| Alejandro Garro Salazar |
| Data Analytics & Big Data 2018c-8 (L) |

Online Shoppers Purchasing Intention Report

Project Overview

Arguably, one of the most popular online activities/services is shopping, and it is not hard to see why. Shopping online has clear advantages over shopping in-store: you have the entire product catalog at the tip of your fingers, you can do your shopping at your own pace, practically wherever you want (as long as you have an internet connection), and whenever you want; online stores are open 24/7. There is no closing time. Also, you can compare the same product from various sellers in seconds. And these are just to name a few advantages… for the customers, that is.

However, although e-commerce has been growing at an exponential pace, and more and more stores make available their products for online shopping, this also means that the competition gets fiercer every day. One of the problems that these types of stores face is that most often than not, potential customers visit the websites and do not actually complete a purchase. This results in loss of revenues for e-tailers (online retailers), as they have to maintain their websites, as well as comply with most of the financial responsibilities that regular brick and mortar stores have to face.

This project aims to provide a solution to this problem, by studying the customers' behavior while doing their online shopping. This is achieved by analyzing the actions taken by the website visitors and determining their shopping intent. This information is passed through several machine learning algorithms to create a predictive model that will indicate whether the customer will make a purchase or not. This should enable the online retailer to take further action and target specific promotions or campaigns to lure in the customers to make a purchase.

This project is based on data procured from the UCI Machine Learning Repository. The abstract of this dataset indicates the following:

"*Of the 12,330 sessions in the dataset, 84.5% (10,422) were negative class samples that did not end with shopping, and the rest (1908) were positive class samples ending with shopping.*"

The objective of this report is to provide a summary of key findings and recommendations for the Online Shoppers Purchasing Intention issue.

1. Project framework and plan of attack

The elected approach for this project will be to use Python and the following main libraries:

- NumPy
- Pandas
- Scikit-Learn
- Matplotlib
- Seaborn
- Pickle

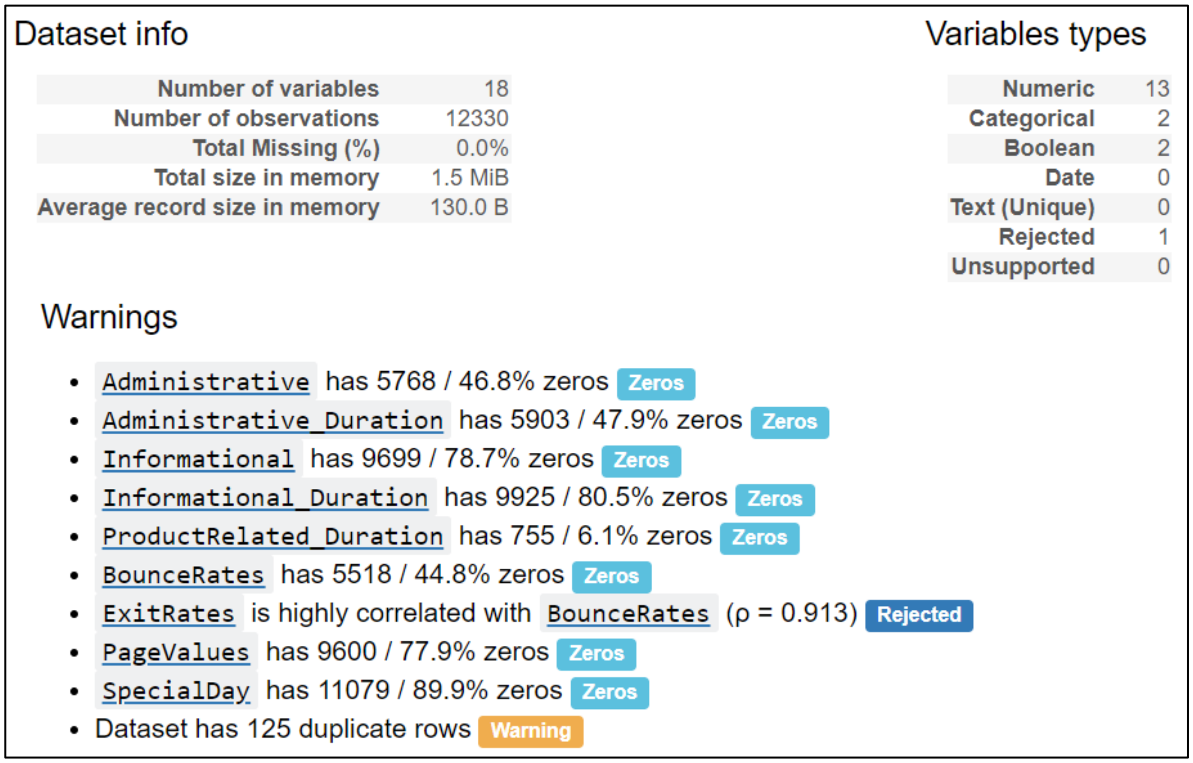
These libraries are fundamental for most data science projects: *NumPy* is great for working on large, multi-dimensional arrays and matrices. It also brings along a large collection of high-level mathematical functions to operate on these arrays. *Pandas* offers data structures and operations for manipulating numerical tables and time series. Scikit-learn is a machine learning library, which features various classification, regression and clustering algorithms. *Matplotlib* and *Seaborn* are plotting libraries, great for data visualization. And lastly, *Pickle* implements binary protocols, used for serializing and de-serializing a Python object structure; for our purposes, this creates a file for saving objects (such as trained models).

This project will determine if the result of a given session will finalize in a purchasing transaction or not, by creating a predictive model that will result in a True/False prediction. Therefore, it is a binary classification problem. We will perform a detailed explanation of the steps taken to address the issue at hand in the following sections.

2. <u>Preliminary Visualizations and Relevant Analyses</u>

In order to familiarize with the data, and have a better understanding of the incumbent variables, we will make use of a very powerful tool called Pandas Profiling. This is a module which performs a very thorough exploratory data analysis, using a single line of code. The output is an interactive report in web format, which includes the data type of the variable, several descriptive statistics (mean, std. deviation, quantiles, etc.), checks for missing values, multicollinearities, histograms, and even correlation matrices.

From the profile report output, we can ratify that -as the abstract indicated- the dataset consists of 18 variables with 12,330 observations. More details can be observed in Figure 1 below.



Dataset info

Number of variables	18
Number of observations	12330
Total Missing (%)	0.0%
Total size in memory	1.5 MiB
Average record size in memory	130.0 B

Variables types

Numeric	13
Categorical	2
Boolean	2
Date	0
Text (Unique)	0
Rejected	1
Unsupported	0

Warnings

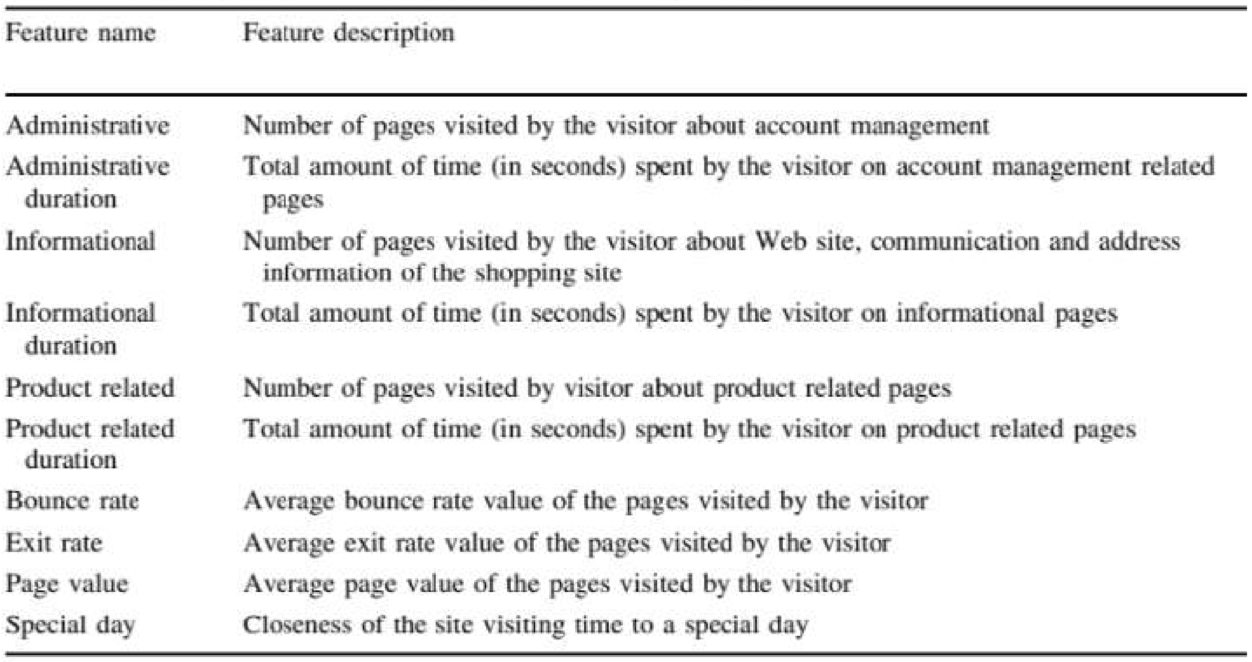
- Administrative has 5768 / 46.8% zeros Zeros
- Administrative_Duration has 5903 / 47.9% zeros Zeros
- Informational has 9699 / 78.7% zeros Zeros
- Informational_Duration has 9925 / 80.5% zeros Zeros
- ProductRelated_Duration has 755 / 6.1% zeros Zeros
- BounceRates has 5518 / 44.8% zeros Zeros
- ExitRates is highly correlated with BounceRates (ρ = 0.913) Rejected
- PageValues has 9600 / 77.9% zeros Zeros
- SpecialDay has 11079 / 89.9% zeros Zeros
- Dataset has 125 duplicate rows Warning

Figure 1. EDA Profile Report output

A few important observations that we can quickly derive is that there are no missing values, most of the variables have a large percentage of zeroes, and that it detected a collinearity between two variables: ExitRates and BounceRates.

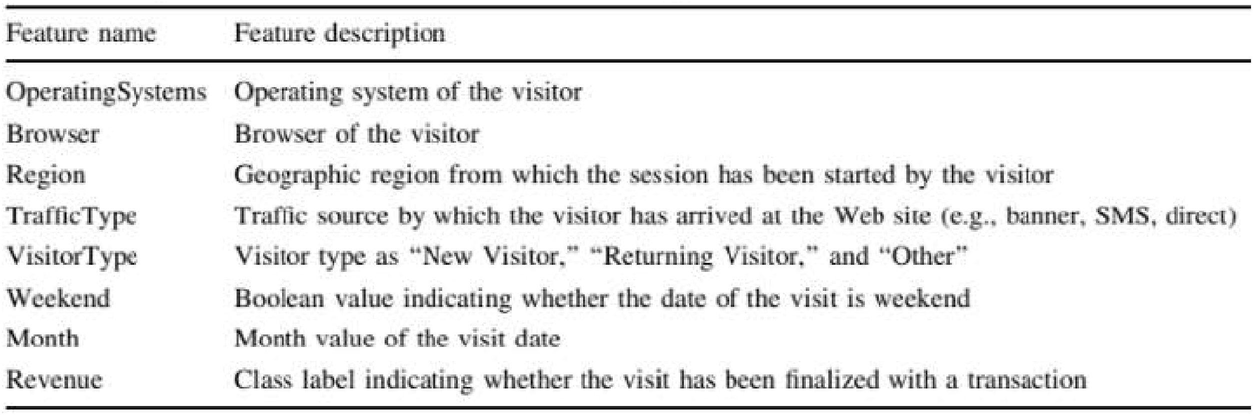
From the dataset documentation (see the citation request[1]) variables descriptions can be observed (divided in numerical and categorical variables) in Table 1 and Table 2 below:

Table 1. Numerical features.



Feature name	Feature description
Administrative	Number of pages visited by the visitor about account management
Administrative duration	Total amount of time (in seconds) spent by the visitor on account management related pages
Informational	Number of pages visited by the visitor about Web site, communication and address information of the shopping site
Informational duration	Total amount of time (in seconds) spent by the visitor on informational pages
Product related	Number of pages visited by visitor about product related pages
Product related duration	Total amount of time (in seconds) spent by the visitor on product related pages
Bounce rate	Average bounce rate value of the pages visited by the visitor
Exit rate	Average exit rate value of the pages visited by the visitor
Page value	Average page value of the pages visited by the visitor
Special day	Closeness of the site visiting time to a special day

Table 2. Categorical features.



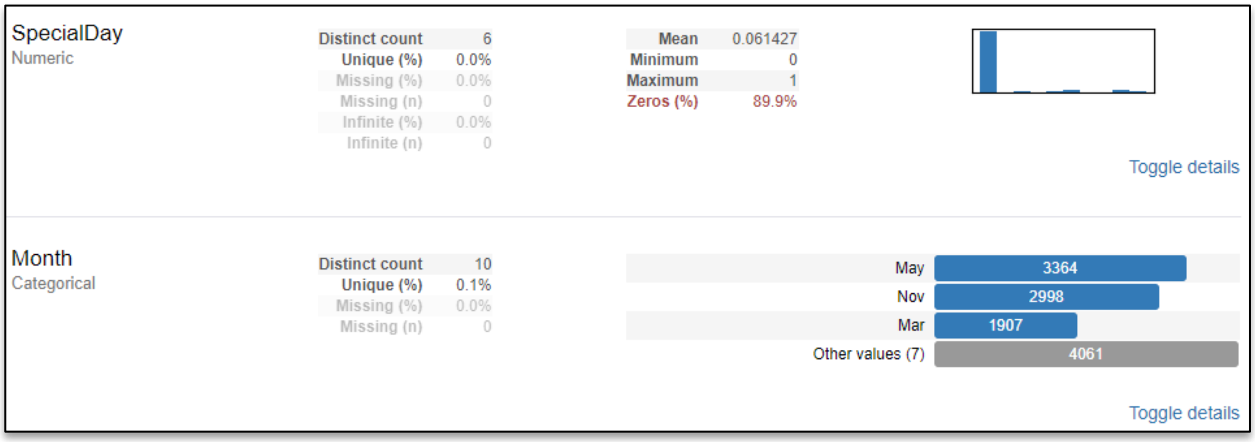
Feature name	Feature description
OperatingSystems	Operating system of the visitor
Browser	Browser of the visitor
Region	Geographic region from which the session has been started by the visitor
TrafficType	Traffic source by which the visitor has arrived at the Web site (e.g., banner, SMS, direct)
VisitorType	Visitor type as "New Visitor," "Returning Visitor," and "Other"
Weekend	Boolean value indicating whether the date of the visit is weekend
Month	Month value of the visit date
Revenue	Class label indicating whether the visit has been finalized with a transaction

However, the data read from the dataset does not match with the appropriate data types for most of the categorical variables. This is because the data encoding is done using numbers, and when loaded into a data frame, Python interprets these numerical values as numerical features. This issue will be addressed further ahead, during the pre-processing steps of this project.

In Figure 2, we can observe an actual Numerical feature (SpecialDay), and an actual categorical

[1] Citation request: Sakar, C.O., Polat, S.O., Katircioglu, M. et al. Neural Comput & Applic (2018).

feature (Month) correctly classified by Python upon loading. However, in Figure 3 we can observe an example of categorical features, that were initially loaded as numerical (understandably).



SpecialDay
Numeric

Distinct count	6
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	0.061427
Minimum	0
Maximum	1
Zeros (%)	89.9%

Toggle details

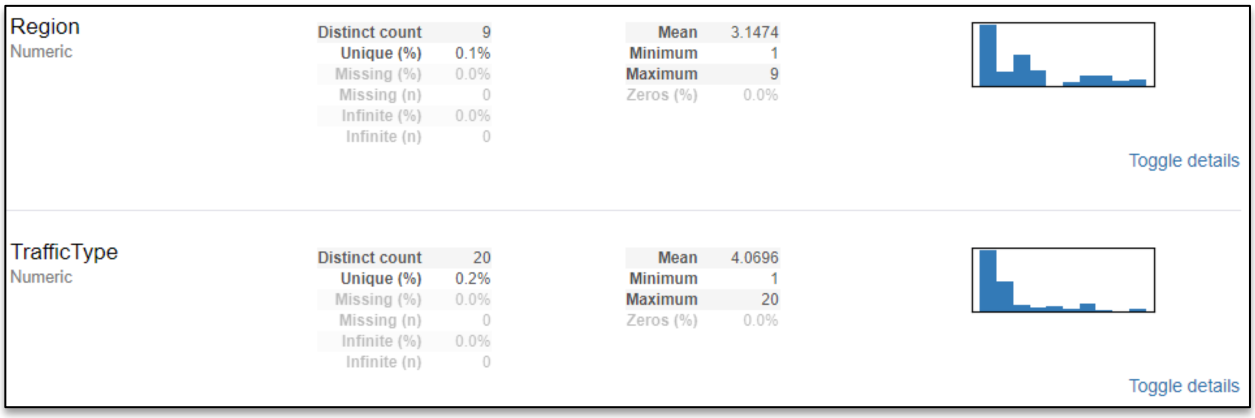
Month
Categorical

Distinct count	10
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0

May	3364
Nov	2998
Mar	1907
Other values (7)	4061

Toggle details

Figure 2. Example of a Numeric Feature and a Categorical Feature.



Region
Numeric

Distinct count	9
Unique (%)	0.1%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	3.1474
Minimum	1
Maximum	9
Zeros (%)	0.0%

Toggle details

TrafficType
Numeric

Distinct count	20
Unique (%)	0.2%
Missing (%)	0.0%
Missing (n)	0
Infinite (%)	0.0%
Infinite (n)	0

Mean	4.0696
Minimum	1
Maximum	20
Zeros (%)	0.0%

Toggle details

Figure 3. Example of Categorical variables read as Numerical.

Finally, the Profile Report provides correlation matrices, using two different correlation coefficients: Pearson and Spearman. In Figure 4, we can observe the Pearson correlation matrix, and how it is more of a qualitative analysis, given the absence of actual values in the cells. These matrices provide a nice preliminary visualization of the correlation between the variables; however, for better granularity and a more precise evaluation, a correlation matrix presenting quantitative results will be created as part of the pre-processing activities.

Nonetheless, it is important to note that -as expected- the results viewed graphically in the correlation matrices from the Pandas Profiling report match the results obtained from the quantitative analysis done using the correlation matrix in the pre-processing steps.

Supplementary to the charts provided by the Pandas Profiling report, further visualizations were created to obtain additional insights of the variables, and how they relate to the target variable, 'Revenue'. These were generated using the plotting capabilities from the Matplotlib Pyplot and Seaborn modules. Some interesting observations resulted from the analysis of these plots.

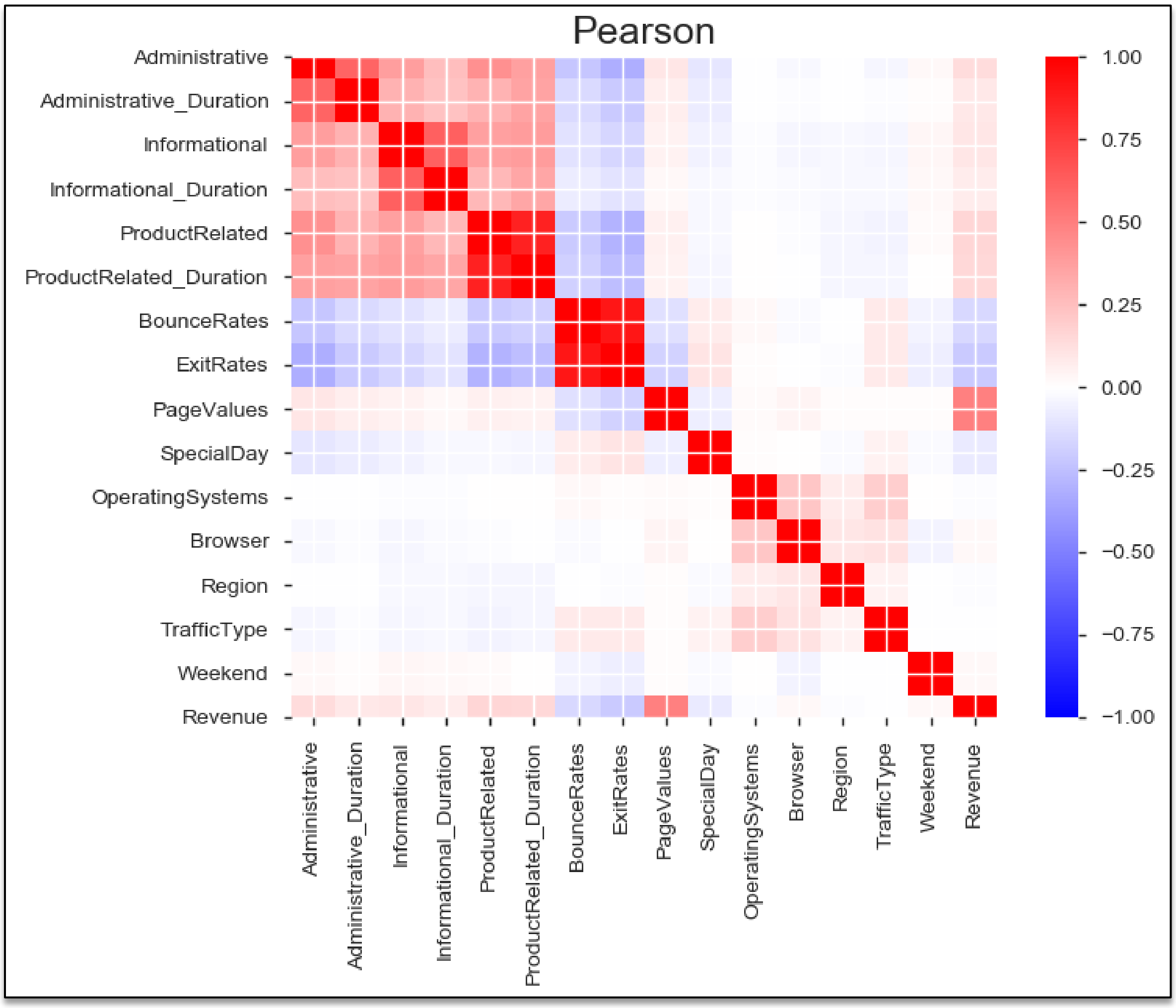


Figure 4. Pearson correlation matrix.

For example, when plotting the Month vs the Revenue (see Figure 5), a very unusual observation was immediately evident: the months of January and April are “missing” from the data.

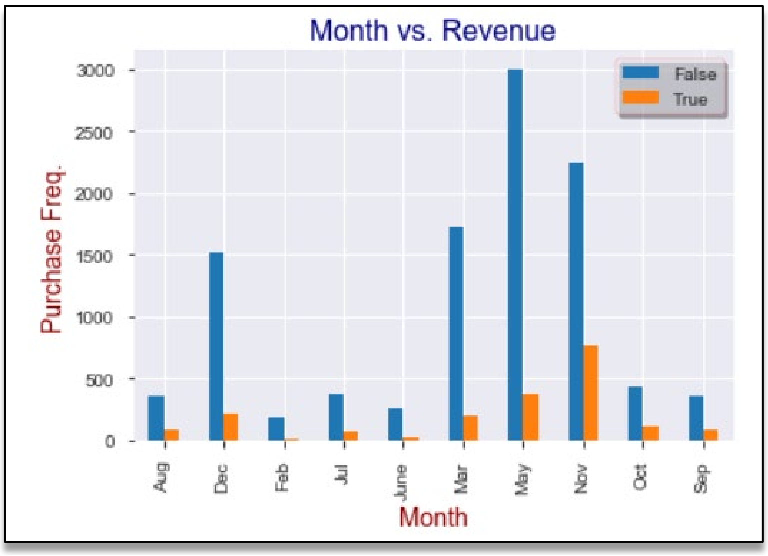


Figure 5. Month vs. Revenue

Also, analyzing the Visitor Type vs. Revenue chart (see Figure 6), the returning visitors have a quite significant traffic, but very few of them actually finalize a purchase. This has tremendous marketing potential and more data should be collected to explore this even further.

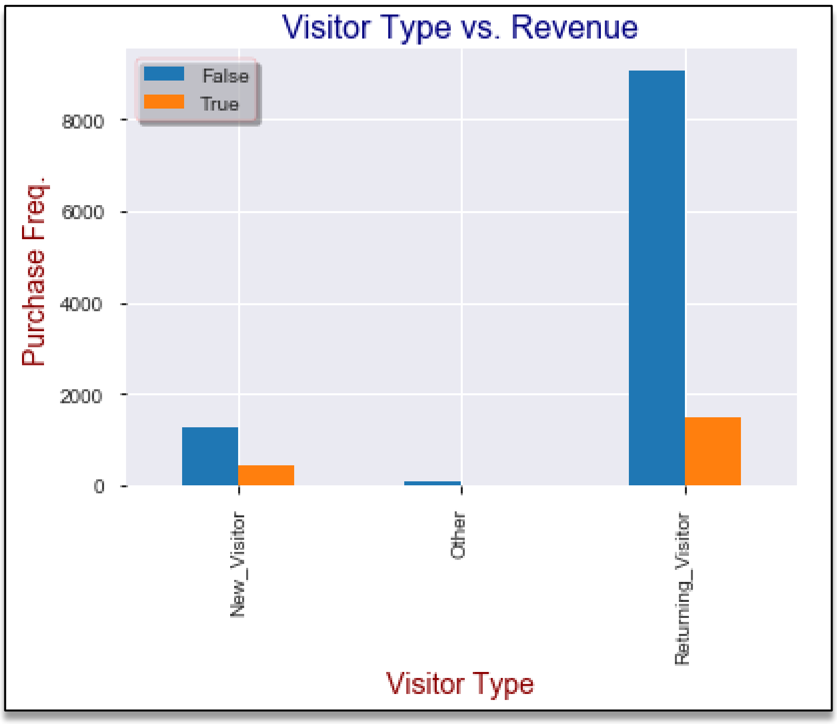
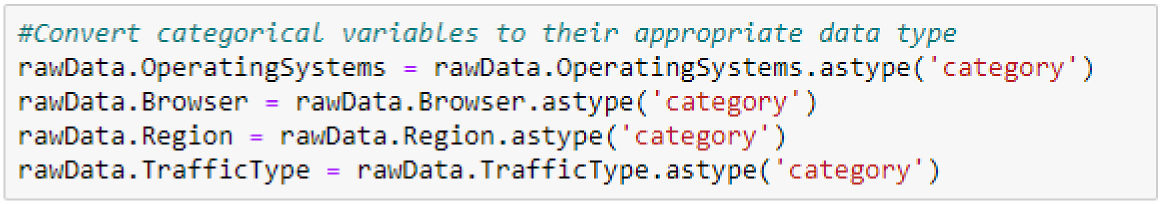


Figure 6. Visitor Type vs. Revenue

3. <u>Data Cleaning and Pre-processing</u>

Based on the analysis from the previous section, the data is already fairly clean, but there are still quite some pre-processing activities that need to take place. Firstly, the data types need to match the variables that will be used. Therefore, the variables '*OperatingSystems*', '*Browser*', '*Region*', and '*TrafficType*' were coerced into *category* data type, by utilizing the `as.type()` function, as seen in Figure 7, below.



```
#Convert categorical variables to their appropriate data type
rawData.OperatingSystems = rawData.OperatingSystems.astype('category')
rawData.Browser = rawData.Browser.astype('category')
rawData.Region = rawData.Region.astype('category')
rawData.TrafficType = rawData.TrafficType.astype('category')
```

Figure 7. Data type coercion to 'category' .

The next step is to address the potential multicollinearities that the variables might present. From the EDA, the profile report already identified *ExitRates* and *BounceRates* to be highly correlated, and therefore one of these two variables must be dropped from the dataset. However, another correlation matrix was created, to perform a more quantitative analysis of the coefficients that the variables will generate. See the resulting correlation matrix in Annex 1.

After analyzing the resulting correlation matrix, there were two collinearities identified that need to be addressed: the first one is the one previously identified by the EDA profile report, between *ExitRates* and *BounceRates* (0.913), and also in a lesser way, variables *ProductRelated_Duration* and *ProductRelated* have a high correlation coefficient o 0.861.

Further analysis is done to these variables, and given that the zeroes on *ProductRelated_Duration* are far greater than those of *ProductRelated* (755 vs none), and the correlation with the target variable (Revenue) is greater on the *ProductRelated* feature, *ProductRelated_Duration* gets dropped from the dataset.

Data Segmentation

The next step will be to segment the data into numerical variables, categorical variables, and the target variable, for individual pre-processing of numerical and categorical data types. The Numerical Features were saved in a variable called `'NumFeat'`, which contains the following features:

- Administrative
- Admistrative_Duration
- Informational
- ProductRelated
- ExitRates
- Informational_Duration
- PageValues
- SpecialDay

And the same with the categorical features, saved in a variable called `'CatFeat'`:

- Month
- OperatingSystems
- Browser
- Region
- TrafficType
- Visitor_Type
- Weekend
- Revenue*

**Revenue is a Boolean variable, and the Target for this project.*

Numerical Data Processing

Given the large difference in the range of the numerical features, these features will be standardized (subtract the mean and then divide by their standard deviation), to avoid issues with the models. For instance, for the '*Administrative*' feature, the minimum is 0, and the maximum value is 27; however, '*Administrative_Duration'* has a minimum of 0, but the maximum is 3,398.8. These are evidently in very different scales for a model to treat them as similar.

Categorical Data Processing

In order for the classifier algorithms to work correctly, they need numerical data as inputs. Hence, we are going to process the categorical variables using a technique called one-hot-encoding to create binary columns that will represent these categories, by their individual values. For instance, the '*VisitorType*' feature has three unique values: *New_Visitor*, *Returning_Visitor*, and *Other*. What this encoding will do is to create three new features that stem from the original one, succeeded by their values: '*VisitorType_New_Visitor*',*' VisitorType_Returning_Visitor',* and '*VisitorType_Other*'.

Then, it will fill the observations with 1 or 0, representing the original value it had. In other words, if an observation had 'Other' as the original value, the column '*VisitorType_Other*' will have a 1, and the other two columns will have a zero. As a result, from the 7 original categorical features (*Month*, *OperatingSystems*, *Browser*, *Region*, *TrafficType*, *VisitorType*, and *Weekend*), a total of 64 columns resulted after encoding the features into numerical values.

Bringing it all together

Now that the numerical features have been standardized, and the categorical features have been one-hot-encoded, these two data frames will be joined, along with the target variable, 'Revenue'. This is done by using the concatenation function from Pandas, `concat()`:

```
data = pd.concat([StdNumFeat, DumData, Target], axis=1)
```

The resulting data frame has now 73 columns with 12,330 observations.

4. Feature Engineering

To avoid longer training times, overfitting, and to better represent the problem to the predictive model, Recursive Feature Elimination will be performed to keep only the best subset of predictors. The aim of this is to obtain improved model accuracy and to *feed* the model with the most representative features that best describe the data.

After passing the data through the RFE algorithm, the resulting number of features to keep in the data set was reduced significantly to 36. This data was assigned to a variable called `'redFeat'`.

Based on the analysis from the previous section, the data is already fairly clean, but there are still quite some pre-processing activities that need to take place. Firstly, the data types need to match the variables that will be used. Therefore, the variables '*OperatingSystems*', '*Browser*', '*Region*', and '*TrafficType*' were coerced into *category* data type, by utilizing the `as.type()` function, as seen in Figure 7, below.

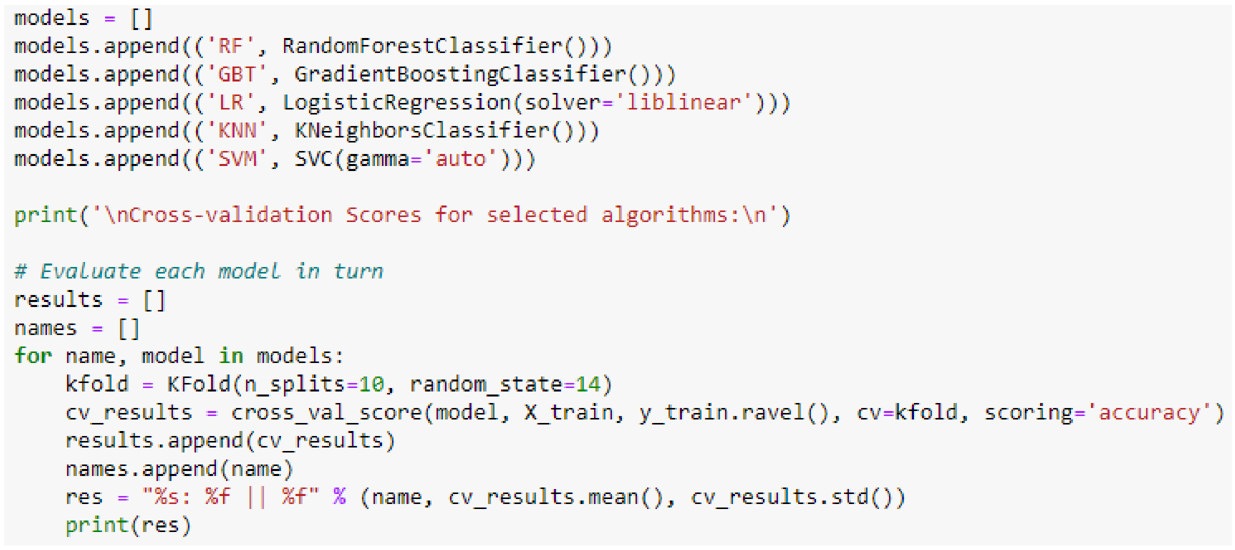
5. Building the Classifier Predictive Models

Now that the data has been pre-processed for the classifier algorithms, it is time to partition this data to train and evaluate the models. The first step is to partition the data into a training set, which will represent 70% of the data set, and then the testing data will be the remaining 30%. This is done using the `train_test_split()` function from Scikit-learn and specifying for the parameters: `test_size=0.3`.

The next step is to perform a 10-fold cross-validation to estimate the accuracy that each of the classifiers will have. The classifier algorithms that will be initially tested are:

1. Random Forest Classifier (RF)
2. Stochastic Gradient Boosting Trees (GBT)
3. Logistic Regression (LR)
4. K-Nearest Neighbors (kNN)
5. Support Vector Machine (SVM)

This was done by creating a variable called `models` and using an iterative process to append the results of the cross-validation for each of the classifier to this variable. See Figure 8



```
models = []
models.append(('RF', RandomForestClassifier()))
models.append(('GBT', GradientBoostingClassifier()))
models.append(('LR', LogisticRegression(solver='liblinear')))
models.append(('KNN', KNeighborsClassifier()))
models.append(('SVM', SVC(gamma='auto')))

print('\nCross-validation Scores for selected algorithms:\n')

# Evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10, random_state=14)
    cv_results = cross_val_score(model, X_train, y_train.ravel(), cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    res = "%s: %f || %f" % (name, cv_results.mean(), cv_results.std())
    print(res)
```

Figure 8. Classifier models pipeline.

The results for the cross-validation of these classifiers are:

Classifier	Accuracy (mean)	Std Dev. (mean)
Random Forest	0.890046	0.012103
Gradient Boosting Trees	0.900474	0.009844
Logistic Regression	0.885297	0.007008
k-Nearest Neighbors	0.888078	0.004187
Support Vector Machine	0.892133	0.006682

A graphical representation of these results can be observed in Figure 9 below. It is a boxplot where the variance and outliers of each method can be observed as well.

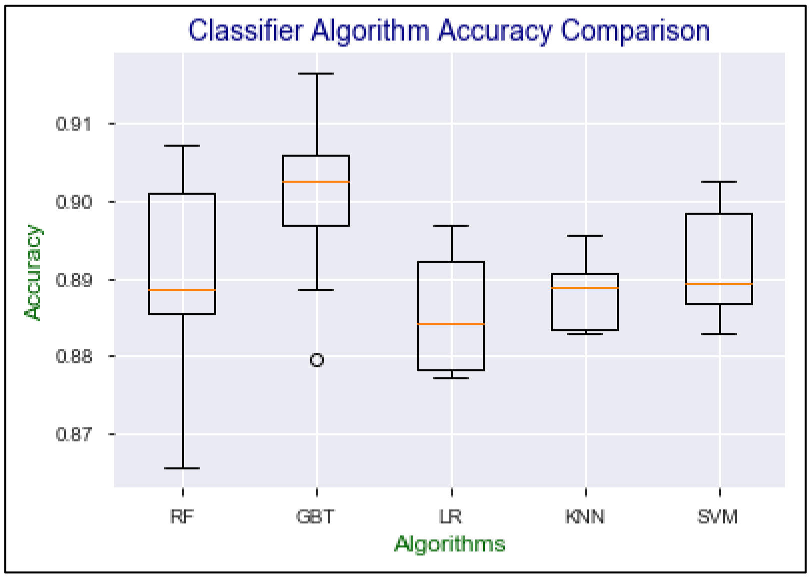


Figure 9. Cross-validation results boxplot chart.

Given these results, the top three classifiers in terms of accuracy are: GBT (0.90), SVM (0.89), and RF (0.89). Therefore, these three classifiers are selected to move forward with the model training process, and their hyperparameters will be tuned, in an attempt to increase their performance metrics.

6. Model Tuning

The model tuning was performed for each classifier, adding a time-keeper variable to measure how long does it take to train each of the models, as follows:

Stochastic Gradient Boosting Trees:

- A *tune grid* was used to try five different number of trees (50, 100, 150, 200, and 250), and three different learning rates (0.001, 0.01, and 0.1).
- The best accuracy was 0.902 using: `'learning_rate'`: 0.01, `'n_estimators'`: 250.
- The training time was ~179 seconds (around 3 minutes).

Random Forest Classifier:

- A *tune grid* was used to try 10 different number of trees (50, 100, 150, 200, 250, 300, 350, 400, 450, and 500), and two max features methods (`'auto', 'sqrt'`).
- The best accuracy was 0.899 using: `'max_features'`: 'auto', `'n_estimators'`: 300.
- The training time was ~315 seconds (around 5.25 minutes).

Support Vector Machine:

- A *tune grid* was used to try 5 different number of 'C' (1, 10, 50, 100, and 200), and two values for gamma (0.01, and 0.1).
- The best accuracy was 0.898 using: `'C'`: 200, `'gamma'`: 0.01.
- The training time was ~175 seconds (around 3 minutes).

A boxplot of these results can be observed In Figure 10 below:

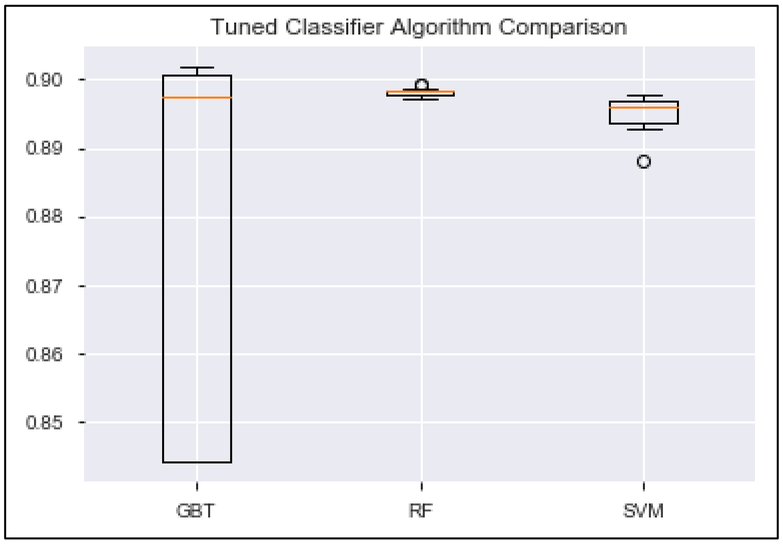


Figure 10. Boxplot for the tuned classifier results

Evaluating these results, it is noted that, although the highest accuracy score was obtained by the Gradient Boosting Trees algorithm, the variance in the values for GBT is the widest of the three classifiers. It ranges from just over ~0.84 to ~0.90, while the least variability is presented by the Random Forest classifier -so tight, that the quantiles show almost as a single line in the boxplot.

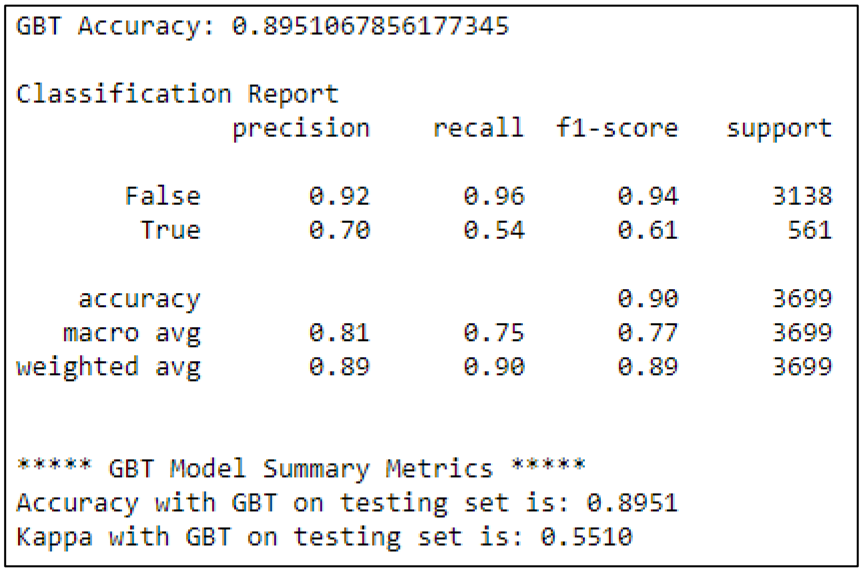
Since Random Forest had the second best accuracy score, but the variance in the accuracy values is so low, this will be the selected classifier, up to this point. These results will be reevaluated later after using the test set.

7. Model Evaluation (with tuned parameters)

Now, the models will be evaluated using the tuned parameters, but using the test set (the 30% of the data that is unseen at this point by the classifiers). The results were as follows:

Stochastic Gradient Boosting Trees:

- The tuned parameters are: `'learning_rate'`: 0.01, `'n_estimators'`: 250.
- The performance metrics are seen in Figure 11, below:



```
GBT Accuracy: 0.8951067856177345

Classification Report
              precision    recall  f1-score   support

       False       0.92      0.96      0.94      3138
        True       0.70      0.54      0.61       561

    accuracy                           0.90      3699
   macro avg       0.81      0.75      0.77      3699
weighted avg       0.89      0.90      0.89      3699


***** GBT Model Summary Metrics *****
Accuracy with GBT on testing set is: 0.8951
Kappa with GBT on testing set is: 0.5510
```

Figure 11. Tuned GBT performance metrics

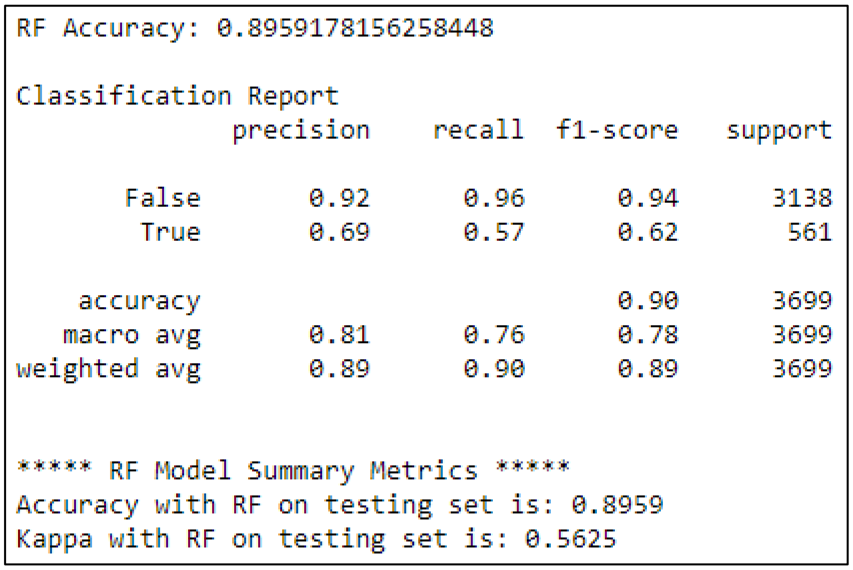
These performance metrics are obtained with the `classification_report()` function, which includes more performance metrics such as *precision*, *recall* (or *specificity*), the *f1 score*, and in the summary metrics, it includes the *Cohen's Kappa* value.

One important observation is that the *precision*, *recall*, and the *f1-score* metrics obtained for the 'True' predictions are considerably inferior than those of the 'False' predictions. This might be due to the high imbalance in the classes: 84.5% (10,422) are 'False', and the remaining 15.5% (1,908).

This results in the predominant class ('False') to have far more training examples, and therefore, it will generalize better to the 'False' resulting classes, than for the 'True' ones. This is true for all classifiers (GBT, RF, and SVM).

Random Forest Classifier:

- The tuned parameters are: `'max_features'`:'auto', `'n_estimators'`: 300.
- The performance metrics are seen in Figure 12, below:

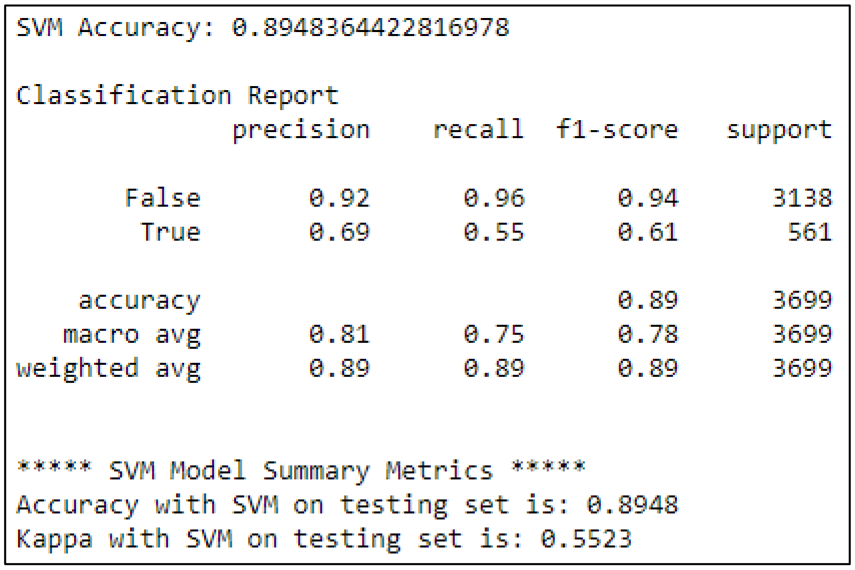


```
RF Accuracy: 0.8959178156258448

Classification Report
              precision    recall  f1-score   support

       False       0.92      0.96      0.94      3138
        True       0.69      0.57      0.62       561

    accuracy                           0.90      3699
   macro avg       0.81      0.76      0.78      3699
weighted avg       0.89      0.90      0.89      3699


***** RF Model Summary Metrics *****
Accuracy with RF on testing set is: 0.8959
Kappa with RF on testing set is: 0.5625
```

Figure 12. Tuned RF performance metrics

Support Vector Machine:

- The tuned parameters are: `'C'`: 200, `'gamma'`: 0.01.
- The performance metrics are seen in Figure 12, below:



```
SVM Accuracy: 0.8948364422816978

Classification Report
              precision    recall  f1-score   support

       False       0.92      0.96      0.94      3138
        True       0.69      0.55      0.61       561

    accuracy                           0.89      3699
   macro avg       0.81      0.75      0.78      3699
weighted avg       0.89      0.89      0.89      3699


***** SVM Model Summary Metrics *****
Accuracy with SVM on testing set is: 0.8948
Kappa with SVM on testing set is: 0.5523
```

Figure 13. Tuned SVM performance metrics

A summary table for the tuned classifiers performance metrics, using the testing (unseen) data is observed in Table 3.

Table 3. Tuned models performance metrics

Classifier:	**Gradient Boosting Trees**		**Random Forest**		**Support Vector Machine**	
Accuracy:	0.8951		0.8959		0.8948	
Kappa:	0.5510		0.5625		0.5523	
Precision:	True: 0.92	False: 0.70	True: 0.92	False: 0.69	True: 0.92	False: 0.69
Recall:	True: 0.96	False: 0.54	True: 0.96	False: 0.57	True: 0.96	False: 0.55
F1-score:	True: 0.94	False: 0.61	True: 0.94	False: 0.62	True: 0.94	False: 0.61

When analyzing the comparative results from Table 3 above, the best overall metrics are exhibited by the Random Forest classifier.

8. <u>Conclusions and Recommendations</u>

By tracking the users actions and session information data, a purchasing intention predictive model using the Random Forest classifier, applying RFE feature engineering, with properly tuned parameters, could predict with almost 90% certainty, which visitors will finalize their visit in a purchasing transaction, and therefore generating revenue for the e-tailer.

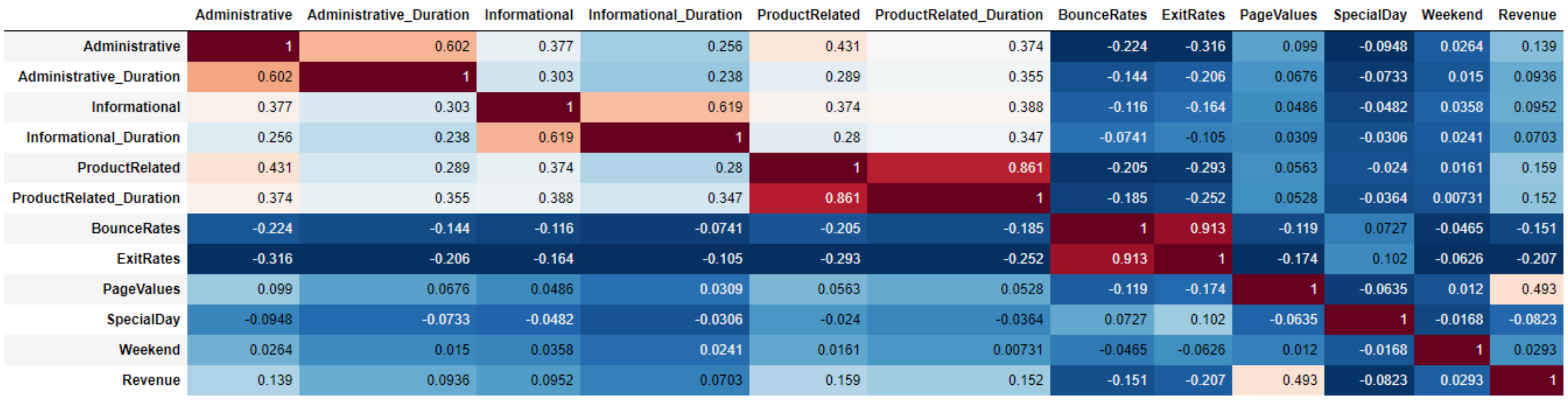
The performance metrics could be improved by tuning the parameters using a Randomized Search Grid method; however, this is a very computationally expensive process, and the use of it would depend on the level of accuracy demanded by the customer.

The Pandas Profiling module is an extremely useful and powerful tool to make Exploratory Data Analysis more efficiently, and with a single line of code. This saves time tremendously and provides a very thorough preliminary analysis of the data.

During the EDA, it was noted that returning visitors represent the highest traffic in the website, but the vast majority do not end up purchasing anything. This has tremendous potential and more data should be collected to explore this even further, to target marketing campaigns and event-based triggers to offer promotional content that would persuade the visitor to make a purchase.

A recommendation for future analyses, is to use oversampling techniques to address the issue of high class imbalance, such as Synthetic Minority Over-sampling Technique (SMOTE) and improve on the 'False' predictions' performance metrics.

ANNEX 1. CORRELATION MATRIX



	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues	SpecialDay	Weekend	Revenue
Administrative	1	0.602	0.377	0.256	0.431	0.374	-0.224	-0.316	0.099	-0.0948	0.0264	0.139
Administrative_Duration	0.602	1	0.303	0.238	0.289	0.355	-0.144	-0.206	0.0676	-0.0733	0.015	0.0936
Informational	0.377	0.303	1	0.619	0.374	0.388	-0.116	-0.164	0.0486	-0.0482	0.0358	0.0952
Informational_Duration	0.256	0.238	0.619	1	0.28	0.347	-0.0741	-0.105	0.0309	-0.0306	0.0241	0.0703
ProductRelated	0.431	0.289	0.374	0.28	1	0.861	-0.205	-0.293	0.0563	-0.024	0.0161	0.159
ProductRelated_Duration	0.374	0.355	0.388	0.347	0.861	1	-0.185	-0.252	0.0528	-0.0364	0.00731	0.152
BounceRates	-0.224	-0.144	-0.116	-0.0741	-0.205	-0.185	1	0.913	-0.119	0.0727	-0.0465	-0.151
ExitRates	-0.316	-0.206	-0.164	-0.105	-0.293	-0.252	0.913	1	-0.174	0.102	-0.0626	-0.207
PageValues	0.099	0.0676	0.0486	0.0309	0.0563	0.0528	-0.119	-0.174	1	-0.0635	0.012	0.493
SpecialDay	-0.0948	-0.0733	-0.0482	-0.0306	-0.024	-0.0364	0.0727	0.102	-0.0635	1	-0.0168	-0.0823
Weekend	0.0264	0.015	0.0358	0.0241	0.0161	0.00731	-0.0465	-0.0626	0.012	-0.0168	1	0.0293
Revenue	0.139	0.0936	0.0952	0.0703	0.159	0.152	-0.151	-0.207	0.493	-0.0823	0.0293	1