 3, random_state = 0)
clf_tree_3 = clf_tree_3.fit(X_train, y_train)
```

[242]:
```
#plot_confusion_matrix: runs the test data down the tree and draws a confusion␣
↪matrix
plot_confusion_matrix(clf_tree_3, X_test, y_test, display_labels = class_names)
```

[242]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x217e66e6190>

[243]:
```
from sklearn.metrics import classification_report

y_pred_3 = clf_tree_3.predict(X_test)
print("Accuracy: ", accuracy_score(y_test, y_pred_3))
print("Precision: ", precision_score(y_test, y_pred_3))
print("Recall: ", recall_score(y_test, y_pred_3))
```

```
Accuracy:  0.8878075155447418
Precision:  0.611864406779661
Recall:  0.659963436928702
```

Based on the above confusion matrix, we see the following observations for the 3699 rows of test data: - Of the 547 sessions where revenue was generated, 361 or ~66% were correctly classified. - Of the 3152 sessions where revenue was not generated, 2923 or ~93% were correctly classified.

1.0.3 Pruning the tree by manipulating Cost Complexity Parameters

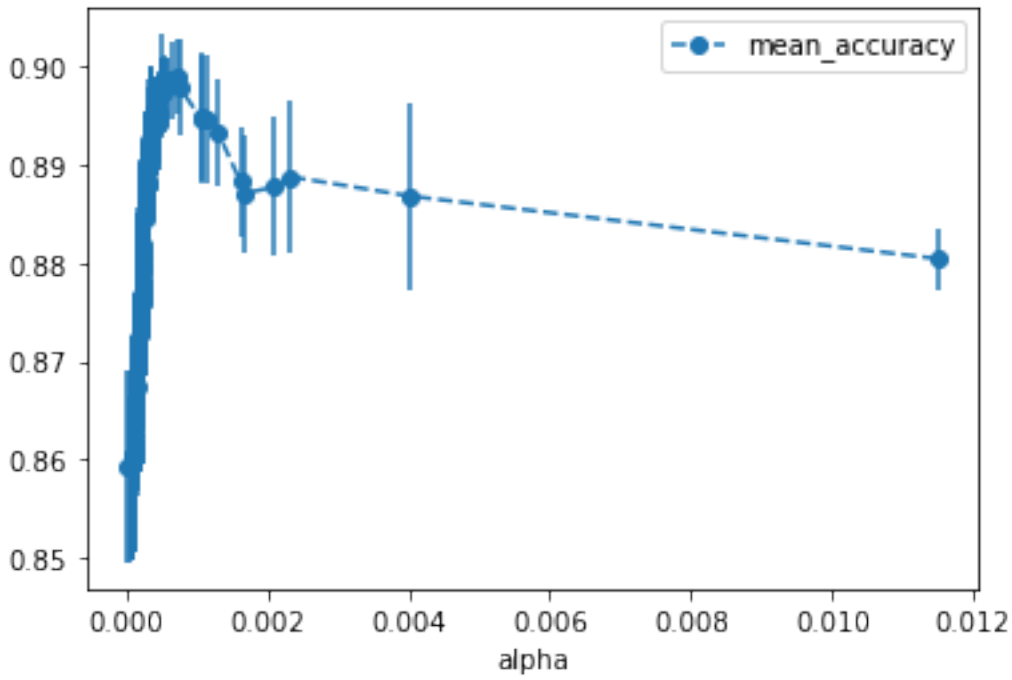
[244]:
```
#import cross validation and evaluation metrics
from sklearn.model_selection import cross_val_score
from sklearn import model_selection
from sklearn.metrics import make_scorer, accuracy_score, precision_score,␣
 ↪recall_score, f1_score

scoring = {'accuracy' : make_scorer(accuracy_score),
           'precision' : make_scorer(precision_score),
           'recall' : make_scorer(recall_score),
           'f1_score' : make_scorer(f1_score)}

path = clf_tree.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
ccp_alphas = ccp_alphas[:-1]

alpha_loop_values_aacuracy=[]
alpha_loop_values_precision=[]
alpha_loop_values_recall=[]

for ccp_alpha in ccp_alphas:
    clf_tree = tree.DecisionTreeClassifier(random_state = 0,␣
 ↪ccp_alpha=ccp_alpha)
    scores = model_selection.cross_validate(clf_tree, X_train, y_train, cv=5,␣
 ↪scoring=scoring)
    accuracy_scores = scores['test_accuracy']
    precision_scores = scores['test_precision']
    recall_scores = scores['test_recall']
#     print(scores['test_precision'])
    alpha_loop_values_aacuracy.append([ccp_alpha, np.mean(accuracy_scores), np.
 ↪std(accuracy_scores)])
```

```
    alpha_loop_values_precision.append([ccp_alpha, np.mean(precision_scores),␣
↪np.std(precision_scores)])
    alpha_loop_values_recall.append([ccp_alpha, np.mean(recall_scores), np.
↪std(recall_scores)])
```

The above snippet takes different values for the cost complexity parameter alpha and employs a 5-fold cross validation to check for variation in performance metrics. Then, we plot the results for accuracy, precision and recall.

[245]:
```
alpha_results = df=pd.DataFrame(alpha_loop_values_aacuracy, columns␣
↪=['alpha','mean_accuracy','std'])
alpha_results.plot(x='alpha',
                   y='mean_accuracy',
                   yerr='std',
                   marker='o',
                   linestyle='--')
```

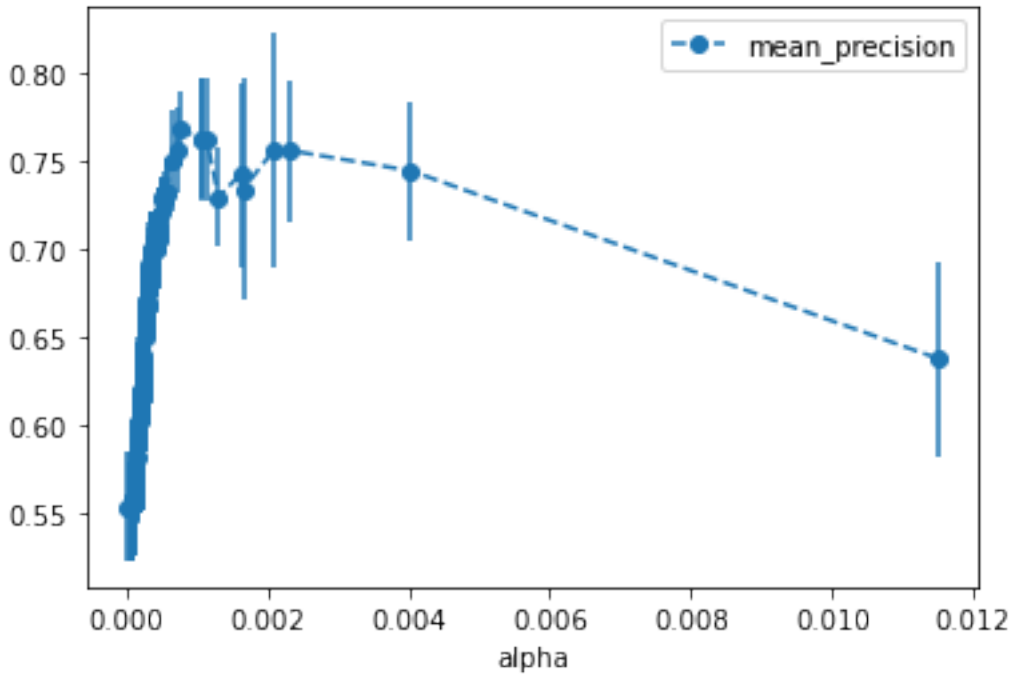
[245]: <AxesSubplot:xlabel='alpha'>



[246]:
```
alpha_results = df=pd.DataFrame(alpha_loop_values_precision, columns␣
↪=['alpha','mean_precision','std'])
alpha_results.plot(x='alpha',
                   y='mean_precision',
                   yerr='std',
                   marker='o',
```

```
                    linestyle='--')
```

[246]: <AxesSubplot:xlabel='alpha'>



[247]:
```
alpha_results[(alpha_results['alpha']>0.0006)
        &
        (alpha_results['alpha']<0.002)]
```

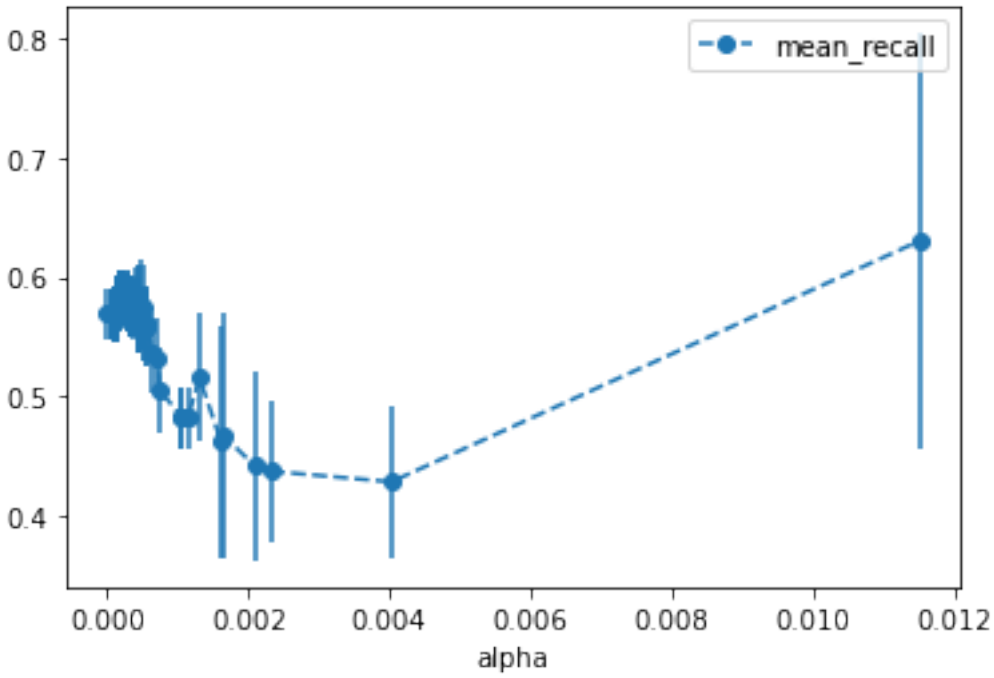
[247]:

	alpha	mean_precision	std
310	0.000644	0.751093	0.028710
311	0.000714	0.756417	0.024104
312	0.000749	0.768489	0.020906
313	0.001049	0.763367	0.034696
314	0.001054	0.762529	0.034957
315	0.001143	0.762529	0.034957
316	0.001307	0.729813	0.028552
317	0.001615	0.742693	0.052674
318	0.001653	0.734185	0.063154

[248]:
```
alpha_results = df=pd.DataFrame(alpha_loop_values_recall, columns␣
↪=['alpha','mean_recall','std'])
alpha_results.plot(x='alpha',
                   y='mean_recall',
                   yerr='std',
                   marker='o',
```

```
                    linestyle='--')
```

[248]: <AxesSubplot:xlabel='alpha'>



[249]:
```
alpha_results[(alpha_results['alpha']>0.0006)
        &
        (alpha_results['alpha']<0.002)]
```

[249]:

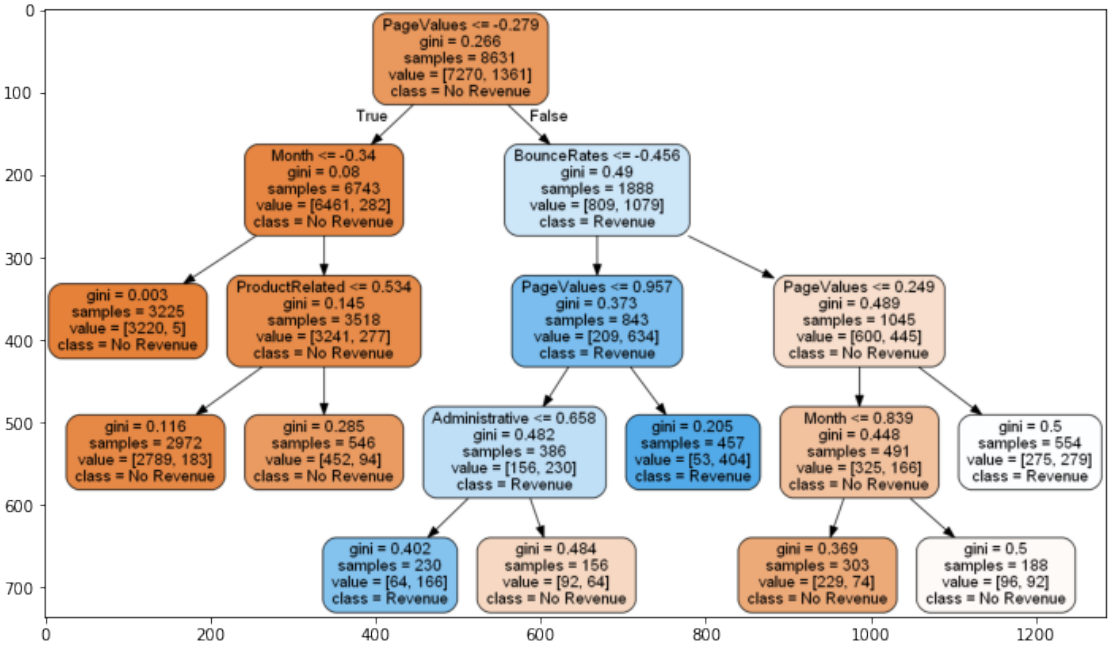
	alpha	mean_recall	std
310	0.000644	0.537134	0.033902
311	0.000714	0.531987	0.032709
312	0.000749	0.505508	0.036171
313	0.001049	0.482735	0.025985
314	0.001054	0.482735	0.025985
315	0.001143	0.482735	0.025985
316	0.001307	0.515740	0.053324
317	0.001615	0.462204	0.097015
318	0.001653	0.467351	0.101790

We focus more on an alpha value that maximizes recall and precision since this is a highly imbalanced dataset.

1.0.4 Final Tree

[250]:
```
clf_tree = tree.DecisionTreeClassifier(random_state = 0, ccp_alpha=0.0013)
clf_tree = clf_tree.fit(X_train, y_train)
```

[251]:
```
import io
import matplotlib.image as mpimg

from six import StringIO
from sklearn.tree import export_graphviz

dot_data = io.StringIO()

export_graphviz(
        clf_tree,
        out_file=dot_data,
        feature_names=predictor_names,
        class_names=class_names,
        rounded=True,
        filled=True
    )

filename = "tree.png"
pydotplus.graph_from_dot_data(dot_data.getvalue()).write_png(filename)

plt.figure(figsize=(12,12))
img = mpimg.imread(filename)
imgplot = plt.imshow(img)

plt.show()
```



[252]:
```
plot_confusion_matrix(clf_tree, X_test, y_test, display_labels = class_names)
```

[252]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2178046d6a0>

[254]:
```
y_pred_4 = clf_tree.predict(X_test)
print("Accuracy: ", accuracy_score(y_test, y_pred_4))
print("Precision: ", precision_score(y_test, y_pred_4))
print("Recall: ", recall_score(y_test, y_pred_4))
print("F1-Score: ", f1_score(y_test, y_pred_4))
```

```
Accuracy:  0.8932143822654771
Precision:  0.6433962264150943
Recall:  0.623400365630713
F1-Score:  0.6332404828226555
```

Based on the above confusion matrix, we see the following observations for the 3699 rows of test data: - Of the 547 sessions where revenue was generated, 341 or ~62% were correctly classified. - Of the 3152 sessions where revenue was not generated, 2963 or ~94% were correctly classified.

[258]:
```
importance = clf_tree.feature_importances_

features_dict = {}
for feature, importance in zip(predictor_names, importance):
    features_dict[feature] = importance

data_list = list(features_dict.items())
importance_values = pd.DataFrame(data_list, columns = ["Feature␣
 ↪Name","Importance"])
importance_values.sort_values('Importance', ascending=False)
```

[258]:

	Feature Name	Importance
8	PageValues	0.843520
6	BounceRates	0.095564
10	Month	0.032707
0	Administrative	0.017355
4	ProductRelated	0.010855
5	ProductRelated_Duration	0.000000
3	Informational_Duration	0.000000
7	ExitRates	0.000000
1	Administrative_Duration	0.000000
9	SpecialDay	0.000000
2	Informational	0.000000
11	OperatingSystems	0.000000
12	Browser	0.000000
13	Region	0.000000
14	TrafficType	0.000000
15	VisitorType	0.000000
16	Weekend	0.000000

Based on this model, we see that PageValue is the most important feature in determining whether or not a purchase was made at the end of a session, and hence revenue was generated. This is also backed up by the fact that PageValues is the root node in the decision tree model.

It also confirms our observation from EDA that SpecialDay doesn't really make any difference to the revenue generation. Similarly, 13 other columns like Administrative, Operating System and so on also have no effect on the revenue. This is very helpful in eliminating non-necessary features.

[259]:
```
y_train_R = (y_train == 1) # True for all revenue values 1, False for all 0
y_test_R = (y_test == 1)
```

[260]:
```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(clf_tree, X_train, y_train_R, cv=3)
```

[261]:
```
from sklearn.metrics import precision_score, recall_score, accuracy_score

print("Accuracy: ", accuracy_score(y_train_R, y_train_pred))
print("Precision: ", precision_score(y_train_R, y_train_pred))
print("Recall: ", recall_score(y_train_R, y_train_pred))
print("F1-Score: ", f1_score(y_train_R, y_train_pred))
```

```
Accuracy:  0.897230911829452
Precision:  0.73558648111332
Recall:  0.5437178545187362
F1-Score:  0.6252640473172794
```

1.1 Comparing the Single Decision Tree with Ensemble Learning Methods

The following models train a Random Forest Classifier and a Bagging Classifier with 500 trees.

[267]:
```
from sklearn.ensemble import RandomForestClassifier

clf_rnd = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16,␣
↪random_state=42)
clf_rnd.fit(X_train, y_train)

# y_pred_rf = rnd_clf.predict(X_test)
```

[267]:
```
RandomForestClassifier(max_leaf_nodes=16, n_estimators=500, random_state=42)
```

[268]:
```
precision_scores = {}
accuracy_scores = {}
recall_scores = {}
f1_scores = {}
for clf in (clf_tree, clf_rnd):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy_scores[clf.__class__.__name__] = accuracy_score(y_test, y_pred)
    precision_scores[clf.__class__.__name__] = precision_score(y_test, y_pred)
    recall_scores[clf.__class__.__name__] = recall_score(y_test, y_pred)
    f1_scores[clf.__class__.__name__] = f1_score(y_test, y_pred)
```

[269]:
```
tmp_list = [accuracy_scores, precision_scores, recall_scores, f1_scores ]
```

[270]:
```
df = pd.DataFrame(tmp_list, index =␣
↪["Accuracy","Precision","Recall","F1-Scores"])
```

[271]:
```
df
```

[271]:

	DecisionTreeClassifier	RandomForestClassifier
Accuracy	0.893214	0.896999
Precision	0.643396	0.741279
Recall	0.623400	0.466179
F1-Scores	0.633240	0.572391

Decision Trees are comparatively efficient in dealing with imbalanced datasets. Accuracy is not a good metric to measure efficiency here since only ~15% of the observations in the sample have true values for revenue generation. All the models have similar accuracy in this case. We focus on precision, recall and F-1 scores.

As we know, high precision indicates a low number of false positives. In this case, Random Forest presents a higher precision score, which means this classifier is more exact than the other two.

Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives. Here, Decision Trees have a higher recall score than random forests.

F1-Score the weighted average of precision and recall. In case of online shoppers' behavior prediction, F1 score seems to be a perfect metrics to combine both precision and recall in mesuring the effectiveness of our model in predicting purchasing vs. non-purchasing behaviour.

1.2 K Nearest Neighbors

[272]:
```
knn = KNeighborsClassifier(n_neighbors = 1)
knn_fit= knn.fit(X_train, y_train)
y_pred= knn.predict(X_test)
```

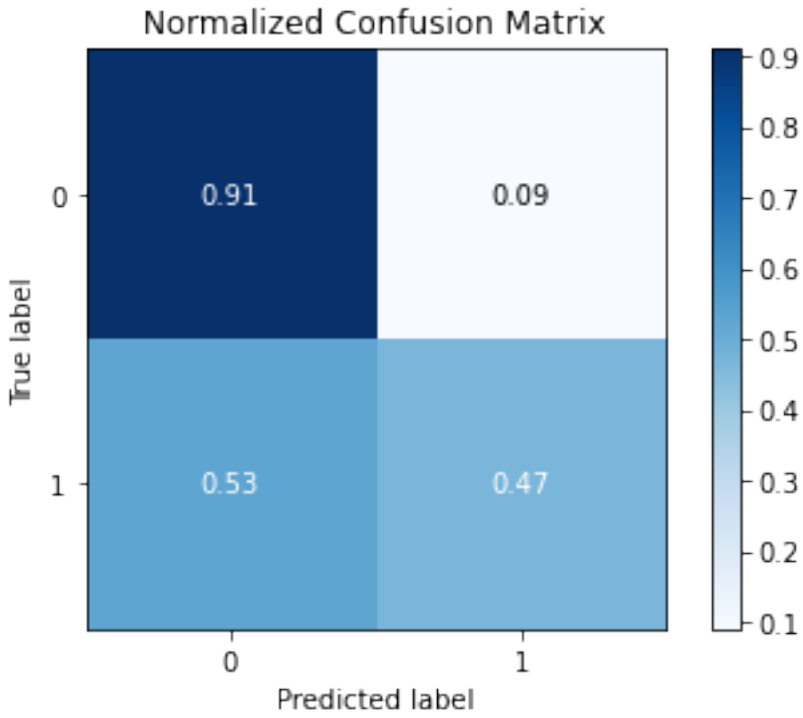
1.2.1 Normalized classification matrix

[279]:
```
import sys
!conda install --yes --prefix {sys.prefix} scikit-plot
```

```
Collecting package metadata (current_repodata.json): …working… done
Solving environment: …working… done

## Package Plan ##

  environment location: C:\Users\shree\anaconda3\envs\cmps530

  added / updated specs:
    - scikit-plot
```

```
The following packages will be UPDATED:

  ca-certificates    conda-forge::ca-certificates-2021.10.~ --> pkgs/main::ca-
certificates-2021.10.26-haa95532_2

The following packages will be SUPERSEDED by a higher-priority channel:

  certifi            conda-forge::certifi-2021.10.8-py38ha~ -->
pkgs/main::certifi-2021.10.8-py38haa95532_0
  openssl            conda-forge::openssl-1.1.1l-h8ffe710_0 -->
pkgs/main::openssl-1.1.1l-h2bbff1b_0


Preparing transaction: …working… done
Verifying transaction: …working… done
Executing transaction: …working… done
```

[281]:
```
import scikitplot as skplt
plt = skplt.metrics.plot_confusion_matrix(y_test,y_pred, normalize=True)
```



1.2.2 Confusion matrix with actual values

[282]:
```
plot_confusion_matrix(knn_fit, X_test, y_test, display_labels = class_names)
```

[282]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21781726ac0>
```

[283]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      3152
           1       0.48      0.47      0.47       547

    accuracy                           0.85      3699
   macro avg       0.69      0.69      0.69      3699
weighted avg       0.84      0.85      0.85      3699
```

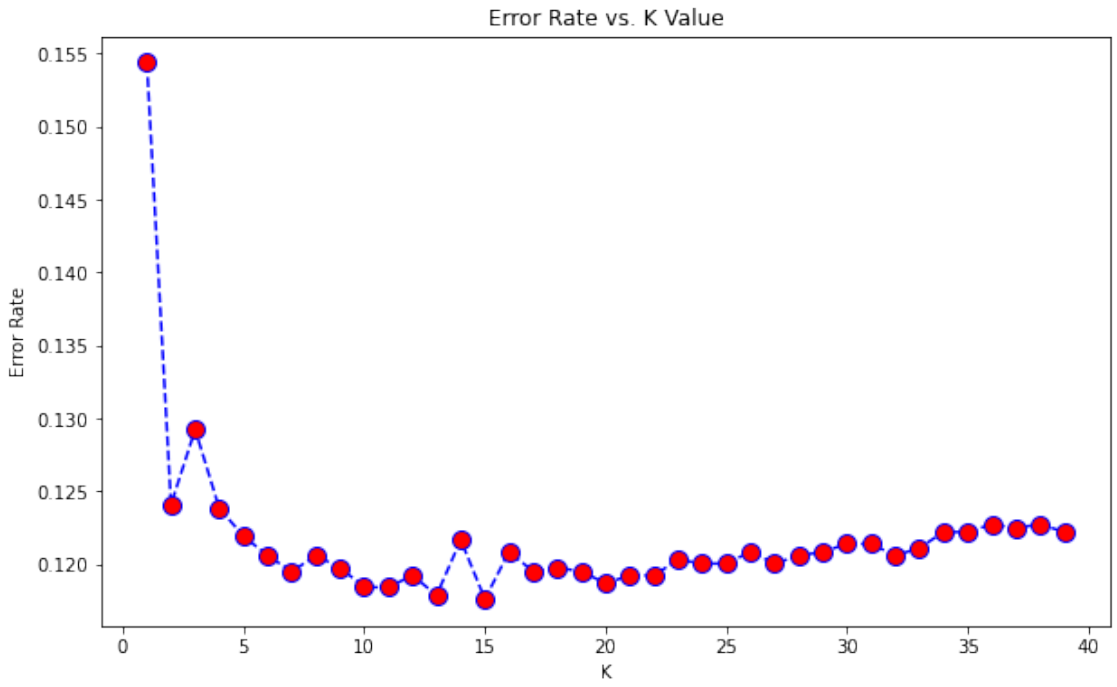
[284]:
```
accuracy_ = accuracy_score(y_test, y_pred)
precision_ = precision_score(y_test, y_pred)
recall_ = recall_score(y_test, y_pred)
f1_ = f1_score(y_test, y_pred)
print("Accuracy: " ,accuracy_)
print("Precision: " ,precision_)
print("Recall: " ,recall_)
print("F1: ", f1_)
```

```
Accuracy:  0.8456339551230062
Precision:  0.4777777777777778
Recall:  0.4716636197440585
F1:  0.4747010119595217
```

[285]:
```
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

[291]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

[291]:
```
Text(0, 0.5, 'Error Rate')
```

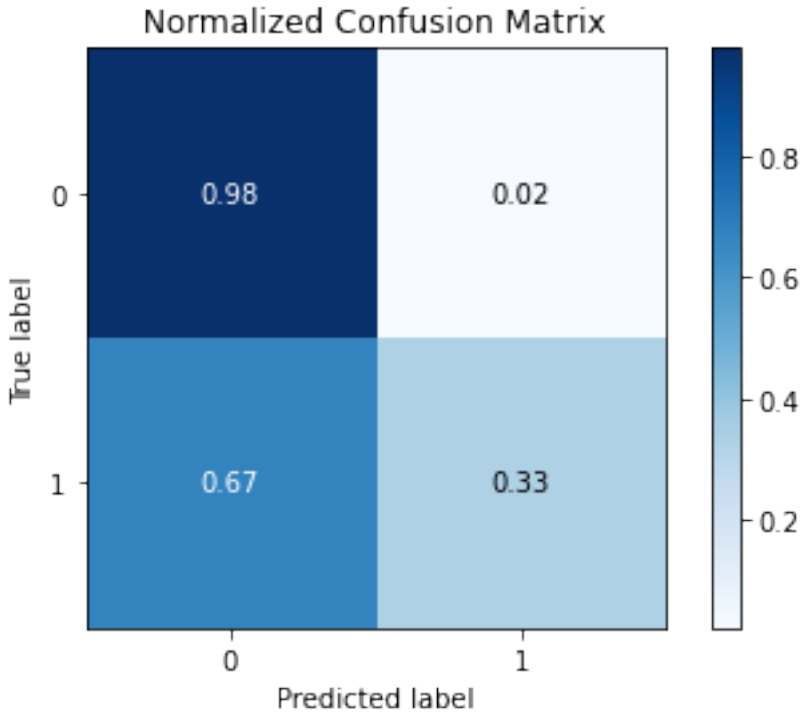


[292]:
```
knn = KNeighborsClassifier(n_neighbors = 10)
knn_fit= knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

[293]:
```
print(classification_report(y_test, y_pred))
```

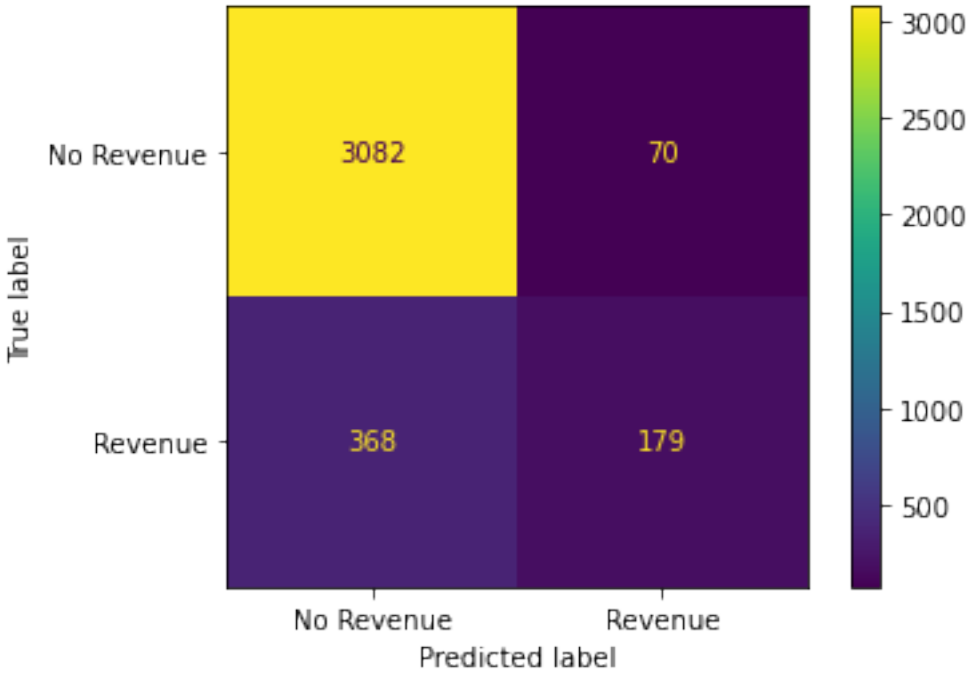
```
              precision    recall  f1-score   support

           0       0.89      0.98      0.93      3152
           1       0.72      0.33      0.45       547

    accuracy                           0.88      3699
   macro avg       0.81      0.65      0.69      3699
weighted avg       0.87      0.88      0.86      3699
```

[294]:
```
plt = skplt.metrics.plot_confusion_matrix(y_test,y_pred, normalize=True)
```



[295]:
```
plot_confusion_matrix(knn_fit, X_test, y_test, display_labels = class_names)
```

[295]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x217812f1e20>



[296]:
```
accuracy_ = accuracy_score(y_test, y_pred)
precision_ = precision_score(y_test, y_pred)
recall_ = recall_score(y_test, y_pred)
f1_ = f1_score(y_test, y_pred)
print("Accuracy" ,accuracy_)
print("Precision" ,precision_)
print("Recall" ,recall_)
print("F1", f1_)
```

```
Accuracy 0.8815896188158961
Precision 0.7188755020080321
Recall 0.3272394881170018
F1 0.44974874371859297
```

[ ]:

[ ]: