Experimentation and Prediction of Online Shopper's purchasing Intention using decision tree and keras (neural network)

By Dayo Samuel, Course Work 2022

Abstract

In this research, I propose a system that can predict the shopping intent of online shoppers in real-time as they navigate through different pages on an e-commerce website. To achieve this, I use data on the types of pages visited by returning and new shoppers, as well as session information for some shoppers. These features are then input into a decision tree classifier, which I also optimize using oversampling techniques to improve its accuracy and scalability with neural network. My findings indicate that the decision tree classifier is the most accurate algorithm for this task, based on its accuracy and F1 score compared to other methods tested.

Keywords: Online shopper Intention , Decision Tree, Experimentation Design, User navigation

1.0 Introduction

The advancement of technology and the shift towards a knowledge-based society has led to an increase in the popularity of e-commerce globally. Many businesses have transitioned from physical stores to online platforms, while others have maintained both types of shopping environments. It has been demonstrated that a significant number of consumers prefer the convenience of virtual shopping, which has led to increased competition among online retailers, similar to the competition that exists in physical shopping environments. The growing prevalence of virtual environments has prompted companies and e-commerce businesses to invest in systems and products that can accurately identify potential visitors and those who are losing interest as they browse websites. The goal of this is to prevent the risk of losing users who may navigate away from the site in search of a similar store. This is similar to a trained salesperson who works to retain potential customers by offering a range of userdefined marketing actions that are likely to motivate and increase the likelihood of completing one or more transactions.

In a study, Ganguly et al. (2010) examined the relationship between website design, online shopping behaviour, and trust in online transactions. They explored how trust mediates the connection between website design and purchase intention, and also how culture moderates this relationship. The goal of the study was to provide evidence on how website design can affect purchase intention and perceived risk through trust in the online store. The results of the study provide insights into the factors that contribute to trust in online shopping for individuals of different cultural values, as well as how web design elements can foster trust in online shopping

Another study, Michael et al. (2007) conducted a study to understand the factors that influence consumers' choices to make purchases online or through other channels. Previous research on this topic has employed various approaches, including behavioural economics, lifestyle analysis, and merchandising effects, and has identified the potential influence of personality traits as well as factors such as time, costs, benefits, and shopping context. In their study, Michael et al. used a hierarchical

approach to personality and data from an online consumer panel to develop a model of personality that could predict consumers' intentions to purchase products and services online

Sakar et al. (2019) introduced a real-time online shopper behaviour system that comprises of two modules. The first module utilizes a multi-layer perceptron model that employs deep learning to predict user behaviour and applies oversampling during the training phase to enhance the model's performance. Ultimately, an accuracy of 85% was achieved. The second module trains a long shortterm memory-based recurrent neural network, which addresses the issue of incorporating the time dimension, using only clickstream data instead of pageview data and some sessions used in the first module.

Baati et al. (2020) proposed a real-time shopper behaviour prediction system that predicts a visitor's shopping intent as soon as they land on the website. This study leverages session and visitor information and evaluates naive Bayes, C4.5 decision tree, and random forest classifiers, using oversampling to improve the performance and scalability of each classifier. The random forest method exhibited the highest accuracy and F1 score compared to the other techniques used.

Karim Baati (2021) advanced his study by implementing a hybridization of adaboost with random forest. Similar to the previous study, the proposed system relies on session and user information and employs oversampling to improve classification performance. The results demonstrate that the system based on adaboost and random forest ensemble classification outperforms the previous system in terms of accuracy and F1 score.

Sang and Wu (2022) proposed a real-time online shopper prediction system that predicts a visitor's shopping intent as soon as the website is visited. The authors designed the system to retain the greatest number of potential customers by creating a strategy.

Qiang and Chen (2015) introduced the idea of unsupervised learning to solve prediction problem of visitors purchase intention. They use an unsupervised learning clustering algorithm led by K-means which clusters visitor is according to their browsing behaviour and individual characteristics and further distinguish them into classes with high purchase intention and low purchase intentions for various products. Hence according to the users' lands on the site, they can easily be grouped into high purchase intention or users with low purchase intention, to provide guidance to merchants.

All these predictions, models have done well in its respective experiments, interestingly, a lot of info set has been created that support analytics trailing code we have Google analytics, and other system that uses these same techniques however there are still more problems. Since the dataset is quite new and many researchers have focused on solving and many on improvement.

1.1 Datasets

The Data used for the experimentation and predicting online shopper intention is gotten from the UCI (UCI Machine Learning Repository: Online Shoppers Purchasing Intention Dataset Data Set), which was credited to Sakar and Kastro as the source. The dataset consists of feature vector belonging to 12,330 sessions and there are 10 numerical and 8 categorical attributes. The "Revenue" attribute would be used as the class label.

The dataset consists of 18 variables and is divided into train and test sets. The training data is used to build the model, while the test data is used to evaluate the model's performance. According to Sakar et al. (2018), the dataset includes the following variables: "Administrative," "Administrative Duration," "Informational," "Informational Duration," "Product Related," and "Product Related Duration," which represent the number of distinct types of pages visited by the visitor in that session and the total time spent in each of these page categories. These values are derived from the URL information of the pages visited by the user and updated in real time as the user navigates from one page to another. The "Bounce Rate," "Exit Rate," and "Page Value" features represent metrics measured by "Google Analytics" for each page on the e-commerce site. The "Bounce Rate" for a web page refers to the percentage of visitors who enter the site from that page and then leave ("bounce") without triggering any other requests to the analytics server during that session. The "Exit Rate" for a specific web page is calculated as the percentage of pageviews that were the last in the session. The "Page Value" represents the average value of a web page visited by a user before completing an ecommerce transaction. The "Special Day" feature indicates the proximity of the visit to a specific special day (e.g., Mother's Day, Valentine's Day) when sessions are more likely to result in a transaction. This value is determined based on the dynamics of e-commerce, such as the duration between the order date and delivery date. For example, for Valentine's Day, this value takes a non-zero value between February 2 and February 12, is zero before and after this date unless it is close to another special day and has a maximum value of 1 on February 8. The dataset also includes the operating system, browser, region, traffic type, visitor type as returning or new, a Boolean value indicating whether the date of the visit is a weekend, and the month of the year.

1.2 Explanation and preparation of datasets

In this section, I provide a brief overview of the dataset, the work conducted, and the model used to classify the dataset. Similar to Sakar et al. (2018), I utilize the same dataset. However, I only consider the part of the dataset related to page visits, session, and user information. The dataset consists of 12330 sessions, of which 10551 were returning users, 1779 were new users or other users that could not be tracked (considered as null values), and 10422 were negative class samples that did not complete a transaction. The remaining 1908 were positive class samples that did complete a transaction.

A widespread problem that cannot be overlooked with my dataset is the imbalance samples where over 80% of the samples were negative sample of incomplete transaction and less than 20% are positive samples of complete transaction. For features used for system validation as shown in Table 1, it is observed that some variables are continuous and some discrete. Some are numerical and some are categorical. This requires different data processing method, and I used a binning algorithm to discretize continuous variables.

For this task, the challenging part is building a proper system learning technique that would well normalize the input ab d get good prediction and experimentation. First is I need to find a good normalization standard and then choose the suitable model since my task is classification, I consider a decision tree model, and to tackle the imbalance I used the Keras neural network algorithm using smote to fix the imbalance problem.

Table 1 *Features used for system validation*

Feature name	*Feature description*	*Feature type*
Administrative	Number of pages visited by the visitor about account management	Numerical
Administrative duration	Total amount of time (in seconds) spent by the visitor on account management related pages	Numerical
Informational	Number of pages visited by the visitor about Web site, communication, and address information of the shopping site	Numerical
Informational duration	Total amount of time (in seconds) spent by the visitor on informational pages	Numerical
Product related	Number of pages visited by visitor about product related pages	Numerical
Product related duration	Total amount of time (in seconds) spent by the visitor on product related pages	Numerical
Bounce rate	Average bounce rate value of the pages visited by the visitor	Numerical
Exit rate	Average exit rate value of the pages visited by the visitor	Numerical
Page value	Average page value of the pages visited by the visitor	Numerical
Operating Systems	Operating system of the visitor	Categorical
Browser	Browser of the visitor	Categorical
Visitor Type	Visitor type as ‘‘New Visitor,’’ ‘‘Returning Visitor,’’ and ‘‘Other’’	Categorical
Revenue	Class label indicating whether the visit has been finalized with a transaction	Categorical

I explore some of the important patterns in the dataset in the next section. In my dataset visitor type has categorical value, I converted them to numerical values replacing them with numbers.

1.2.1 Data-mining techniques

This section consists of some important pattern and other valuable information from my dataset. I imported all the required libraries into my jupyter notebook before loading my dataset. Using seaborn, I did explore the following patterns and got some insight into user’s behaviour on the website. The average value for the web page that the users visited before completing a transaction is shown in figure 1. While exploring the dataset further, it was observed the completed transaction based on page value were not so significant in figure 2. The rate at which visitor’s exit the website does not necessarily mean a completed transaction, in this case, the exit rate tends to contribute negatively to the completion as seen in figure 3. Similarly, the bounce rate shows similar trend, figure 4.

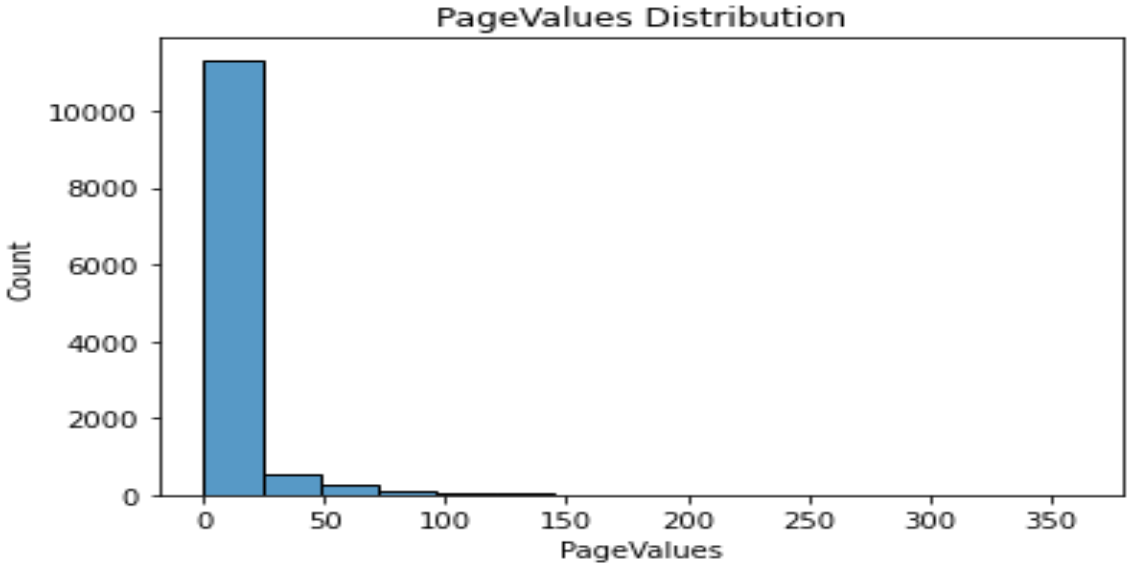


Figure 1. PageValues Distribution in Online Shopper's Intention Dataset

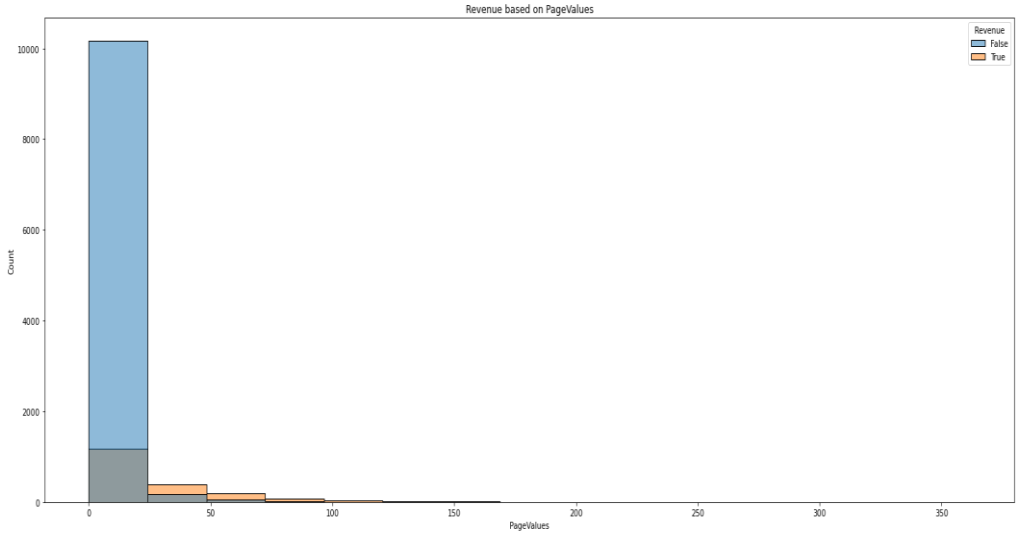


Figure 2. Revenue based on PageValues

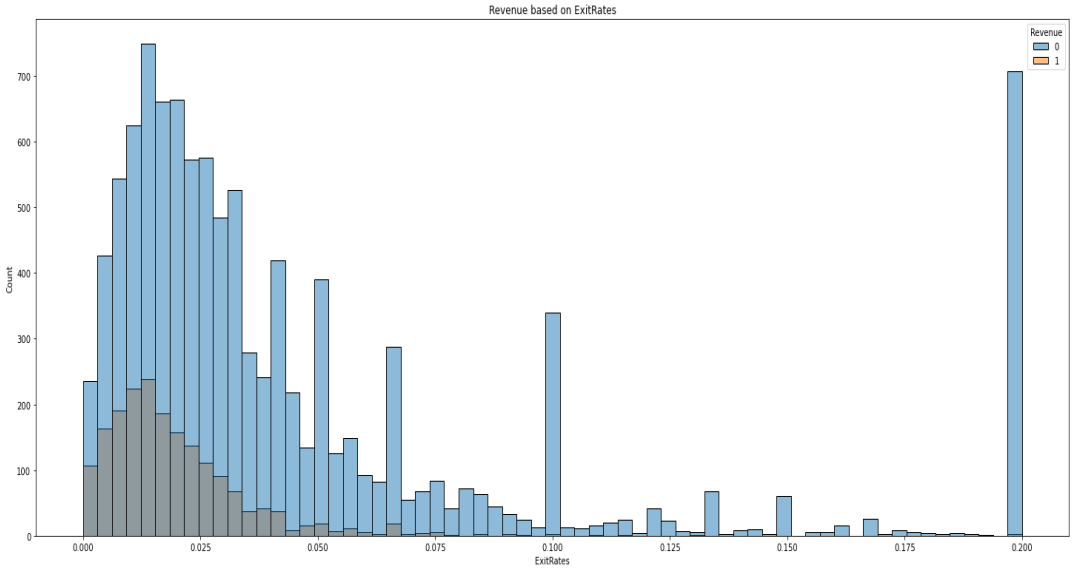


Figure 3. Revenue based on ExitRates

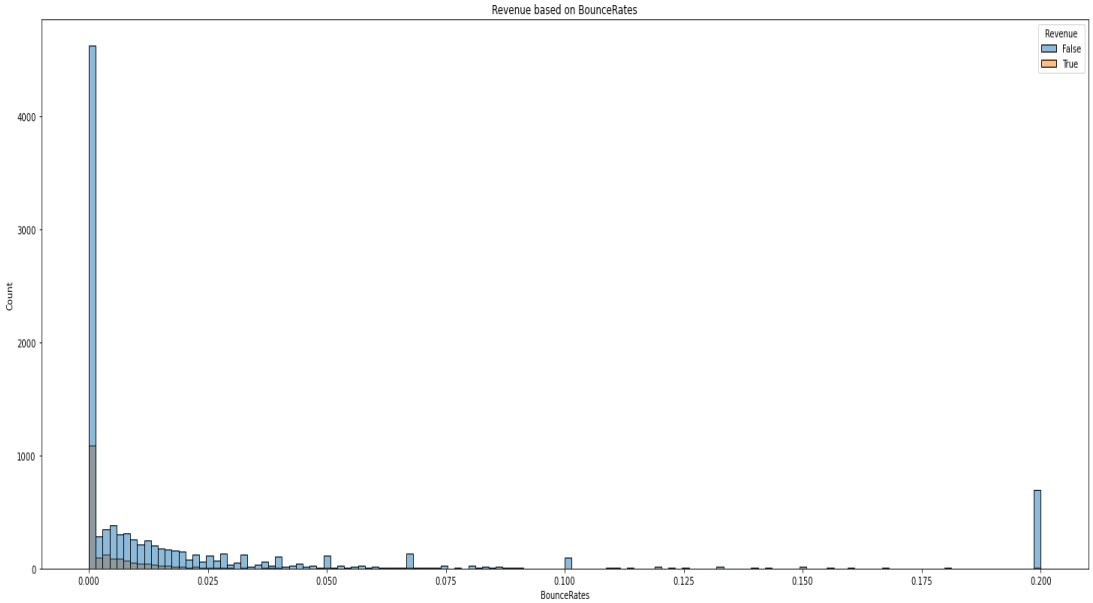


Figure 4. Revenue based on BounceRates

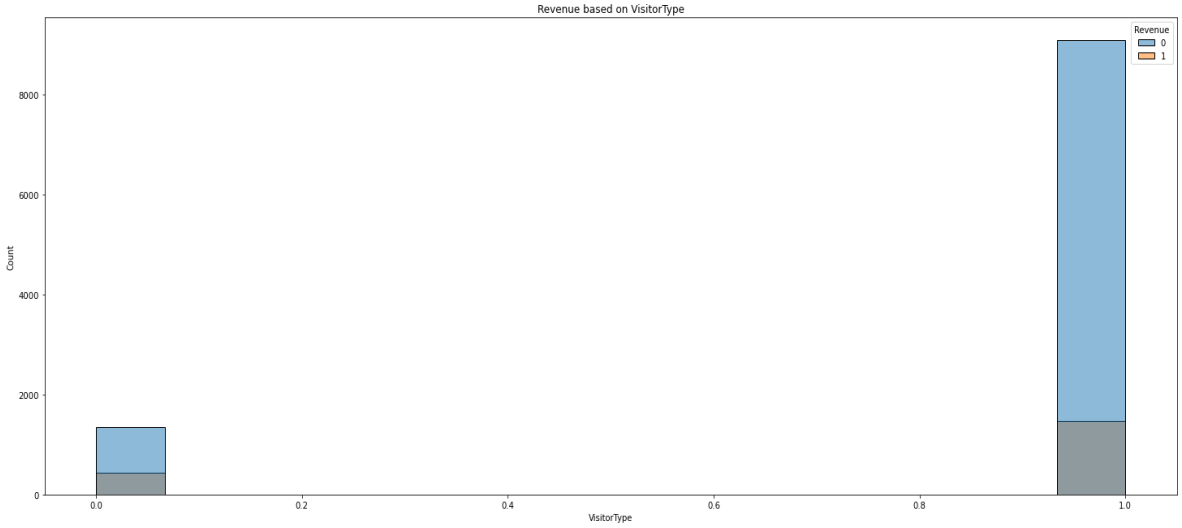


Figure 5. Revenue based on VisitorType

1.3 Classification Methodology

In data analysis, classification is a technique used to predict the class or category to which a given data point belongs. This is often done using machine learning algorithms, which are trained on a labelled dataset where the correct class for each data point is known. The goal of the classification process is to use the patterns learned from the training data to accurately predict the class of new, unseen data. Classification algorithms can be used in a wide range of applications, including image and speech recognition, fraud detection, and email filtering. There are many different classification algorithms available, including decision trees, k-nearest neighbors, support vector machines, and artificial neural network.

1.3.1 Decision Tree

This classification algorithm likewise called random forest, in many cases of similar studies, RF have performed well compared to KNN and other classification techniques. K-NN is not suitable for real time prediction since it is a lazy learning algorithm (Sakar et al 2018). Hence it would not be used here. Decision tree is an effective method that would be used in this classification. The advantage of this classification algorithm is its ability to feeding different feature subsets and decision rules at various stages of classification. A general decision tree consists of one root node, several internal and leaf nodes, and branches. Cheng-Jin (2008). There is different type of decision tree algorithms, Hunt's Algorithm which is one of the earliest, ID3, C4.5, CART, SLIQ, SPRINT and so on. The basic algorithm is constructed in a top-down recursive manner and at start, all the training examples are at the root. Attributes are often categorical, however, if continuous-valued, they are discretized in advance. Examples are partitioned recursively based on selected attributes. Attributes are selected based on heuristic or statistical measure.

1.3.2 Neural Network

Neural networks are a powerful class of machine learning models that have proven successful in tackling a wide range of classification and regression problems. These models are inspired by the structure of neurons in the brain and nervous system and consist of interconnected nodes that interact in complex networks. One type of neural network, called a recurrent neural network (RNN), is particularly well-suited for processing sequential data. RNNs have the ability to store relevant information from past inputs, allowing them to consider this context when predicting future outputs. For example, when training an RNN on a language modelling task in which it generates text one character at a time, it is useful to store the characters that have been predicted in previous time steps, as the next character is dependent on these previous predictions.

1.3.3 Experimentation and Prediction using Decision Tree

The following are the steps in the Decision tree or random forest construction algorithm:

1. From the original training set of N instances, create a subsample using bagging, which involves randomly selecting N instances with replacement.

2. Select m input variables at random from the M variables for each instance, where m is much less than M. The best split on these m variables is used to split the node.

3. Allow each tree to grow to the maximum extent possible without pruning, based on the predetermined stopping criteria.

4. Repeat this process until the desired number of trees is obtained for the forest."

Determining the class feature and input features:

In this study, the goal is to use a classification model to predict the value of the "Revenue" feature, which is the dependent variable or class label, based on the other features listed in Table 1, which are the independent variables or input features. To prepare the data for this classification task, I divided it into input and output data and removed any variables that were not necessary for this classification. The aim is to use the input features to accurately predict the value of the class label, which is the Revenue feature.

Splitting the dataset into the training set and the test set:

To evaluate the performance of the classification model, I divided the data into a training set and a test set. The train_test_split() function was used to split the data, with the test set comprising 30% of the data and the random state parameter set to 0 to ensure that the same training and test sets are obtained across multiple runs. The random state hyperparameter controls the shuffling process during the data split, ensuring that the data is randomly shuffled before being divided into the training and test sets. This allows for a fair evaluation of the model's performance on unseen data.

Scaling features:

To ensure that all of the input features are on the same scale, I applied feature scaling to the training data. This was done by subtracting the mean of each feature from the respective feature values and then dividing by the standard deviation of the feature. The same mean and standard deviation values were used to scale the test data. To implement this scaling, I used the fit_transform() method on the training data to learn the scaling parameters and apply the scaling. For the test data, I used the transform() method, which applies the scaling using the same mean and standard deviation values that were learned from the training data. This ensures that both the training and test data are scaled consistently and allows for a fair evaluation of the model's performance.

Training and evaluating the model:

To build the classification model, I used a decision tree algorithm on the training data. I then applied the trained model to the test data using the predict function to generate predictions. To evaluate the performance of the model, I used various metrics from the scikit-learn library. These metrics allowed me to quantify the accuracy of the model and understand how well it was able to predict the class label (Revenue) based on the input features. By comparing the predicted values to the actual values in the test set, I was able to assess the model's performance and identify any areas where it may be lacking

Performance evaluation

To thoroughly evaluate the performance of the decision tree model, I calculated several different metrics for each repetition of the experiment. The accuracy of the model was assessed, which measures the proportion of correct predictions made by the model. The precision of the model was also calculated, which reflects the proportion of true positive predictions relative to all positive predictions made by the model. The recall of the model was determined, which reflects the proportion of true positive predictions relative to all actual positive cases. The F1 score was also calculated, which is a weighted average of the precision and recall and provides an overall measure of the performance of the model. By analyzing these various metrics, I was able to get a comprehensive understanding of the effectiveness of the decision tree model in predicting the class label (Revenue) based on the input features.

Visualisation of the results.

In the previous section, I used a confusion matrix to evaluate the performance of the decision tree model. A confusion matrix is a table that displays the number of true positive, true negative, false positive, and false negative predictions made by the model. To visualize the confusion matrix, I used the seaborn heatmap function. The heatmap allows for an easy and intuitive way to interpret the results of the confusion matrix, providing a visual representation of the model's performance. By examining the heatmap, I was able to understand the model's ability to correctly classify instances as positive or negative and identify any areas where the model may be struggling.

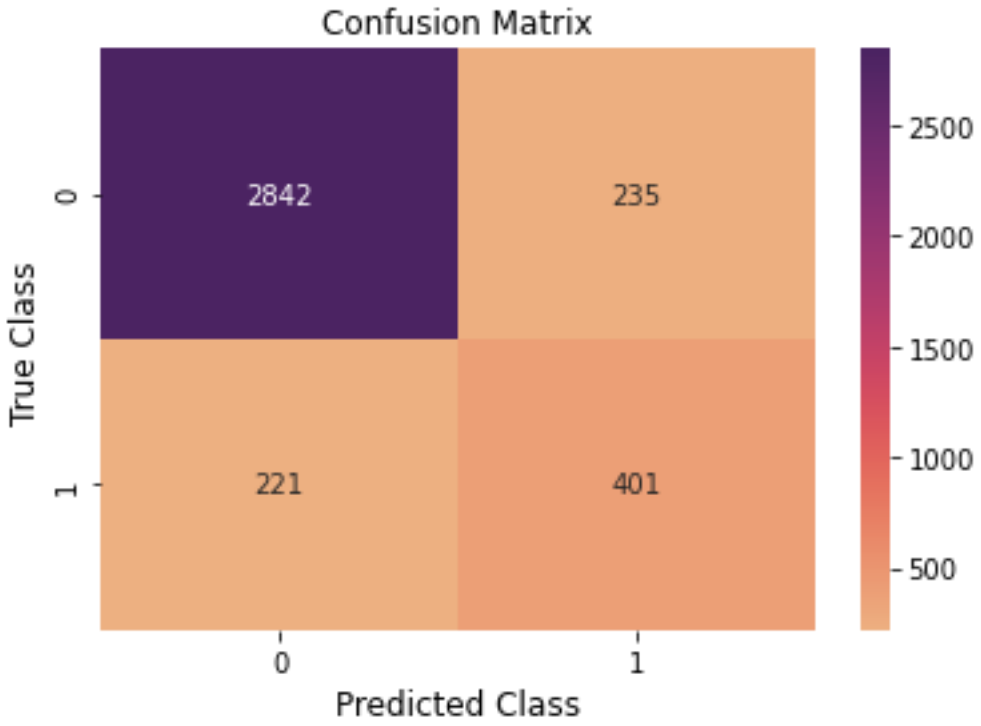


Figure 6. Visualisation of the result from decision tree

1.3.4 Experimentation and Prediction using Neural Network with Keras

TensorFlow is a powerful library for building and training deep neural networks. In this case, I used TensorFlow in conjunction with Keras, which provides a more intuitive interface for building and training neural network models. The same input variables and class label as used in the previous classification method were used in this case. The steps involved in implementing the Keras neural network algorithm are outlined below:

1. Define the model architecture: This involves specifying the number and size of the layers in the network, as well as the activation functions and any other hyperparameters.
2. Compile the model: This involves specifying the optimizer and loss function to be used during training.
3. Fit the model to the training data: This involves using the fit() method to train the model on the training data.
4. Evaluate the model on the test data: This involves using the evaluate() method to measure the model's performance on the test data.
5. Make predictions on new data: This involves using the predict() method to generate predictions on unseen data.

Before implementing the neural network model, I imported all of the necessary libraries, including numpy, pandas, tensorflow, seaborn, and StandardScaler. I used a seaborn count plot to visualize the class label and noticed that there was an imbalanced class, with 80% of the observations belonging to the

"incomplete transaction" category. To address this issue, I implemented techniques such as oversampling to balance the class distribution in the training data. This is important because imbalanced classes can lead to biased models that perform poorly on the minority class. By balancing the classes, I was able to improve the model's performance and ensure that it was able to accurately predict the class label for all classes

Normalising the data:

This is important when training a neural network do avoid delays or failures in convergence during training. I used StandardScaler estimator from ScikitLearn to scale my data so that the mean of each variable is 0 and the standard deviation is 1. Since Keras assumes my class label to start from 0 and my class labels are 0 and 1 which mean Incomplete transactions and complete transactions, respectively. Hence my data is normal. I dropped the month, Specialday, Region, TrafficType and weekend since these would not be used in my model as in Table 1. Except “Revenue” which is my output.

Addressing class imbalance:

In the process of solving the problem of imbalanced samples in my dataset, I chose oversampling using the Imbalanced Learn library. Using oversampling, this takes the minority class and randomly duplicates occurrences until it matches the frequency of the most commonly occurring target class. Using RandomOverSampler, I conducted random oversampling, and I used count plot to visualise the result.

Building and training my neural network:

With Keras sequential model which allows me to build my neural network layer by layer, I did this by instantiating the model, then using the ‘add’ method to add layers to the model, in a simple feedforward network, I used ‘Dense’ layer which implies each neuron in the layer is connected to each neuron in the previous layer. For each layer, I specify the numbers of neurons it is going to contain, and the activation function I used for the layer. For the first layer, I define my input shape and set it to 12 as there are 12 input variables in my data.

In the final layer, I used two classes (Incomplete transaction and Complete transaction), so I have two neurons in my final layer.

Evaluating the model

After compiling the model, and have used optimizer and a loss function, I used summary () to view the neural network I have built. And after training the model in Scikit Learn using fit method, upon completing the training, I plot the training and validation accuracy.

Visualisation of the results.

The previous section shows the confusion matrix, I there used the seaborn heatmap to visualise the confusion matrix as shown in figure 7 for the class imbalanced and with oversampling respectively.

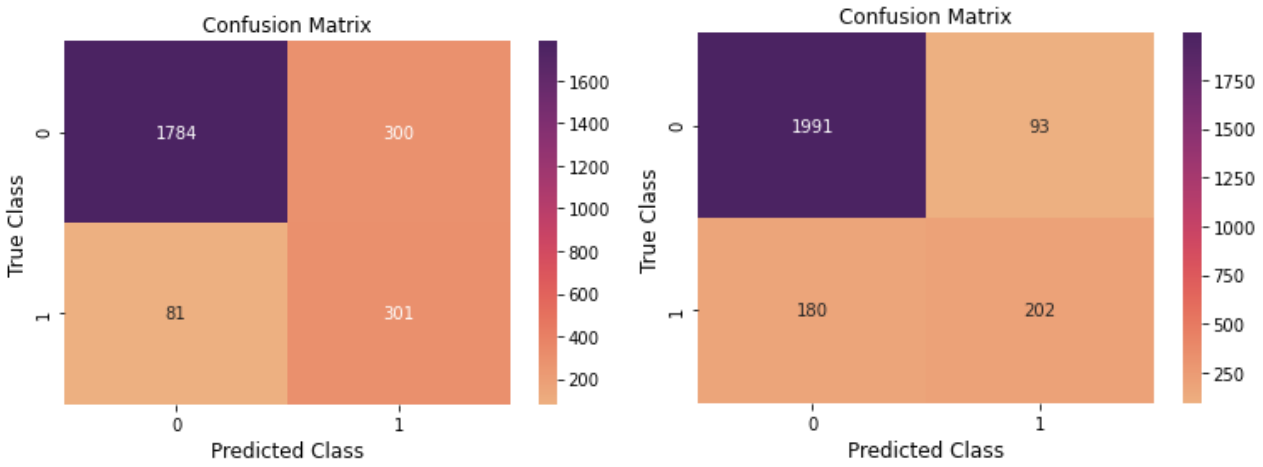


Figure 7. Visualisation of the result from neural networks

1.4 Results analysis and discussion Part 1

Before the data is pre-processed, univariate and muti-variate analysis are performed to discover some important patterns and it is very vital because in machine learning the importance of finding out imbalance samples cannot be overemphasized as they may influence the results. In other to perform specific experiments, I divided the data set into train and test set. 70% as the training and 30% as the test set. The following table shows the performance results of my model on my selected dataset Namely accuracy, and F1 score Performance results on Decision tree, Balanced and Unbalanced dataset, respectively. In the process of dealing with imbalanced samples, I have chosen oversampling. Figure 8 shows example of a univariate analysis and the result of oversampling performance on the class imbalance.

Table 2 shows the results obtained from Decision tree algorithm, class imbalance dataset with class weight and with oversampling. The results shows that decision tree algorithm gives the best accuracy rate on the test set. However, the imbalanced dataset tends to show an equal F1 score with decision tree's score, this cannot be trusted as the accuracy validation was quite inconsistent as in Figure 10. The result from the oversampling shows a stable validation accuracy close to the result obtained from decision tree.

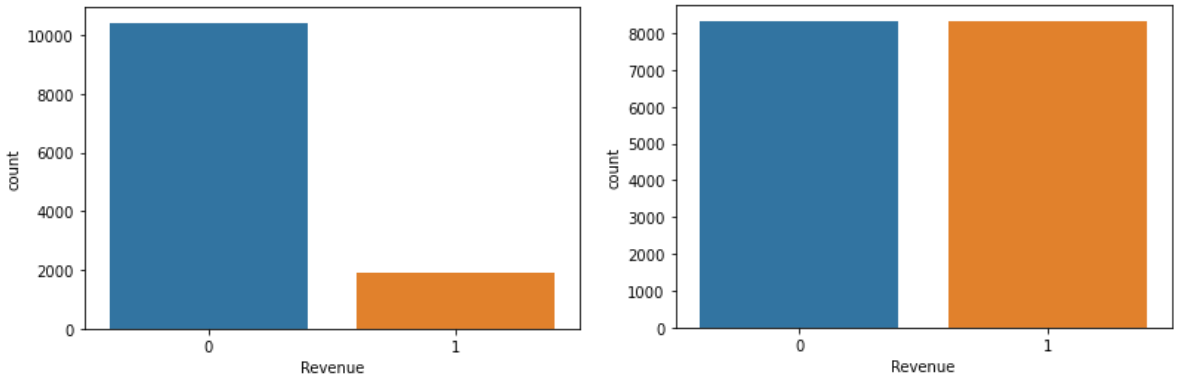


Figure 8. Univariate and Imbalance Analysis

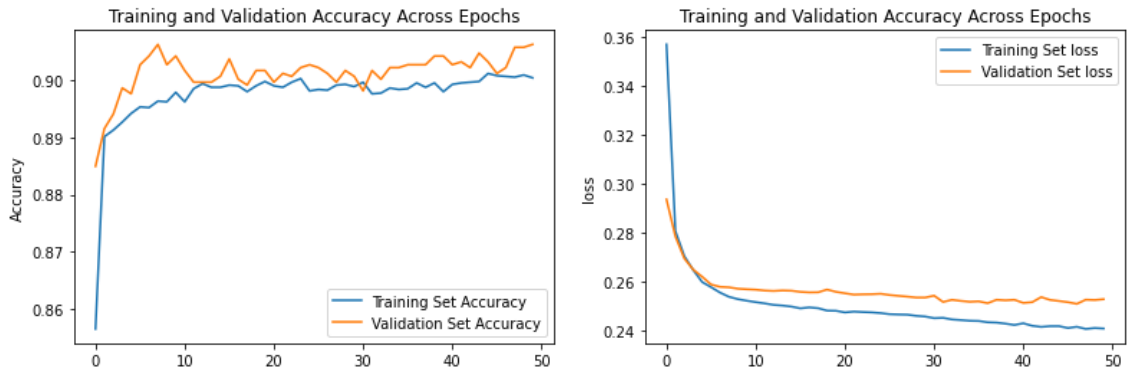


Figure 9. Accuracy and Loss with oversampling

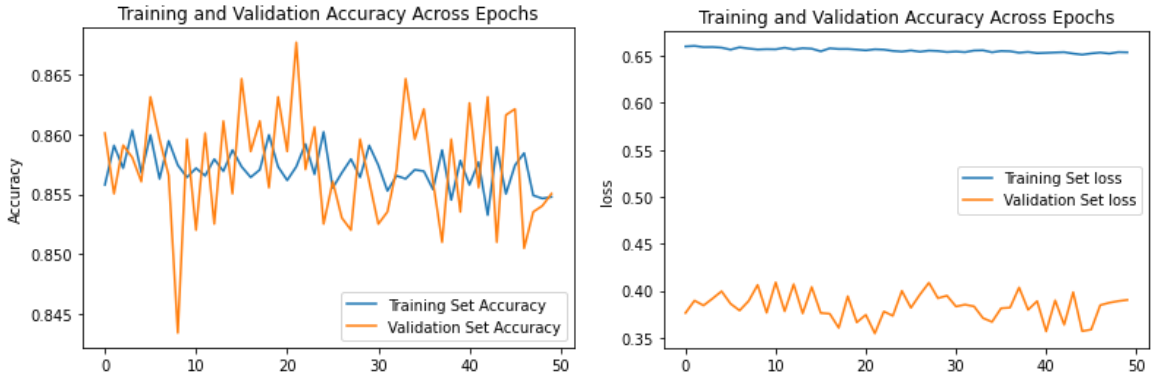


Figure 10. Accuracy and Loss with imbalance samples

Table 2. Results obtained on Decision tree, Imbalanced dataset & oversampling

Data	Accuracy	F1 score
Decision Tree	0.88	0.78
Imbalanced	0.85	0.77
Oversampling	0.88	0.75

1.5 Experimentation and Prediction using Azure Machine Learning Designer

Azure Machine Learning Designer is a user-friendly tool that enables individuals to construct, test, and deploy machine learning models without the need to write code. It is part of the Azure Machine Learning suite, which offers a cloud-based platform for creating and deploying machine learning models. The Designer allows users to easily explore various algorithms, process and transform data, and assess the performance of their models. It also integrates with popular open-source libraries such as TensorFlow and scikit-learn, allowing users to utilize their existing skills and code when working with the Designer. The visual, drag-and-drop interface makes it easy for users to build and deploy machine learning models without the need for extensive coding knowledge. In Azure Machine Learning Designer, we can create multiple neural network models and compare their performance. To do this, we can use the "Neural Network" module, which provides a variety of pre-configured neural network

architectures, such as feedforward, convolutional, and recurrent networks. We can also customize the architecture of our network by adding, removing, or modifying the layers.

Once we have created our neural network models, we can use the "Split Data" module to divide our dataset into training and testing sets. Then, we can use the "Train Model" module to train each of our models on the training data. Finally, we can use the "Evaluate Model" module to compare the performance of our models on the testing data. This will allow us to see which model performs best on our dataset. The steps of the Azure Machine Learning Designer and two Neural Network algorithm used after filtering the needed variables and loading by data into the canvas are detailed below:

Filtering Data Needed:

Based on the objective and the variables selected for this experiment, it was important to filter and remove the variables that were afore assumed to be irrelevant for this specific work, as shown in the screenshots (Appendix 2), the following variables were exclude from the dataset been loaded into the canva: SpecialDay, Month, TrafficType and Weekend.

Adding Transformations:

In this step pre-processing transformation is done by using the Normalize Data under Azure Machine Designer's component. I placed the Normalize data on the canva and connect the output from the bottom of the loaded dataset "Onlineshopper" to the input at the top of the Normalize data module. Setting the transformation method to MinMax And using 0 for constant columns when checked to True. This helps to normalise the numeric columns to put them on the same scale, which helps to prevent columns with large values from dominating model training.

Adding Training Modules:

Here as it is a common practice to train the model using a subset of my data, whiling holding back some data with which to test the train model. Using the following modules, Split Data module and configured the setting. Splitting mode was set to split rows and Fraction of the rows in the first output dataset was set at first to 0.7 and later set to 0.2 for an experiment. Randomized split was set to True and random seed was set to 123 stratified split was false. Two Class Neural Network module was added to the canva, and split data module output was connected to the untrained model input of the train model module. In other to test the train model, scoring the validation dataset was required hence using score model from the component.

Evaluating the Model:

Using the evaluate model module, where the score model is placed and connecting the output of the score model module to the scored dataset input of the evaluate model module as shown in Appendix 2.

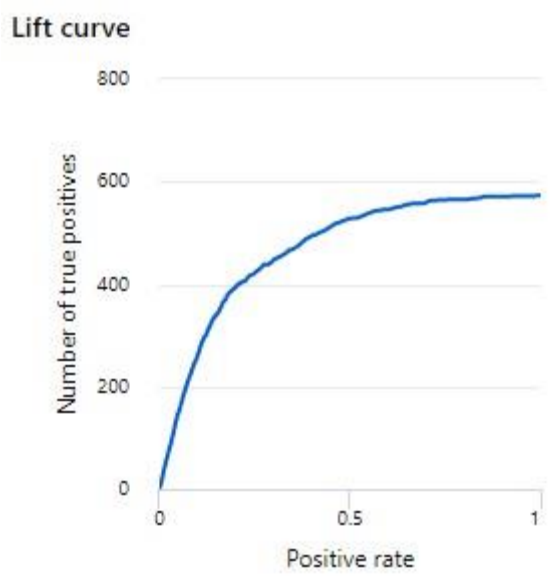
1.6 Results analysis and discussion Part 2

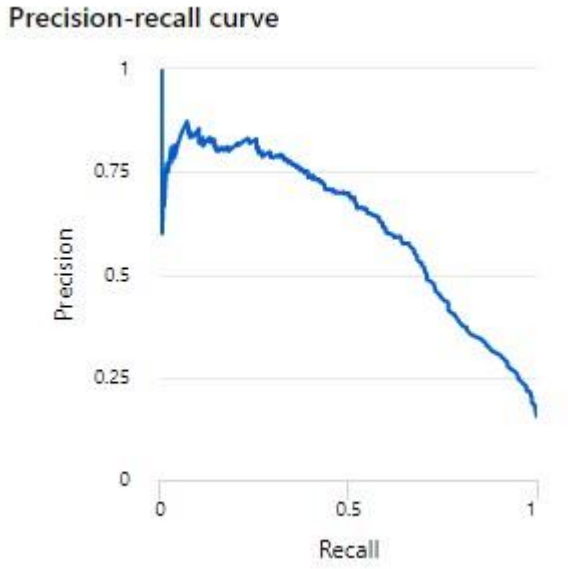
In Table 3, I present the results of using the Azure machine learning designer with a neural network classifier on an imbalanced dataset, as well as with oversampling using SMOTE. The results show that there is only a slight or insignificant difference between the accuracy and precision when they are approximated. Therefore, it is recommended to use alternative metrics that take into account the class imbalance, such as the F1 Score, to evaluate the performance of the classifier. In this study, the results are reported together with the general accuracy rate as well as the individual class accuracies and F1 Score, as shown in table 3. Two-Class Neural Network with oversampling achieved the highest F1 Score

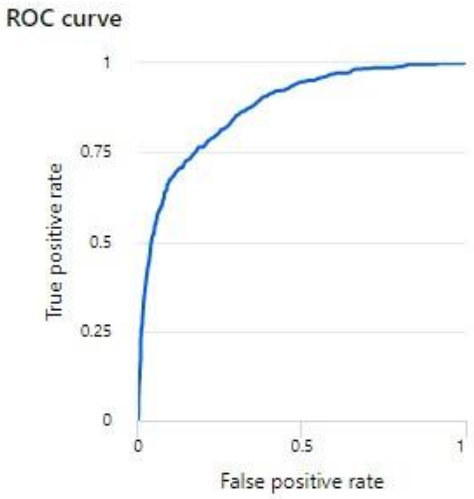
of 0.6, while class imbalance resulted in a lower F1 Score of 0.463. Oversampling equalizes the number of samples in the positive and negative classes, allowing both accuracy and F1 Score to be used as evaluation metrics. As shown, oversampling resulted in an accuracy of 88.8% and an F1 Score of 0.6

Table 3. Results obtained on Imbalanced dataset & oversampling using Azure

	Accuracy	**Precision**	**Recall**	**F1 Score**
Class Imbalance	0.882	0.772	0.33	0.46
Oversampling(smote)	0.888	0.778	0.59	0.66







Conclusions

This study presents a system designed to assist online shopping web analysts and owners in identifying potential customers through the use of algorithms such as decision trees and neural networks, implemented using Python and Azure Machine Learning Designer. The decision tree algorithm yielded the most promising results for this system. My work aims to forecast the real-time

purchase intentions of online customers as they navigate from different pages on the website and within the site funnel. In other words, customers make decisions about whether to purchase as they move through the site. The decision tree classifier shows high performance and the results of the model evaluation demonstrate its usefulness. There are several areas for future work that could be explored, such as using different algorithms that were not employed in this study and considering different pages on the site that were not included in the current analysis. These avenues of investigation could provide additional insights and further improve the effectiveness of the system.

References

Baati, Karim & Mohsil, Mouad. (2020). Real-Time Prediction of Online Shoppers' Purchasing Intention Using Random Forest. 10.1007/978-3-030-49161-1_4.

Baati, Karim. (2021). Hybridization of Adaboost with Random Forest for Real-Time Prediction of Online Shoppers' Purchasing Intention. 10.1007/978-3-030-73050-5_23.

Ganguly, B., Dash, S. B., Cyr, D., & Head, M. (2010). The effects of website design on purchase intention in online shopping: the mediating role of trust and the moderating role of culture. *International Journal of Electronic Business*, *8*(4-5), 302-330.

G. Sang and S. Wu, "Predicting the Intention of Online Shoppers' Purchasing," (2022) 5th International Conference on Advanced Electronic Materials, *Computers and Software Engineering* (AEMCSE), 2022, pp. 333-337, doi: 10.1109/AEMCSE55572.2022.00074.

Michael Bosnjak, Mirta Galesic, Tracy Tuten,(2007) Personality determinants of online shopping: Explaining online purchase intentions using a hierarchical approach, Journal of Business Research, Volume 60, Issue 6, 2007, Pages 597-605, ISSN 0148-2963, https://doi.org/10.1016/j.jbusres.2006.06.008.

Qiang Su, Lu Chen(2015), A method for discovering clusters of e-commerce interest patterns using click-stream data, *Electronic Commerce Research and Applications,* Volume 14, Issue 1, 2015, Pages 1-13, ISSN 1567-4223, https://doi.org/10.1016/j.elerap.2014.10.002. (https://www.sciencedirect.com/science/article/pii/S1567422314000726)

Sakar, C.O., Polat, S.O., Katircioglu, M. *et al.* Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks. *Neural Comput & Applic* **31**, 6893–6908 (2019). https://doi.org/10.1007/s00521-018-3523-0

Appendices

Appendix one

(These screenshots show the referenced procedures that were completed for Task 1. The source of the information is a combination of self-written notes, a Jupyter notebook, and Azure)

1. Experimentation and Prediction using Decision Tree

```
#Splitting the dataset into the Training set and the Test set:
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3, random_state=0)
```

```
#The Following code standardizes the train and test dataset.
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train_s=sc.fit_transform(x_train)
x_test_s=sc.transform(x_test)
```

```
#Training the model:fitting Decision Tree to the training set
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
#Evaluating the model:
#predicting the test set results
y_pred=classifier.predict(x_test_s)
print(y_pred)
```

```
[0 0 0 ... 0 0 0]
```

```
print(y_test)
```

```
[0 0 0 ... 0 0 0]
```

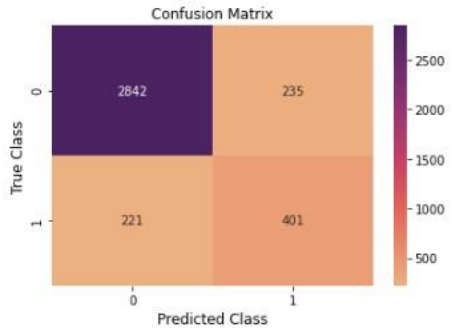
- Performance evaluation

```
#To evaluate the performance of model, you can use the following code:
from sklearn import metrics
acc=metrics.accuracy_score(y_test,y_pred)
print('accuracy:%.2f\n\n'%(acc))
cm=metrics.confusion_matrix(y_test,y_pred)
print('Confusion Matrix:')
print(cm,'\n\n')
print('---------------------------------')
result=metrics.classification_report(y_test,y_pred)
print('Classification Report:\n')
print(result)
```

```
accuracy:0.88


Confusion Matrix:
[[2842  235]
 [ 221  401]]


---------------------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.93      0.92      0.93      3077
           1       0.63      0.64      0.64       622

    accuracy                           0.88      3699
   macro avg       0.78      0.78      0.78      3699
weighted avg       0.88      0.88      0.88      3699
```

```
ax = sns.heatmap(cm, cmap='flare',annot=True, fmt='d')
plt.xlabel("Predicted Class",fontsize=12)
plt.ylabel("True Class",fontsize=12)
plt.title("Confusion Matrix",fontsize=12)
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```



2. Experimentation and Prediction using Neural Network with Keras

```
# Using train_test from Scikit Learn to divide the data into a training dataset and test dataset
#Normalising the data to train a neural network, Using standardscaler estimator from Scikit Learn : Thereby making varaible 0 and
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
x = dataset.drop(columns=["Month","Revenue","SpecialDay","Region","TrafficType","Weekend"])
y = dataset["Revenue"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, stratify=y, random_state=99)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
%pip install imblearn
```

```
from imblearn.over_sampling import RandomOverSampler

resampler = RandomOverSampler(random_state=0)
x_train_oversampled, y_train_oversampled = resampler.fit_resample(x_train,y_train)

sns.countplot(x=y_train_oversampled)
```

- Building and training my neural network:

```
# Part two: Building and training my neural network
# Using Keras sequential model allows me to build my neural network layer by layer
# using "Dense" layer
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(8,activation="relu",input_shape=(12,)))
model.add(tf.keras.layers.Dense(2,activation="softmax"))
```

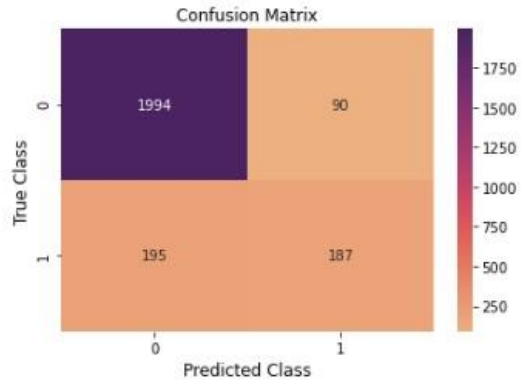
- Evaluating the model

```
# Evaluating Network
accuracy = history.history["accuracy"]
validation_accuracy = history.history["val_accuracy"]

plt.plot(accuracy, label="Training Set Accuracy")
plt.plot(validation_accuracy,label="Validation Set Accuracy")
plt.ylabel("Accuracy")
#plt.ylim([min(plt.ylim()),0.7])
plt.title("Training and Validation Accuracy Across Epochs")
plt.legend()
```

```
#For Loss
loss = history.history["loss"]
validation_loss = history.history["val_loss"]

plt.plot(loss, label="Training Set loss")
plt.plot(validation_loss,label="Validation Set loss")
plt.ylabel("loss")
#plt.ylim([min(plt.ylim()),0.7])
plt.title("Training and Validation Accuracy Across Epochs")
plt.legend()
```

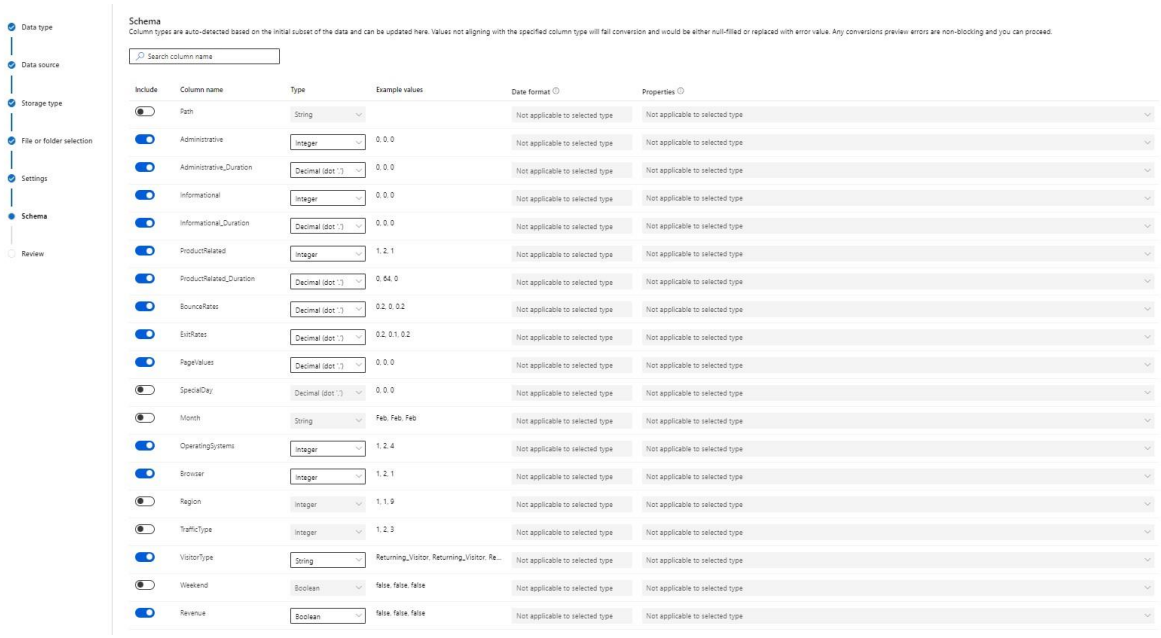
- Visualisation of the results.

```
confusion_matrix = confusion_matrix(y_test,y_pred)
ax = sns.heatmap(confusion_matrix,cmap='flare',annot=True,fmt='d')
plt.xlabel('Predicted Class',fontsize=12)
plt.ylabel('True Class',fontsize=12)
plt.title('Confusion Matrix',fontsize=12)
plt.show()
```

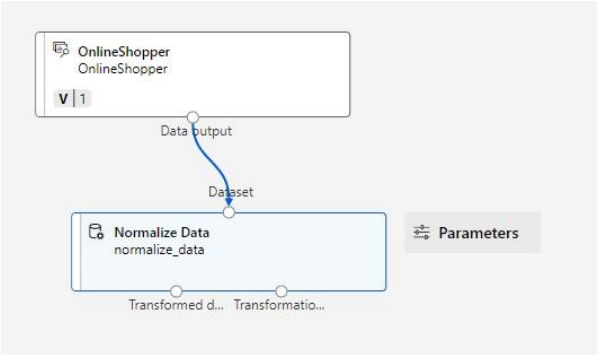


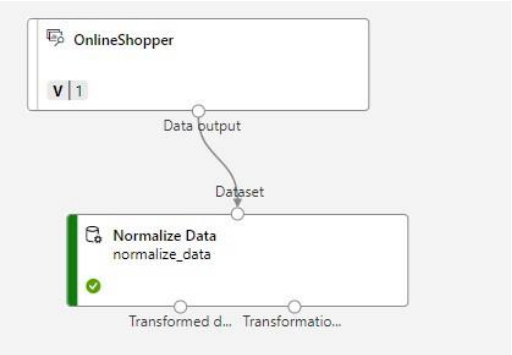
3. Experimentation and Prediction using Azure Machine Learning Designer

- Filtering Data Needed:

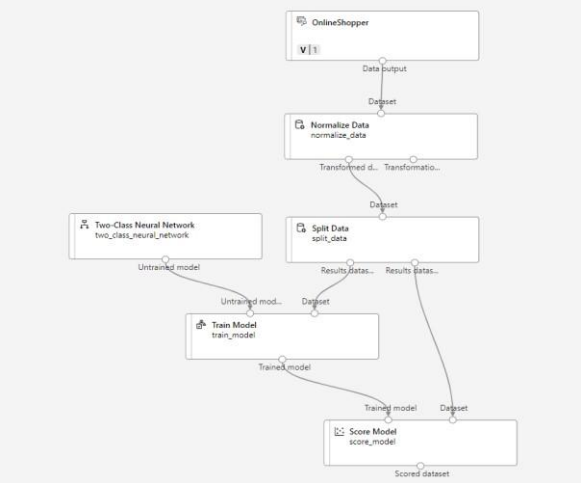


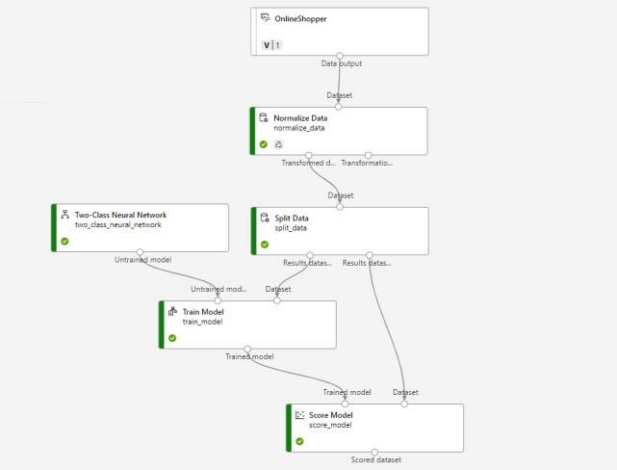
- Adding Transformations:



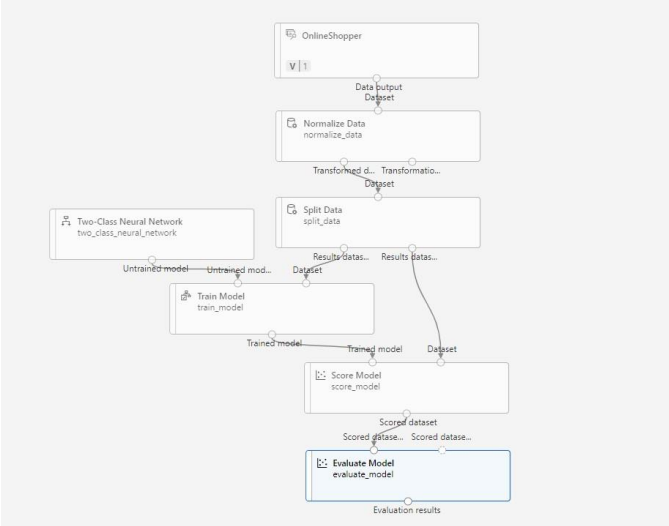


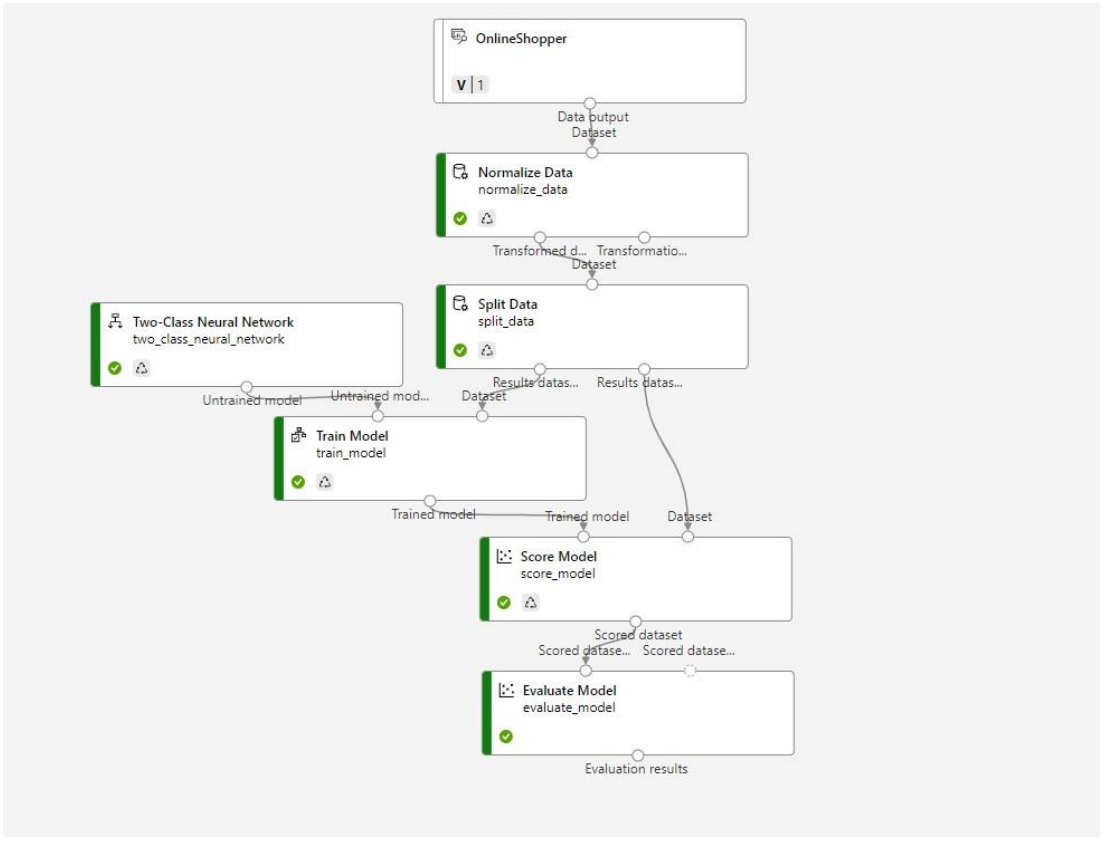
- Adding Training Modules:

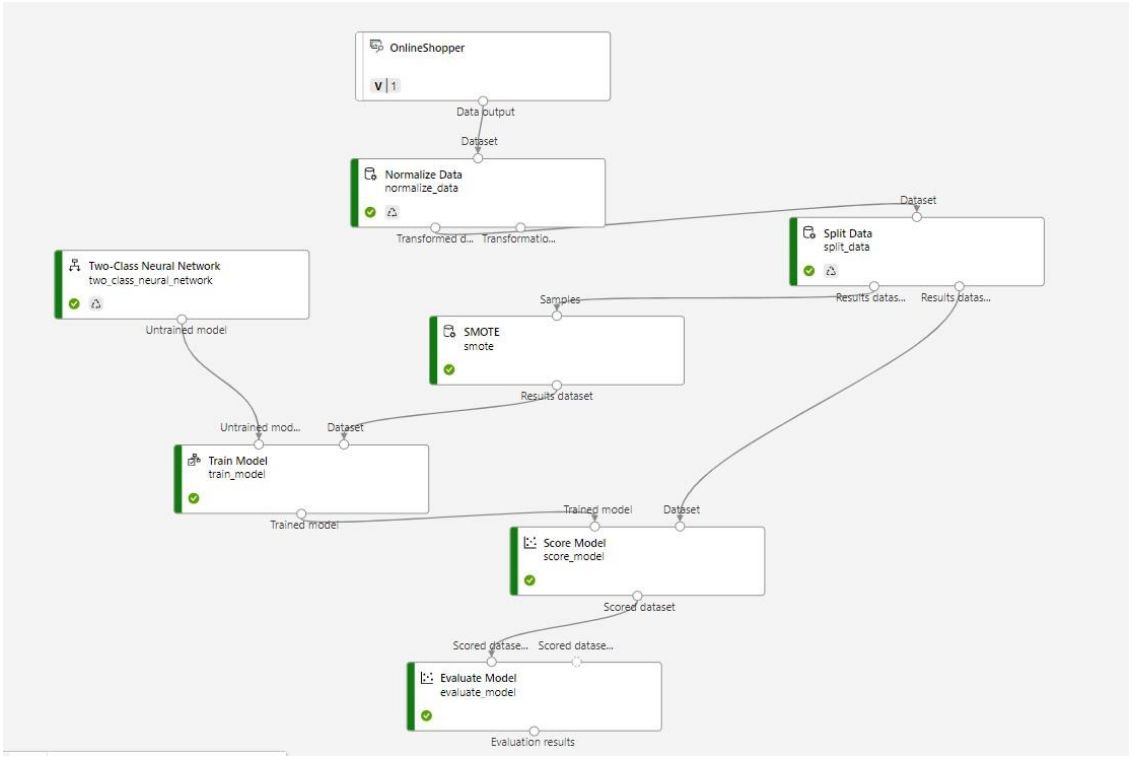




Evaluating the Model:


OnlineShopper
V | 1
Data output
Dataset
Normalize Data
normalize_data
Transformed d...
Transformatio...
Dataset
Two-Class Neural Network
two_class_neural_network
Split Data
split_data
Results datas...
Results datas...
Untrained model
Untrained mod...
Dataset
Train Model
train_model
Trained model
Trained model
Dataset
Score Model
score_model
Scored dataset
Scored datase...
Scored datase...
Evaluate Model
evaluate_model
Evaluation results


OnlineShopper
V | 1
Data output
Dataset
Normalize Data
normalize_data
Transformed d...
Transformatio...
Dataset
Two-Class Neural Network
two_class_neural_network
Split Data
split_data
Results datas...
Results datas...
Untrained model
Untrained mod...
Dataset
Train Model
train_model
Trained model
Trained model
Dataset
Score Model
score_model
Scored dataset
Scored datase...
Scored datase...
Evaluate Model
evaluate_model
Evaluation results


OnlineShopper
Data output
Dataset
Normalize Data
normalize_data
Transformed d...
Transformatio...
Dataset
Split Data
split_data
Results datas...
Results datas...
Two-Class Neural Network
two_class_neural_network
Untrained model
Samples
SMOTE
smote
Results dataset
Untrained mod...
Dataset
Train Model
train_model
Trained model
Trained model
Dataset
Score Model
score_model
Scored dataset
Scored datase...
Scored datase...
Evaluate Model
evaluate_model
Evaluation results