PREDICTIVE ANALYSIS AND DEPLOYMENT FOR CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING TECHNIQUES

FINAL REPORT

Higher Diploma in Science in Data Analytics

Author: Ciaran Finnegan / 10524150

E-mail: 10524150@mydbs.ie / ciaran@feefinnegan.com

Supervisor: Dr Shahram Azizi Sazi

Final Report Submission Date: 25th September 2020

Abstract

In 2016, the total value of fraud for credit cards issues within the SEPA region was €1.8 billion (Whatman, 2019). The company with which I am employed works in the domain of financial crime prevention software, so I chose to focus this project on one of the key challenges in this area. My project delivers an end-to-end solution that uses an Azure hosted predictive model, accessed via a separate Shiny R dashboard, to assess individual credit card transactions in real time for the likelihood of fraud.

A dataset of 25K+ historical US credit card transactions (2104), each one labelled as 'Fraud' or 'Not Fraud', is engineered to train and deploy a model to predict if 'future' transactions appear to be fraudulent.

The Machine Learning modelling process is managed through the online Microsoft Azure Machine Learning Studio (classic) platform. This includes the hosting of a REST Endpoint for the model, to be accessed as a Web Service by an external application for fraud prediction.

A separate Shiny R dashboard is included in this project to access the predictive fraud model through an API call, passing the details of 'new' card transactions as parameters one-by-one in real time to the Azure Web Service.

This Shiny dashboard is hosted on the ShinyIO platform and also provides a secondary interface to provide key data visualisation graphs on the original dataset

Acknowledgments

I want to acknowledge the advice and support provided by two fellow work colleagues from BAE Systems Applied Intelligence, Eddie Baggott and Dan Branley. Both worked in the area of software development for fraud prevention and were able to provide me with the demo data that I re-purposed as a dataset for this data analytics project.

I also wish to acknowledge direction given by my project supervisor Dr Shahram Azizi Sazi, which helped guide my approach to this Final Report and the with the wider project in general.

Contents

1. Introduction

1.1. What the Project Aimed to Deliver

The artefact at the end of this project is an application that invokes a bespoke predictive model and provides a user with an online interface to retrieve a score for whether a given credit card transaction is likely to be fraudulent.

The project user interface is a Shiny R Dashboard hosted on the *Shinyapps.io* platform. See Section 7 of this document for the URL and User Guide.

The predictive model itself has been built in an *Azure Machine Learning Classic Studio Workspace*. See Section 9.2 of this document for a detailed description of the Machine Learning workflow process employed to engineer the dataset, train and evaluate the model, and then deploy the Web Service to allow access to the predictive model.

1.2. How the Project Delivery was Implemented

The user interface is built as a hosted Shiny R dashboard application, which provides two primary functions:

- A means to select a given 'new' credit card transaction and assess in real time if this record is likely to be fraudulent.
- Provide a visual analysis of the credit card dataset used to build the predictive card model.

The predictive fraud model was built and deployed using the following steps:

- My credit card fraud dataset contains 25K rows, each containing a label for 'fraud/non-fraud'.
- A dedicated Azure Machine Learning project has been used for all the project artifacts: datasets, experiments, models, Web Services etc.
- A sequence of experiments in Azure ML Studio (classic) is used for Feature Engineering of the dataset prior to modelling.
- Various classification algorithms are evaluated, leading to further iterations of the Feature Engineering experiments.
- A Model is trained based on the most optimal algorithm, using the final Feature Selection decisions. This model is deployed as an Azure hosted Web Service.

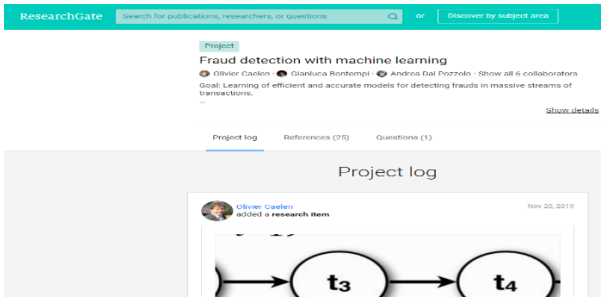
2. Background / Literature Review

2.1. Credit Card Fraud Detection: Further Research on Predictive Models

Section 2.2 of the Interim Report on this project elaborated on two Kaggle submissions based on the credit card fraud dataset generated by the work of the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles).

References in both those submissions referred to ongoing studies in the domain of credit card fraud detection that are being collected by the ResearchGate network for scientists and researchers.

Figure: www.researchgate.net/project/Fraud-detection-with-machine-learning



The latest submission on the ResearchGate *FraudDetection* site, as of September 2020, contains an interesting paper on credit card fraud detection with a focus on transaction sequences. However, the initial sections of this submission (Lucas et al., 2019) also provide an excellent overview of the challenges facing credit card detection in the real world and machine learning solutions that have emerged over the last 10+ years.

Reading through this material I have drawn on certain key observations to direct my work on this project.

Algorithm Selection

A paper from Bhattacharyya, Jha, Tharakunnel and Westland in 2011 described research on a real-world US credit card dataset. It involved a comparison of Support Vector Machine, Random Forest, and Logistic Regression, which – as expected - are all algorithm options I have access to in Azure ML Studio (classic).

Important points of which I took note (and are repeated in other articles) were:

- Credit card data is often very imbalanced. Fraud can be disastrous when it happens, but it is a tiny proportion of overall transaction numbers. A defined sampling approach is a definite requirement.
- The Fraud/non-Fraud imbalance can make the use of 'Accuracy' in a Confusion Matrix somewhat ineffective.
- Accurate identification of fraud is often a primary requirement so there is a need to look at the trade-offs in improving Recall and Precision.

- Logistic Regression can perform consistently well but is dependent on the approach to Feature Engineering.

One interesting opinion from other research is that the imbalanced nature of credit card data makes predicting fraud a candidate for anomaly detection routines (Ceronmani Sharmila et al., 2019). Algorithms such as 'Isolated Forest' or 'Local Outlier Factors (LOF)' are frequently recommended.

As I explain in Section 2.2 of this document, I chose not to adopt these unsupervised approaches and focused much of my time in the Machine Learning workflow for this project on Feature Engineering leading into supervised learning techniques.

Feature Engineering

Although I was not able to read all the details in the paper by Mahmoudi and Duman, 2015 on fraud detection analysis, several commentators on this study referred to the benefit of being able to work with the 'raw' features of a credit card dataset.

This is an advantage I have with my dataset, as opposed to the previously mentioned ULB data that is heavily anonymised through PCA.

However, the opening lines of a paper from Lima and Pereira, 2017 on '*Feature Selection Approaches to Fraud Detection in e-Payment Systems*' states that "..*Due to the large amount of data generated in electronic transactions, to find the best set of features is an essential task to identify frauds*."

Given that my starting dataset has 380 columns, this was a guiding principle for me.

I was also going to have to code in R to invoke an API to call my predictive fraud model with all the 'important' features on 'new' credit card transactions passed as parameters. Therefore, reducing the complexity of setting up this parameter list for the API code would help improve the robustness of the UI code.

Transaction Sequences

The ResearchNet articles provided references to additional papers on how to improve Feature Engineering for fraud analysis by creating aggregates and time series analyses of the transactions. I choose not to explore this avenue because of the potential complexity.

There are many columns in my dataset that look at time since transaction but my primary response to this data was just to remove any highly correlated features.

2.2. Credit Card Fraud Detection: In Context – My Dataset

To give an overview of my credit card dataset:

1. It contains 25,128 rows and 380 columns.

2. This is a live dataset of North American credit card transactions from 2013. Only names and initial address lines have been anonymised. Apart from data cleansing, no other alterations to the 'raw' transactions have taken place.

3. Previously, the data was used for a, now discontinued, credit card fraud product that relied on a 'Rules Engine' to generate alerts for potential fraud.

4. The data is expected to be free of corrupt data elements, and largely free of missing data.

5. Many columns still present in the dataset were created as the result of ETL processes from other peripheral systems and are redundant. No domain knowledge in this area has been documented.

6. Approximately 15% of the dataset records are known fraud cases. The data has been balanced over a period of time in 2013. This is a significant advantage/difference from other comparable research datasets in the public domain.

7. The original project proposal, as described in the Interim Report, was to use a dataset of 280K records but that became infeasible due to reasons elaborated on in Section 8 of this document (Conclusions).

8. 10% of the dataset was used for initial feature engineering in Azure ML Studio (classic), but the full dataset was used to train the production model.

Based on the research described in the previous section (2.1), my approach to building this predictive credit card fraud model focused on:

- The assumption that data sampling and balancing would be relatively straightforward for my project. Other industry papers have devoted significant amounts of time to addressing the challenge of balancing a very small sub-set of actual fraud data.
- Feature Engineering would be very important. This is true of most Machine Learning problems, but I need to reduce the feature set from a starting number of 380.
- Algorithm selection, to build my predictive fraud model, would focus on binary Classification options for supervised learning.

3. Requirements: Specification and Design

3.1. High Level Project Requirements

In order to achieve the objectives of the project mission statement, as described in the Interim Report document, the following requirements needed to be met;

- A dataset is provided with sufficient volume and richness of attributes to allow for appropriate data preparation and modelling to be executed.

- A predictive model for Credit Card fraud detection is built and deployed using an effective Machine Learning workflow process, which produces results that are as accurate as reasonably possible. Detecting fraud is a priority, so a good Recall score from the model was important.

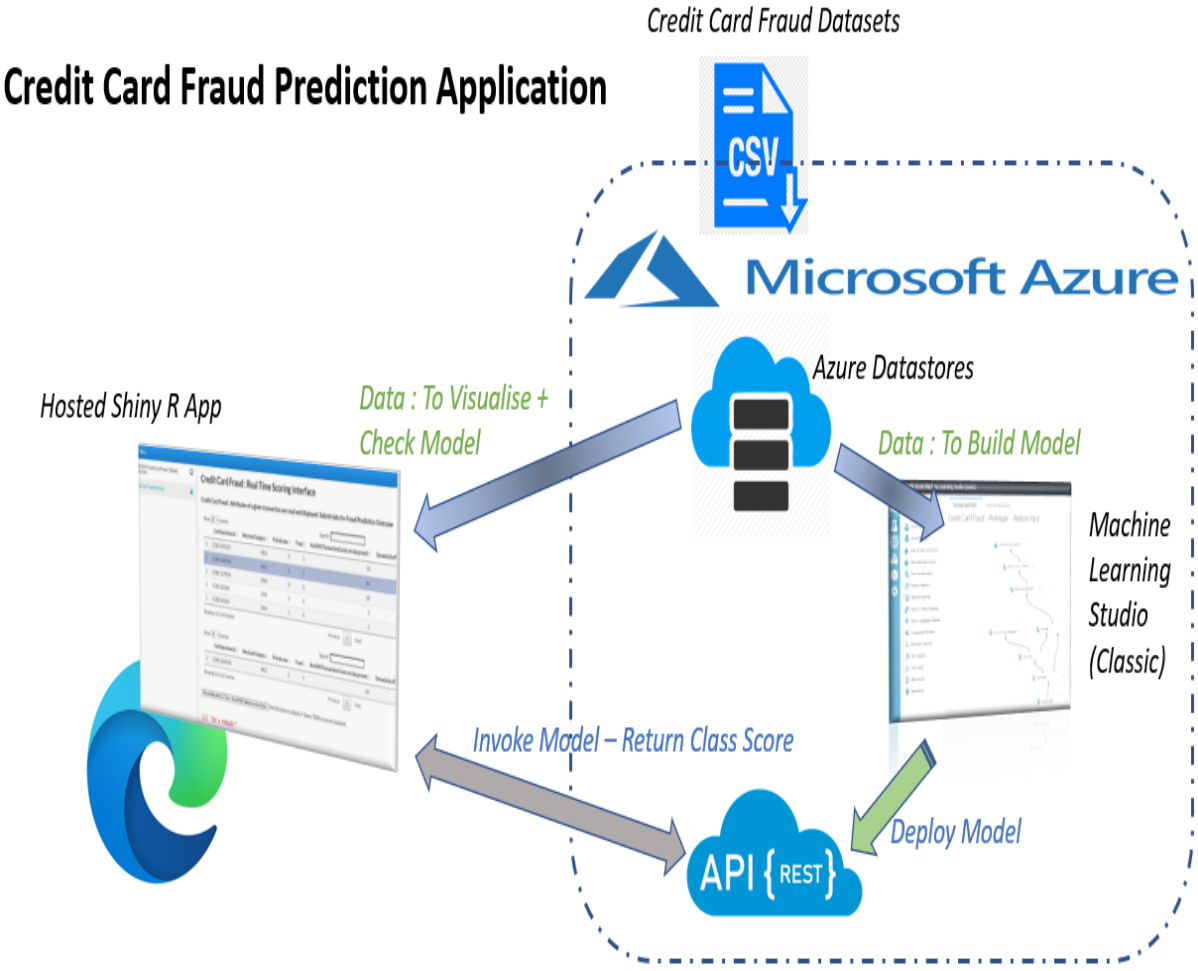
- All development and system execution takes place on cloud-based platforms. There is no dependency on local PC libraries or IDEs, etc.

- The end user will work with a Shiny R application interface, built using RStudio Cloud, and choose a given single credit card fraud transaction to investigate. A real-time prediction of the likelihood of fraud will be provided to the user on screen.

- The R Shiny application can access the source datasets, hosted in Azure, to provide data visualisations as a peripheral service to the end user.

3.2. Project Architecture Diagram

Figure: High Level Application Architecture Diagram

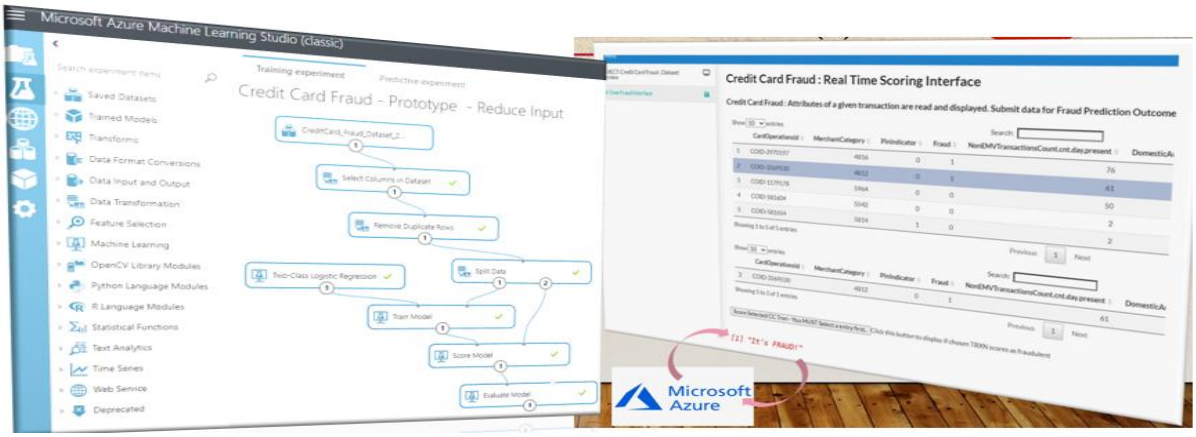


3.3. High Level Project Design

The Interim report provided a detailed overview of the User Stories used to map out the design and implementation of this project.

3.3.1. Prototype Development – Initial User Stories

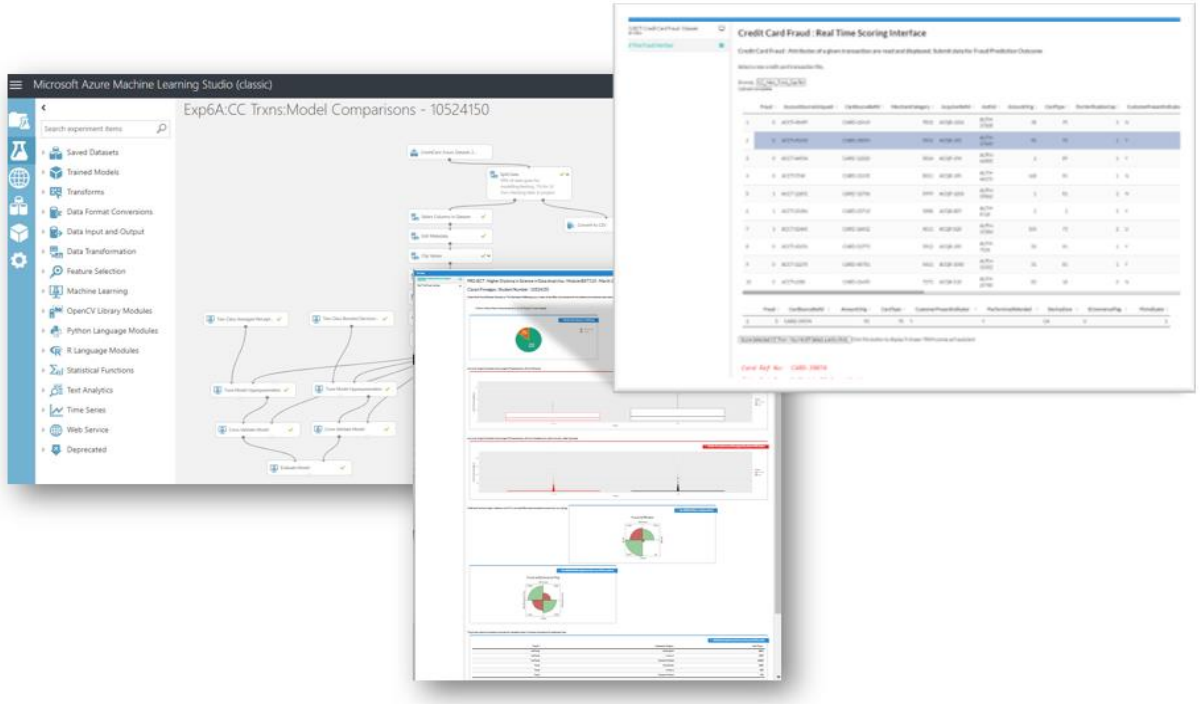
Figure: Initial Basic Modelling in Azure ML Studio (Classic) and Basic UI Deployment



For the Interim Report, a basic model, with limited Feature Engineering and no tuning, was trained and deployed as a Web Service. The Shiny R dashboard was a hosted application but used pre-loaded transactions for fraud assessment.

3.3.2. Final Project Deliverable – Further User Stories

Figure: Enhanced Modelling in Azure ML Studio (Classic) and Enhanced UI Deployment



The final version of the project involved a full Machine Learning workflow process to train and deploy a reliable predictive model. The user can select 'new' credit card transactions from multiple files through the UI and assess any given one for fraud in real time.

4. Project Implementation (1) – Azure Modelling

4.1. The Machine Learning Workflow

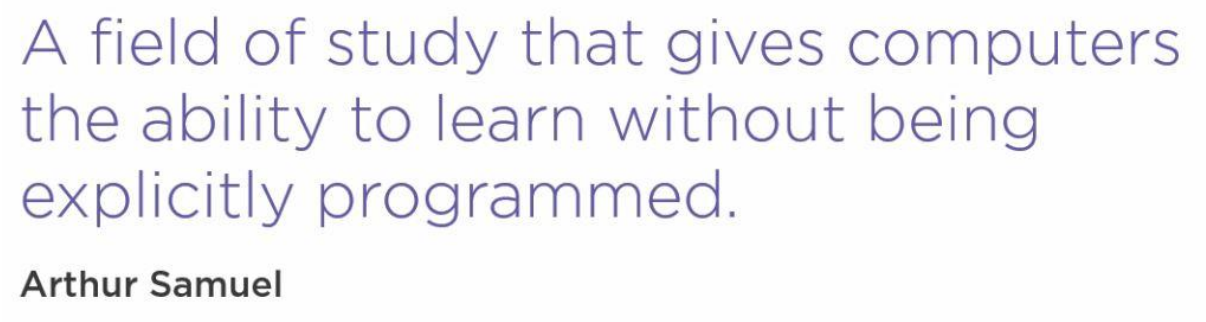
A significant amount of training and reference material to which I had access came from Pluralsight courses on the Microsoft Azure Machine Learning Studio platform.

I have reproduced a number of illustrations from those courses (and cited the sources) to;

- Explain my general approach to using Machine Learning processes to build my credit card Fraud predictive model.
- Describe how Azure Machine Learning Studio was used to implement the key steps in the Machine Learning process for this project.

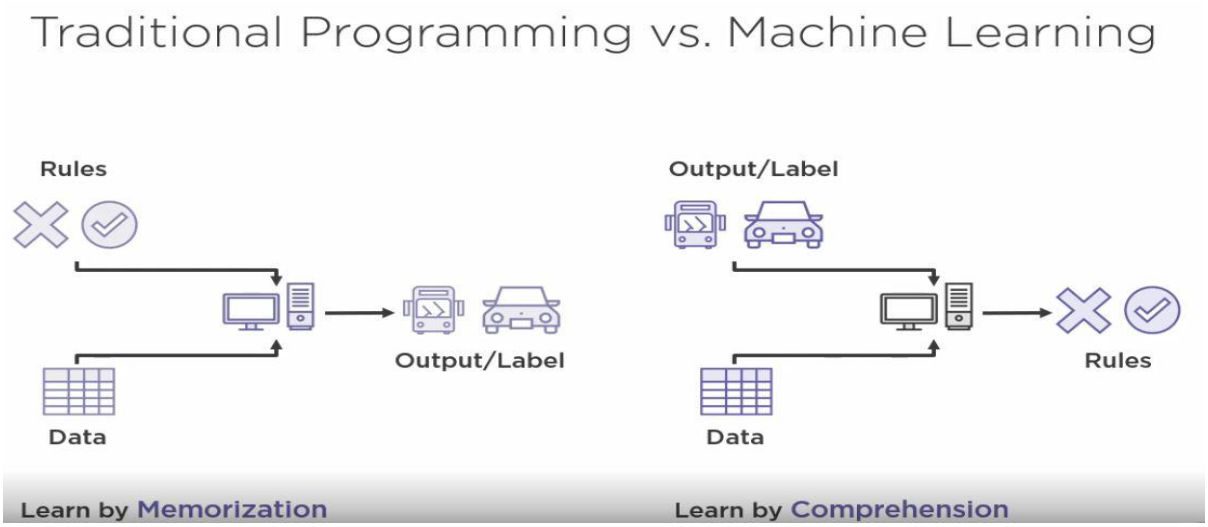
To start with a quote…"what is Machine Learning?"

Figure: Reproduced Quote Image from Pluralsight (Kurata, 2016)



The difference between Machine Learning and 'traditional programming' can be illustrated briefly as follows.

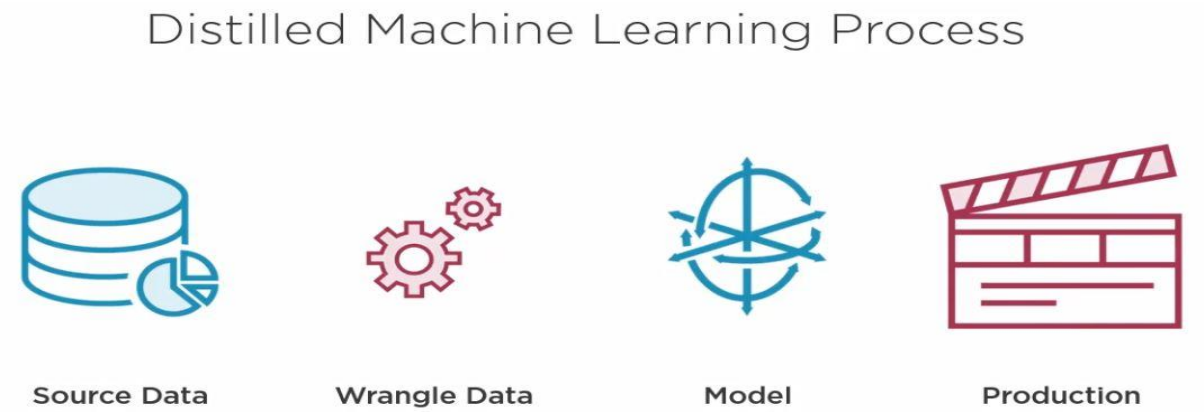
Figure: Traditional Programming v Machine Learning - *Reproduced Image from Pluralsight* (Rhodes, 2020)



This project aims to create a model that can take unseen data and determine a prediction as to whether the transaction is fraudulent, as opposed to an approach such as writing code that implement a sequence of pre-defined rules.

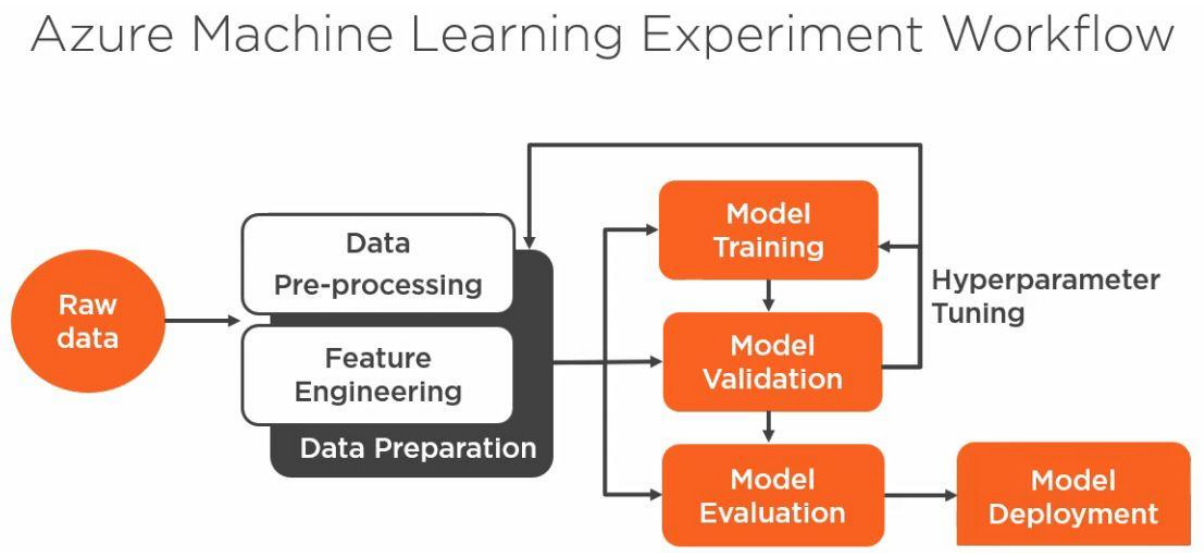
This is a simplified diagram of how the Machine Learning process is applied.

Figure: Reproduced Image from Pluralsight (Kurata, 2016)



The following figure shows the steps in Azure Machine Learning Studio about which I will provide further implementation details in Section 4.2 through to Section 0 of this report.

Figure: *Reproduced Image from Pluralsight* (Rhodes, 2020)



4.2. Credit Card Fraud – The Azure Workspace/Machine Learning Studio

The steps to create an Azure account and Workspace are well documented by Microsoft, and I have not sought to reproduce them in detail in this document.

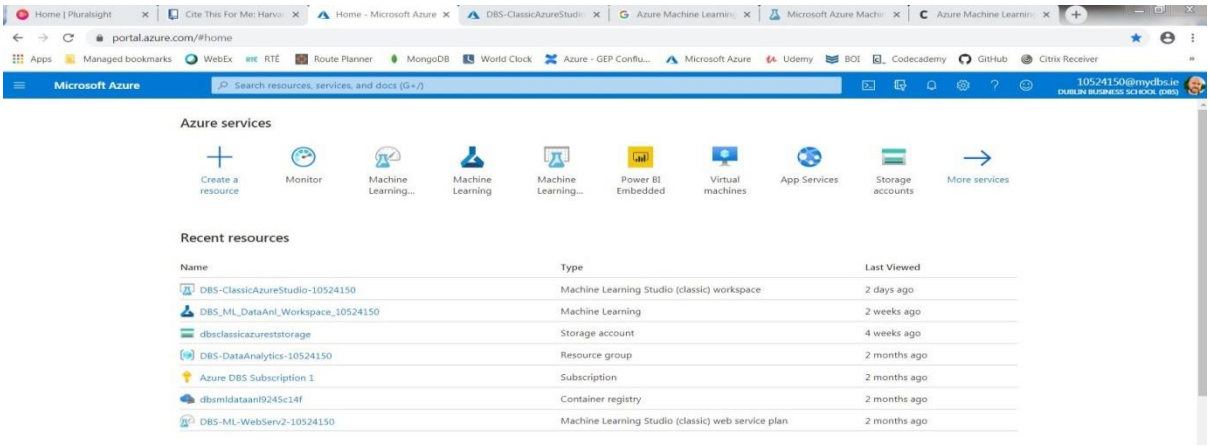
Similarly, the set up required for Azure Machine Learning Studio is equally well documented and accessible from within the Azure portal.

In brief; a description of Azure Workspaces can be found here;
https://docs.microsoft.com/en-us/azure/machine-learning/concept-workspace

A description of the Azure Machine Learning Studio/Services offering is described here:
https://docs.microsoft.com/en-us/azure/machine-learning/overview-what-is-machine-learning-studio

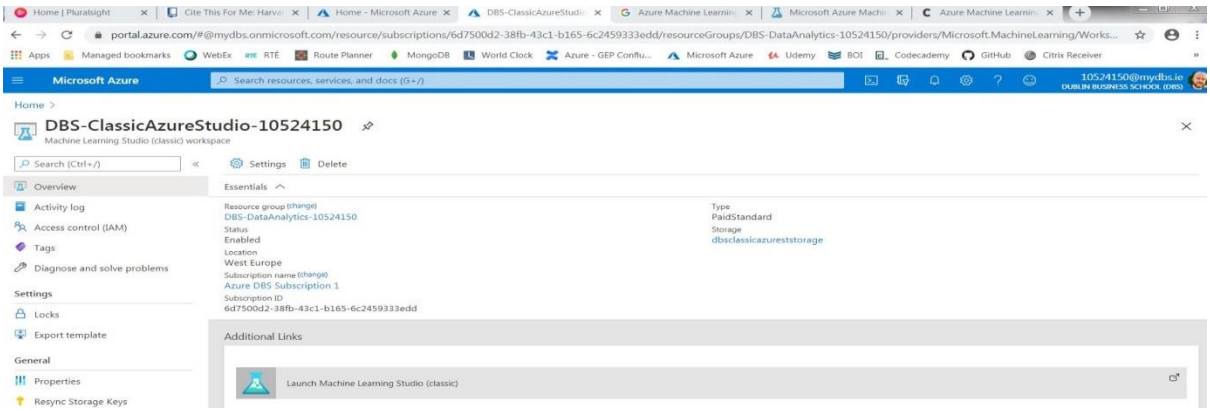
To access the Azure Machine Learning Studio (classic) platform where I developed my project the first step is to log onto the Azure Portal, which I set up with my DBS account.

Figure: Azure Portal (my DBS account)



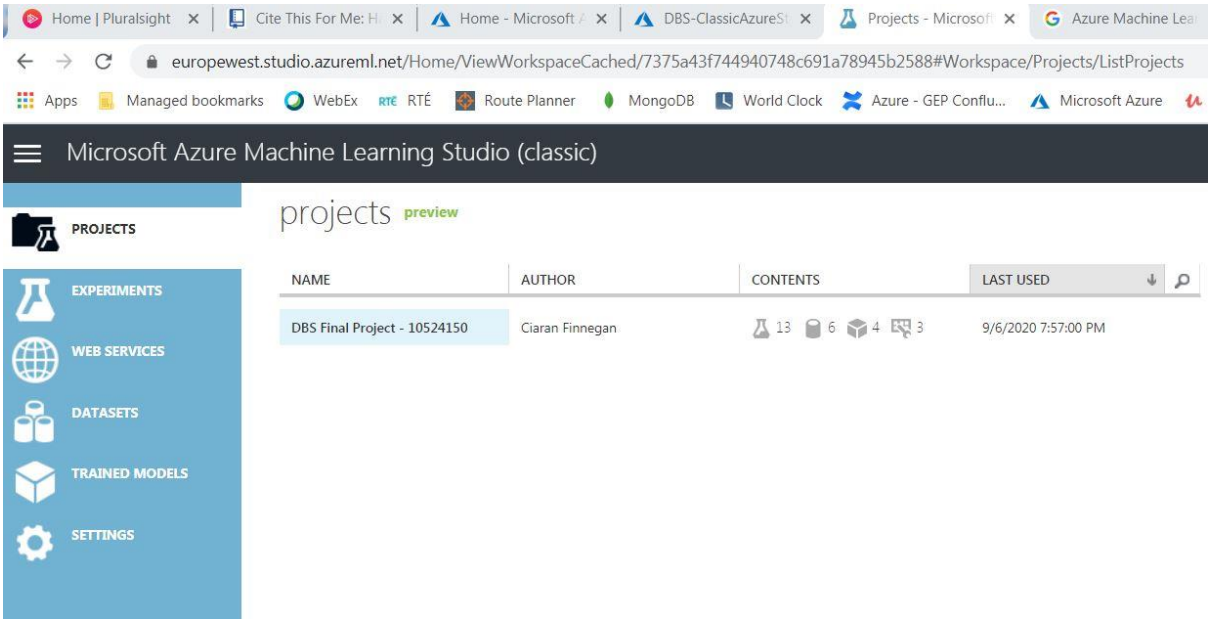
A workspace has been created by me for a Machine Learning Studio (classic) environment.

Figure: Azure ML Studio (classic) Workspace



Launching the Machine Learning Studio (classic) services will, after additional user verification, open the ML Studio (classic) application itself.

Figure: Microsoft Azure Machine Learning Studio (classic)



This ML Studio follows many of the conventions of similar products on the marketplace in terms of organising work under a 'Project' structure.

My fraud ML Experiments use the datasets, or outputs of other experiments, to build up the predictive credit card fraud model for this project.

Once ready, my 'final' experiment is promoted to a 'Web Service' which can then be invoked externally (by my Shiny R application in the case of this project).

The following sections are a sequential analysis of the experiments used in the 'Project' to progress through all the steps of the Machine Learning process.

Experiments have been numbered in sequence, but the machine learning process for this project iterated backwards and forwards across the experiments as refinements and alternative options were identified.

Note: Why use the 'Classic' version of the Microsoft Machine Learning Studio?

In his brief article from 2019 on Codit, Sriram Narayanan, describes the additional features that the more recent Microsoft Azure ML *Services* platform offers in comparison to the 'classic' studio. Microsoft itself tries to encourage use of this ML '*Services*' interface.

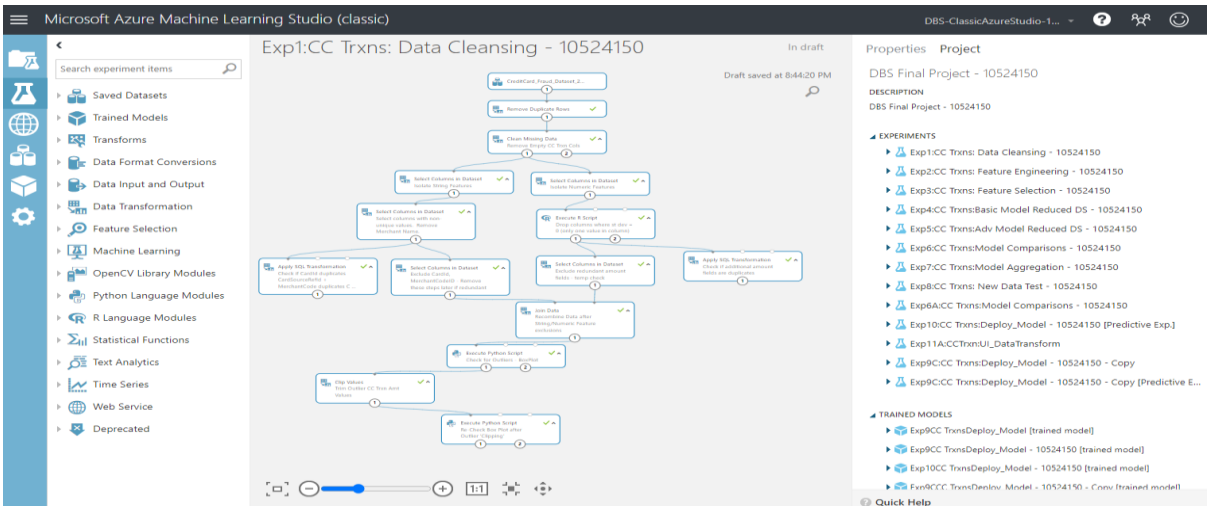
Working iteratively through the prototype phase of this project, I determined that the 'classic' studio was a better option for this delivery for the following reasons:

- **Cost**. The Azure charge for the 'classic' studio is very low and includes the deployment of Web Services / Endpoints. Azure Machine Learning *Services* is significantly more costly for deploying REST Endpoints on AKS Clusters.

- **Complexity and maturity**. Some of the deployment aspects of Microsoft Azure Machine Learning *Services* are still in 'preview' mode. During Prototype development I had to re-code errors within the Python scripts in certain Jupyter Notebooks when using ML Services examples. I believe that the 'classic' option was a more robust platform on which to develop a full 'end-to-end' solution.

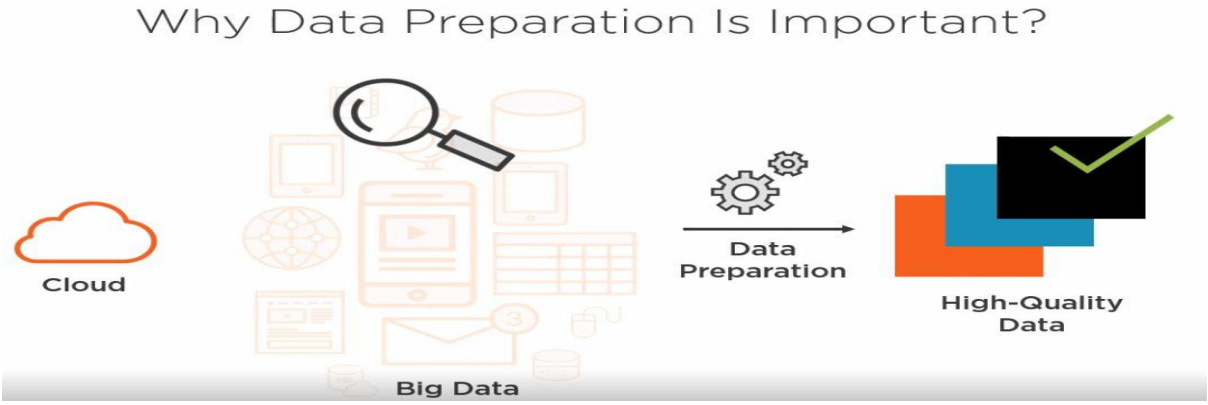
- **Training**. The Pluralsight courses, to which I had access, had a greater range of training material on 'classic' and were an important reference tool for me on this project.

Figure: Overview of Azure ML Studio (classic) environment for this project – Experiment View



4.3. Credit Card Fraud Dataset – Analysis and Preparation

Figure: *Reproduced Image from Pluralsight* (Srinivasulu, 2019)



Section 2 of this document described the importance of 'Feature Engineering' in the general creation of a credit card fraud predictive model.

To focus specifically on my dataset, feature engineering was important because:

- *My original credit card dataset has 380 columns*. Almost certainly, only a fraction of these columns contains information that will directly influence the accuracy of the final model. It was necessary to identity those columns that build the most accurate and performant predictive model for credit card fraud.

- *The dataset is effectively 'clean' but still needs to be checked for 'invalid' data*. There are no invalid characters in the dataset rows, but missing or useless data needs to be identified, if present.

- *40 columns in the original dataset are non-numeric features and will need some form of re-coding*. Many machine learning algorithms can process non-numeric features, but accuracy is likely to be improved if String features are manipulated before the modelling process begins.

This section of the document details the set up and execution of the following experiments:

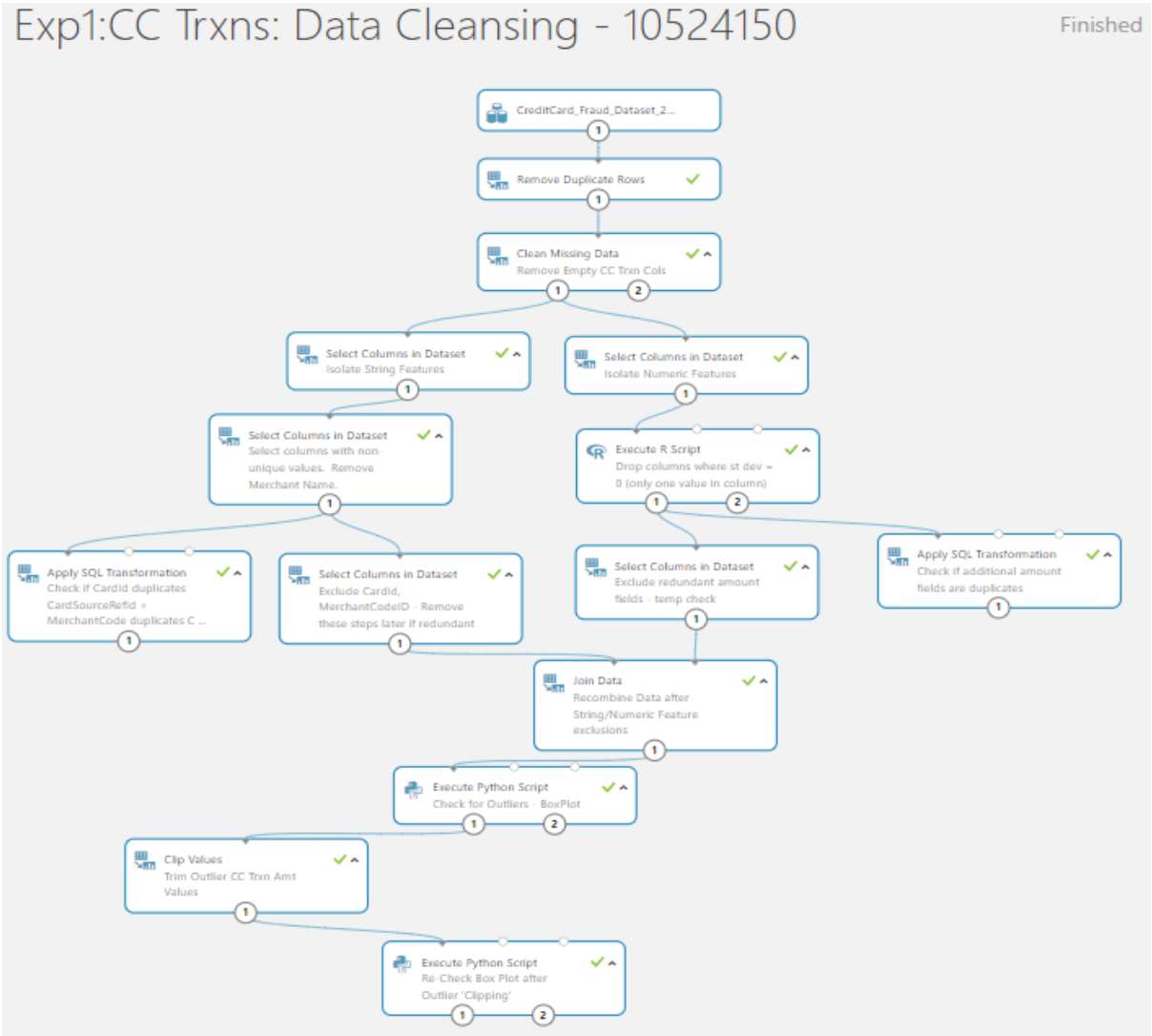
- Experiment 1: Data Cleansing
- Experiment 2: Feature Engineering
- Experiment 3: Feature Selection

Exploratory Data Analysis (EDA) is carried out throughout these experiments but the Shiny App UI provides useful graphical descriptions of the dataset. This can be seen in Section 5.1 of this document.

4.3.1. Experiment 1: Data Cleansing

The figure below illustrates how the Azure Machine Learning Studio (classic) modules were arranged for the data cleansing routines.

Figure: Experiment 1: Data Cleansing



Appendix 9.2 of this document details the specific steps in this experiment.

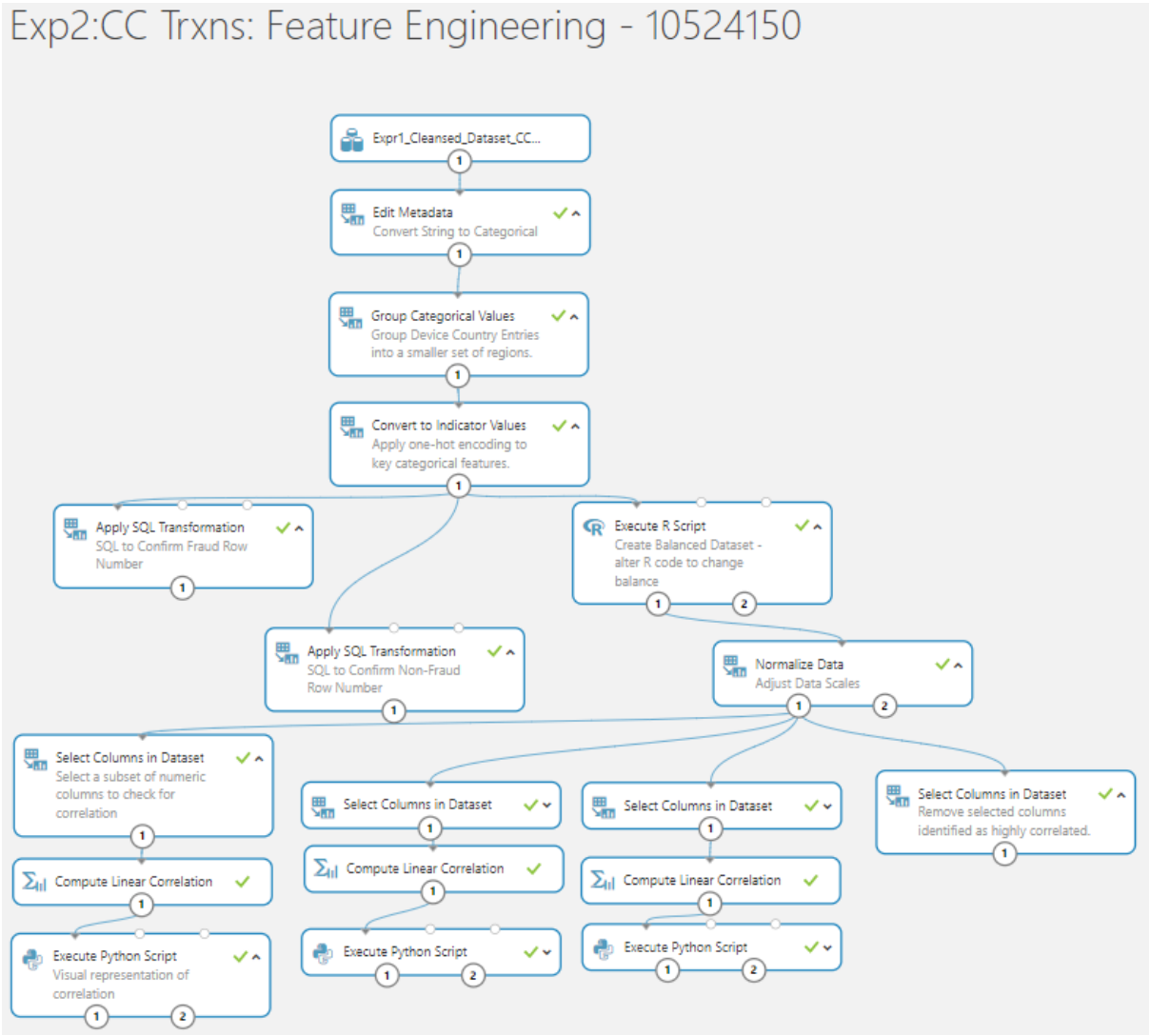
The result of this experiment can be summarised as:

- Dataset reduced to 250 columns of potentially ‘useful’ data.
- Top 5% of outlier values in transaction amount ‘clipped’ to reduce distortion in modelling process.
- Generation of an interim dataset for use in Experiment 2.

4.3.2. Experiment 2: Feature Engineering

The figure below illustrates how the Azure Machine Learning Studio (classic) modules were arranged for the feature engineering routines.

Figure: Experiment 2: Feature Engineering



Appendix 9.2 of this document details the specific steps in this experiment.

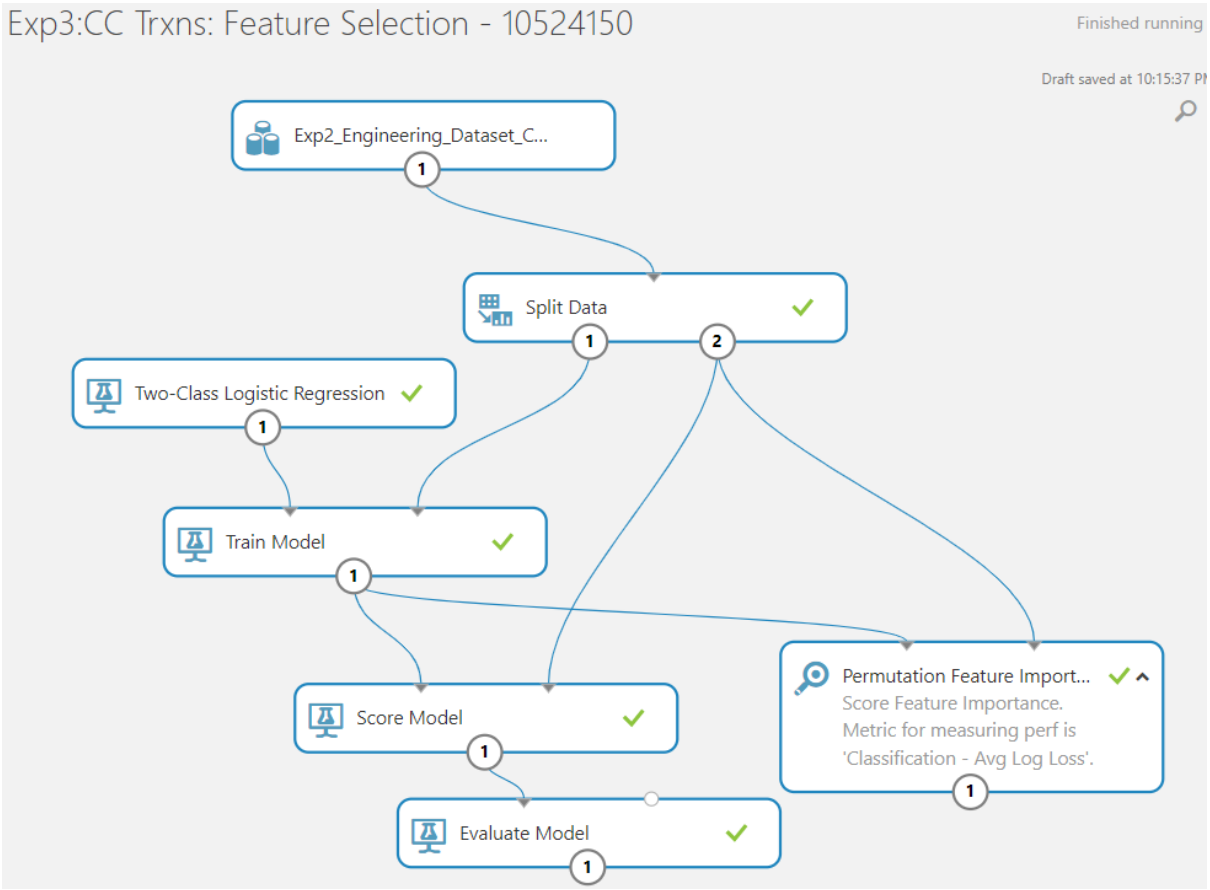
The result of this experiment can be summarised as:

- Conversion of String datatypes to 'Categorical' features
- Grouping of Country Code categorical data and numerical encoding of all categorical features.
- Balancing of dataset (via R code routine) to a 50/50 Fraud/Non-Fraud split.
- Identification and removal of a sub-set of highly correlated features.
- Generation of another interim dataset, which will be the input for Experiment 3.

4.3.3. Experiment 3: Feature Selection

The figure below illustrates how the Azure Machine Learning Studio (classic) modules were arranged for the feature selection routines.

Figure: Experiment 3: Feature Selection



Appendix 9.2 of this document details the specific steps in this experiment.

The result of this experiment can be summarised as:

- Taking the output of the feature engineering steps in Experiment 1 + 2 and generating a predictive fraud model.
- Obtaining a list of features scored in order of importance to the predictive model. The 'Permutation Feature Importance' module produces this output.

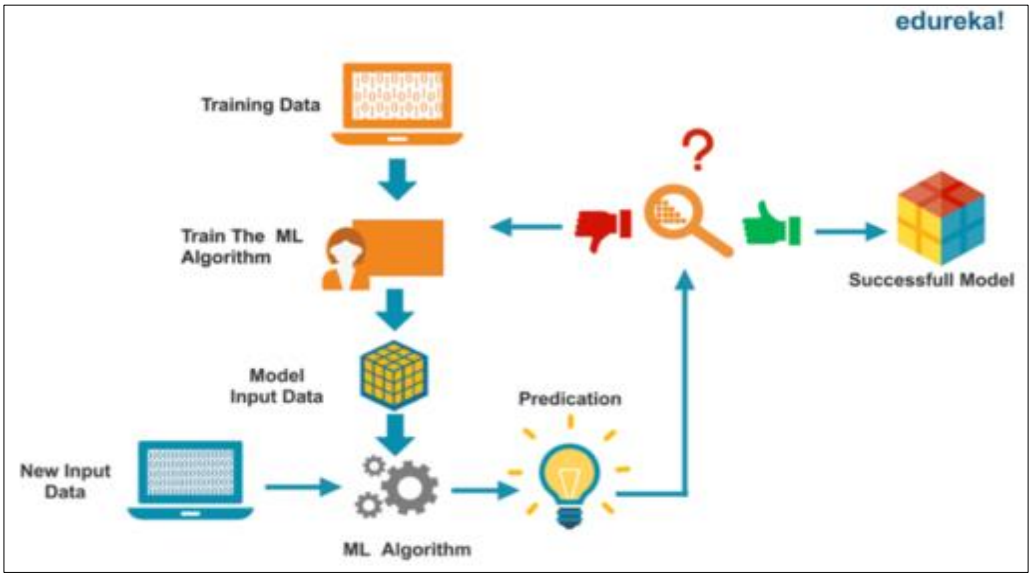
This experiment was run multiple times with various modelling algorithms, based on comparisons seen in later experiments. The 'Two-Class Logistic Regression' algorithm provided the best performing and accurate model and was, hence, used to determine the final list of parameters selected for the model.

This choice of features has a direct impact on the feature set captured in the Shiny App UI and passed to the Rest Endpoint for the predictive model.

4.4. Credit Card Fraud – Building the Azure Model

After a series of iterations backwards and forwards through the experiment sequences, I believed that I now had a refined credit card dataset with which I could run a final batch of modelling experiments.

Figure: Representation of ML Modelling Process Reproduced from Edureka (Lateef, 2020)



The pattern of operations followed the illustration above, but my primary objectives were:

- Determine which classification algorithm, from those available for use in Azure Machine Learning Studio (classic), would be most effective in generating a predictive fraud model based on my credit card transaction dataset. Criteria for algorithm selection would be:
    - Accuracy Score
    - Recall – how well actual Fraud is detected
    - Performance

- Demonstrate the impact on fraud prediction model accuracy, and other metrics, introduced by the following modelling actions:
    - Feature Engineering
    - Cross Validation
    - Hyperparameter tuning

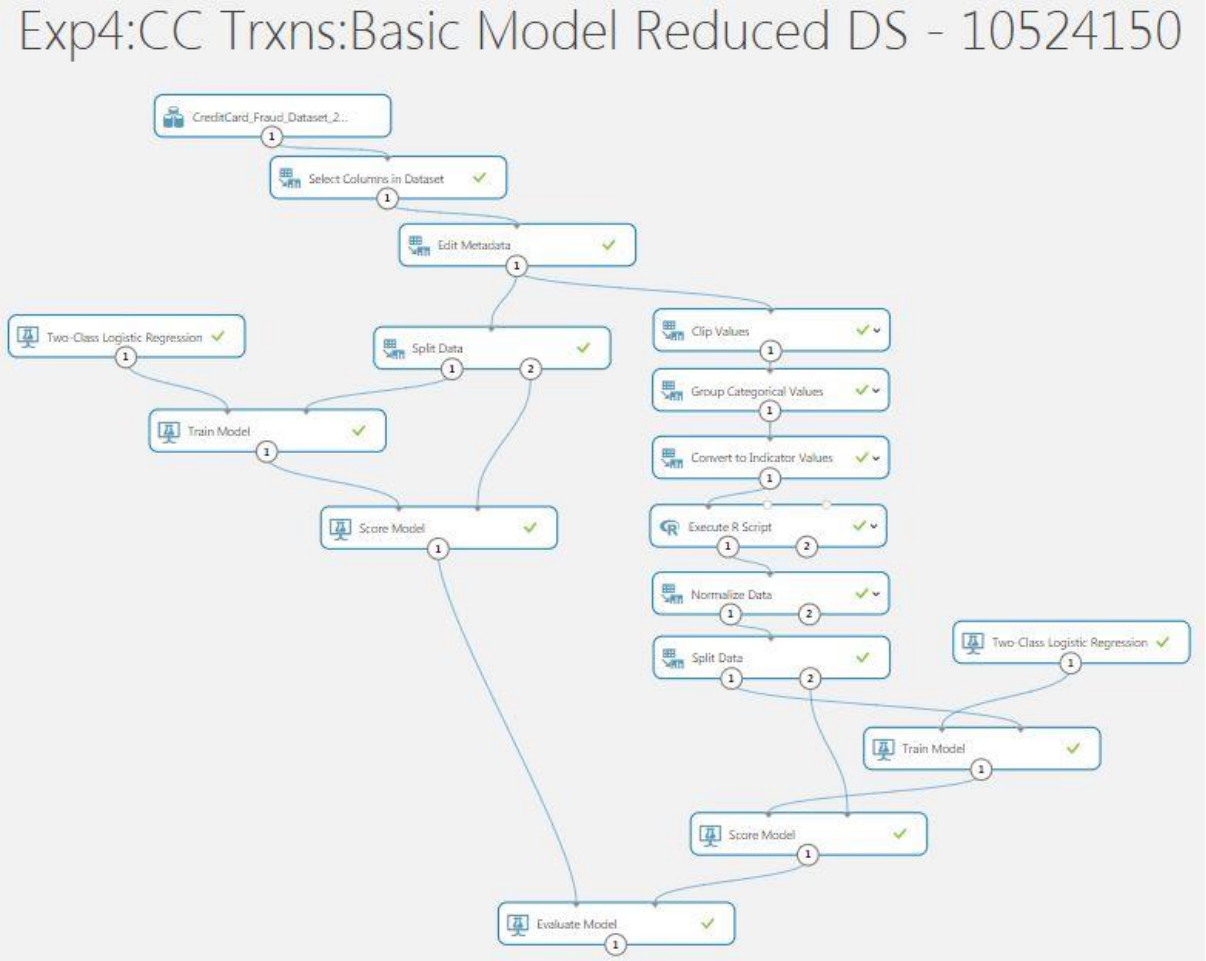
This section of the document details the set-up and execution of the following experiments:

- Experiment 4: Basic Model Evaluation with Feature Engineering
- Experiment 5: Model Evaluation using Cross Validation and Hyperparameter tuning
- Experiment 6 + 7: Comparison of Multiple Classification Algorithms

4.4.1. Experiment 4: Basic Model Evaluation with Feature Engineering

The figure below illustrates how the Azure Machine Learning Studio (classic) modules were arranged to assess the benefits of Feature Engineering.

Figure: Experiment 4: Feature Engineering and Model Evaluation



Appendix 9.2 of this document details the specific steps in the left hand side (LHS) and right hand side (RHS) of this experiment as they largely replicate the work in Experiments 2 and 3.

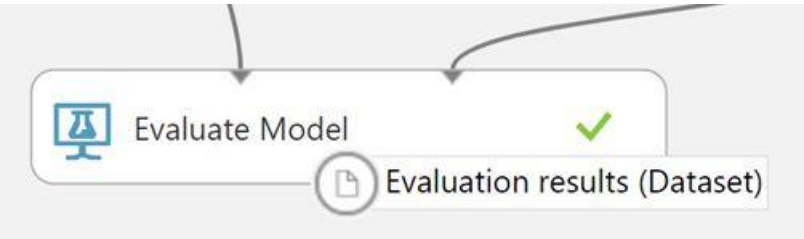
The result of this experiment can be summarised as:

- A demonstration of the impact of feature engineering on model accuracy and other metrics.
- Possible trade-offs that might be acceptable in the modelling process.

Again, 'Two-Class Logistic Regression' is used because of evaluation results in later experiments feeding back into this 'final' version of Experiment 4.

Model Evaluations

The 'Evaluate Model' module provides takes two inputs and provides the key scoring metrics on comparative models as an output.



In Experiment 4:

- The '*Scored dataset*' was the model generated without Feature Engineering, except for the conversion of String features into Categorical features.

- The '*Score dataset to compare*' was the model generated with the Feature Engineering routines in Experiments 1, 2, and 3.

The '*Scored dataset*' produced the following scores:

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
809	355	0.936	0.866	0.5	0.966
False Positive	**True Negative**	**Recall**	**F1 Score**		
125	6249	0.695	0.771		
Positive Label	**Negative Label**				
1	0				

The '*Scored dataset to compare*' produced the following scores:

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
1053	111	0.874	0.852	0.5	0.942
False Positive	**True Negative**	**Recall**	**F1 Score**		
183	981	0.905	0.877		
Positive Label	**Negative Label**				
1	0				

Model Score Assessments

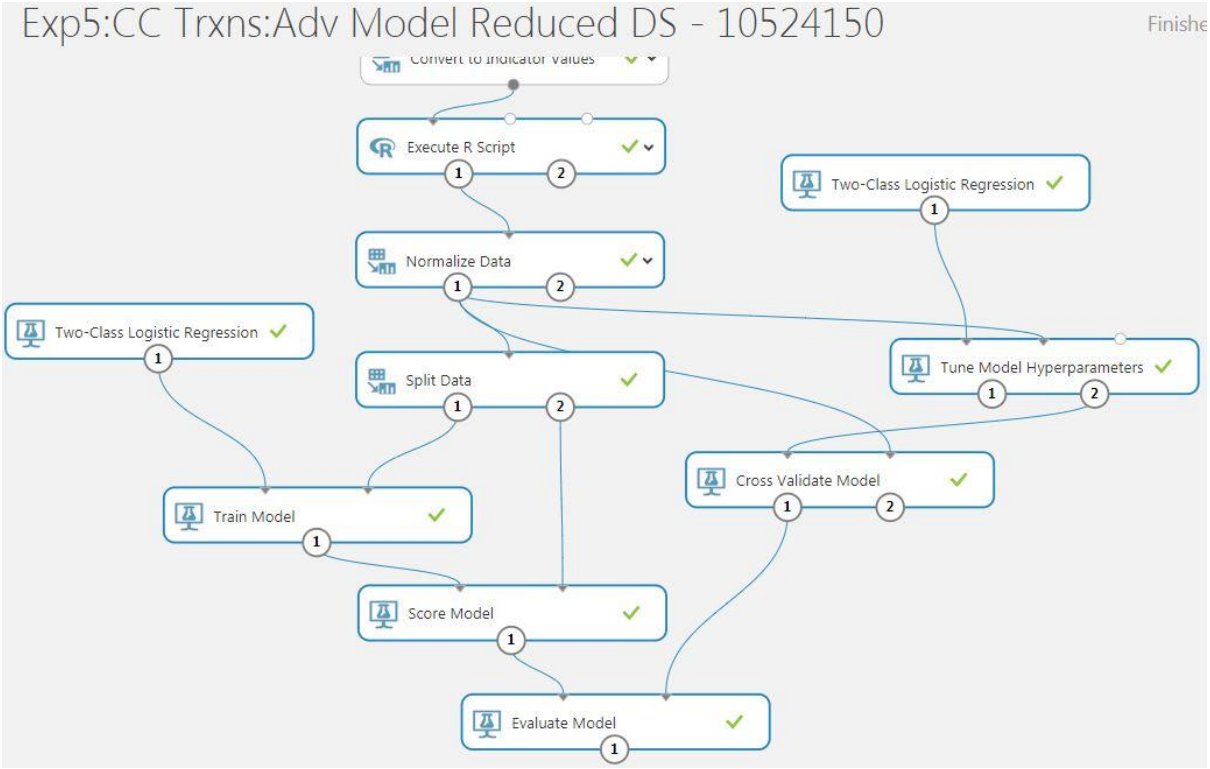
Feature Engineering does not improve the overall accuracy of my credit card predictive model for fraud, but it is much better at detecting actual fraud cases (higher Recall value).

4.4.2. Experiment 5: Model Evaluation with Cross Validation/Hyperparameter Tuning

The figure below illustrates how the Azure Machine Learning Studio (classic) modules were arranged to assess the benefits of using Cross Validation and hyperparameter tuning.

Figure: Experiment 5: Cross Validation and Hyperparameter Tuning

Note: - This image has been deliberate truncated to focus on the modules after Feature Engineering.



Appendix 9.2 of this document details the specific configurations of the 'Tune Model Hyperparameter' and 'Cross Validate Model' modules.

The result of this experiment can be summarised as:

- Experiment 4 conducted a straightforward Test/Train split of the dataset for modelling. Can we determine if Cross Validation will improve the reliability of my predictive model for credit card fraud detection?
- Azure Machine Learning Studio (classic) allows for an automated process to tune the hyperparameter values on an algorithm. Does this also contribute to better fraud prediction for my dataset?

Again, 'Two-Class Logistic Regression' is used because of evaluation results in later experiments feeding back into this 'final' version of Experiment 5.

Model Evaluations

As before, the 'Evaluate Model' module provides takes two inputs and provides the key scoring metrics on comparative models as an output.

In Experiment 5:

- The '*Scored dataset*' was the model generated with Feature Engineering in Experiment 4.

- The '*Score dataset to compare*' was the model generated using Cross Validation on the dataset and tuned hyperparameters for the Two-Class Logistic Regression algorithm.

The '*Scored dataset'* is unchanged from Experiment 4.

The '*Scored dataset to compare*' produced the following scores:

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3639	242	0.926	0.916	0.5	0.973

False Positive	True Negative	Recall	F1 Score
333	3548	0.938	0.927

Positive Label	Negative Label
1	0

Model Score Assessments

Using Cross Validation and hyperparameter tuning in Experiment 5 has produced a model that scores almost as well in 'Accuracy' as the LHS model Experiment 4 (0.936 vs 0.926).

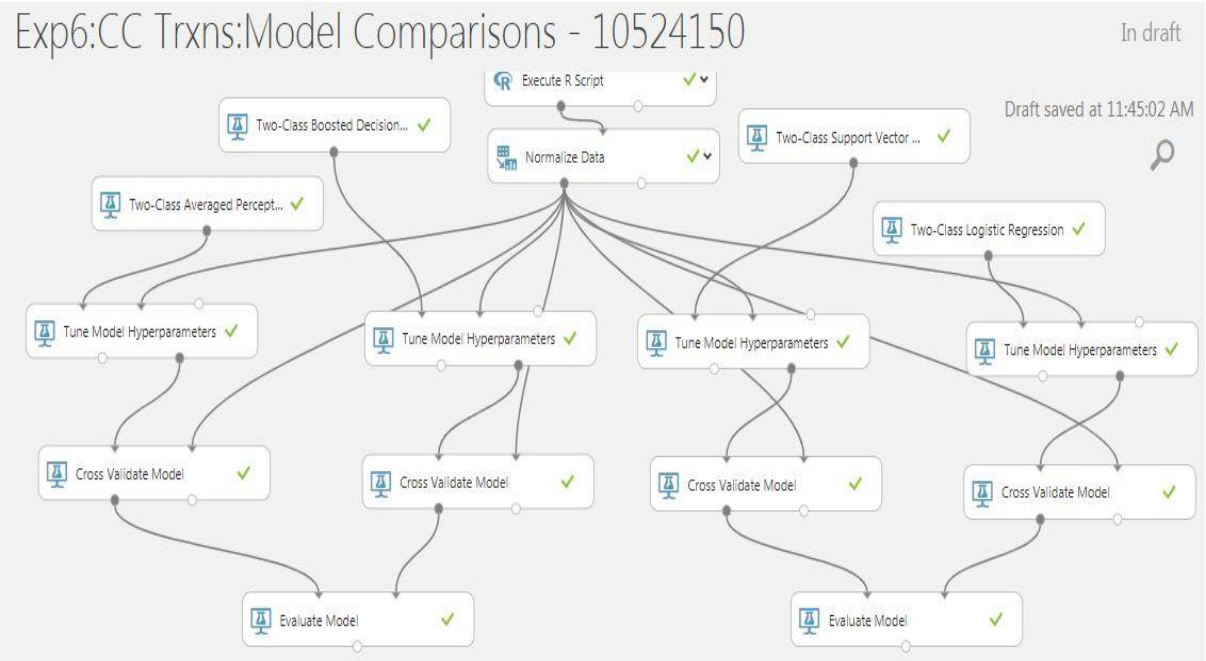
However, the 'Recall' score in Experiment 5 is higher again (0.938) and is thus even better at finding fraud that either of the models in Experiment 5.

4.4.3. Experiment 6: Comparison of Multiple Classification Algorithms (1)

The figure below illustrates how the Azure Machine Learning Studio (classic) modules were arranged to assess the performance of multiple Classification algorithms.

Figure: Experiment 6: Comparing Classification Algorithms

Note: - This image has been deliberate truncated to focus on the modules after Feature Engineering.



Based on the results from Experiment 5, Cross Validation and hyperparameter tuning will be applied to all models built in further experiments to create my credit card predictive model for fraud detection.

The result of this experiment can be summarised as:

- Compare results of four similar 'Two-Class' Classification algorithms when creating a predictive model based on my credit card fraud dataset. The algorithms being compared in this experiment are:
    - Two-Class Averaged Perceptron.
    - Two-Class Boosted Decision Tree.
    - Two-Class Support Vector Machine.
    - Two-Class Logistic Regression.

The selection of classification algorithms in the Azure Machine Learning Studio (classic) is limited to nine options, of which I choose eight. The other classification algorithms specialise in multi-class problems.

Model Evaluations and Assessments

Appendix 9.2 of this document provides a breakdown of all Experiment 6 and 7 evaluation scores for each model.

'Two-Class Logistic Regression' performs best, based on a combination of 'Accuracy' and 'Recall'.

Other observations on the algorithm performances were (based on a 25K row dataset with 39 features):

- The Two-Class Averaged Perceptron algorithm was the quickest to run (< 1 minute) and complete. The Microsoft documentation describes this as a simplified version of a neural network. It is sometimes favoured when the goal is speed over accuracy. (Microsoft, 2019).

- The Two-Class Boosted Decision Tree took the longest to run and complete. Hyperparameter tuning alone took 10+ minutes, and the model was not available for scoring for nearly 20 minutes. The Azure Machine Learning Studio (classic) contained a tutorial recommending this algorithm for client credit risk solution, but performance with my dataset was a concern. (Normalization was probably a redundant step in this modelling process but was left in place for simplicity.)

- The Two-Class Support Vector algorithm took 5+ minutes to complete the modelling process. (The second longest). Microsoft documentation recommends this for simpler datasets where the aim is, again, speed over accuracy. Results were good but performance was slow.

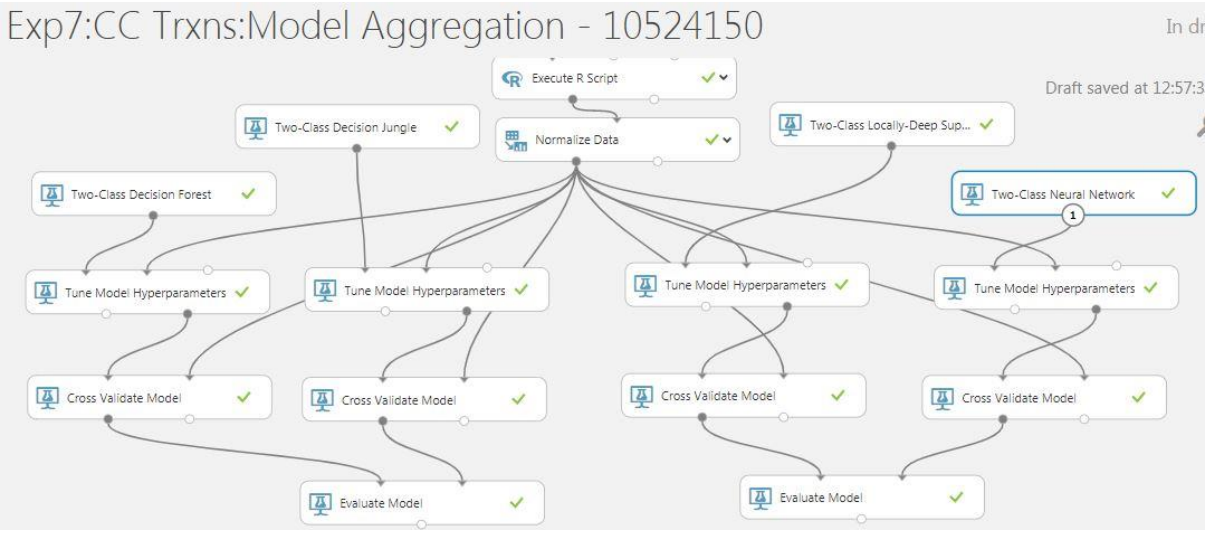
- Two-Class Logistic Regression was dependent on conversion of non-numeric features but performed the best overall.

4.4.4. Experiment 7: Comparison of Multiple Classification Algorithms (2)

The figure below illustrates how the Azure Machine Learning Studio (classic) modules were arranged to assess the performance of further multiple Classification algorithms.

Figure: Experiment 7: Comparing Classification Algorithms – Pt2

Note: - This image has been deliberate truncated to focus on the modules after Feature Engineering.



The result of this experiment can be summarised as:

- Compare results of four similar 'Two-Class' Classification algorithms when creating a predictive model based on my credit card fraud dataset. (). The algorithms being compared in this experiment are:
    - Two-Class Decision Forest.
    - Two-Class Decision Jungle.
    - Two-Class Locally-Deep Support Vector Machine.
    - Two-Class Neural Network.

These are possibly more complex algorithms with greater processing overhead and are included in the project to compare with the group of algorithms in Experiment 6.

Model Evaluations and Assessments

Appendix 9.2 of this document provides a breakdown of all Experiment 6 and 7 evaluation scores for each model.

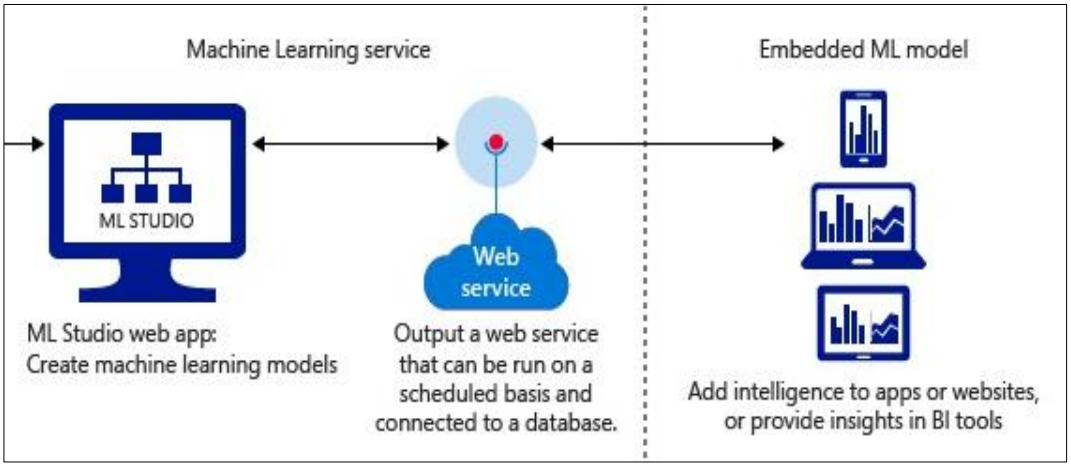
None of the Experiment 7 algorithms generated superior results, in terms of 'Accuracy' and 'Recall' when compared to the Two-Class Logistic Regression based model.

4.5. Credit Card Fraud – Deploying the Azure Model

My iterations through the experiments to evaluate the best algorithm, including the optimum training process, provided me a training model that I now wanted to deploy into production.

This would allow me to host the model in Azure and invoke that model to display fraud predictions on 'new' credit card transactions.

Figure: Reproduced from MicroStrategy Community (Sonobe, 2017)



My objectives, at this stage of the project lifecycle, were to:

- Prepare and validate a 'final' model based on my refined feature engineering routines and training process, using my chosen classification algorithm.
- Create a 'Predictive' version of the trained model, in preparation for the set-up of a Web Service to be hosted in Azure. Deploy this credit card predictive fraud model as a Web Service hosted in Azure and test the deployment with sample data.
- Update the Shiny App UI code with R code that invokes the API to return a real time predictive score on the likelihood of fraud for a given new credit card transaction, selected by the user through the project UI. (See Section 4.5 of this document for details on the code routines to extract key data elements from 'new' card transactions and pass them to the API for the fraud model).

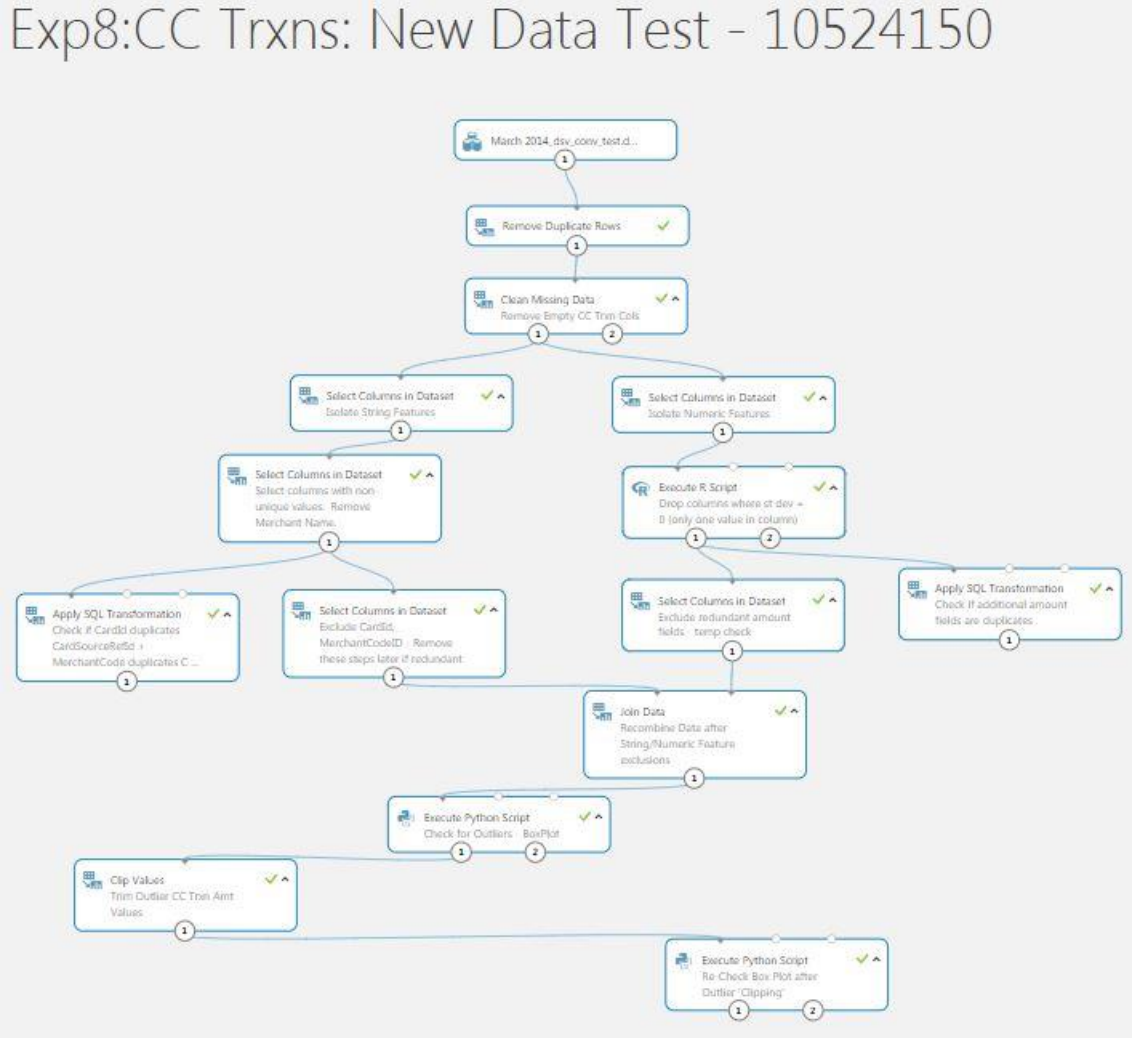
This section of the document details the set-up and execution of the following experiments/actions:

- Experiment 8: Repeat of initial Feature Engineering routines with larger dataset.
- Experiment 9: Creation of 'Predictive' model.
- Deployment and validation of Web Service for trained model.

4.5.1. Experiment 8: Feature Engineering on Larger Dataset

The figure below illustrates how the Azure Machine Learning Studio (classic) modules were arranged to assess process initial Feature Engineering routines on the larger credit card transaction dataset.

Figure: Experiment 8: Feature Engineering on Larger Dataset



Experiment 8 is a re-execution of Experiment 1 but on the larger 25K dataset.

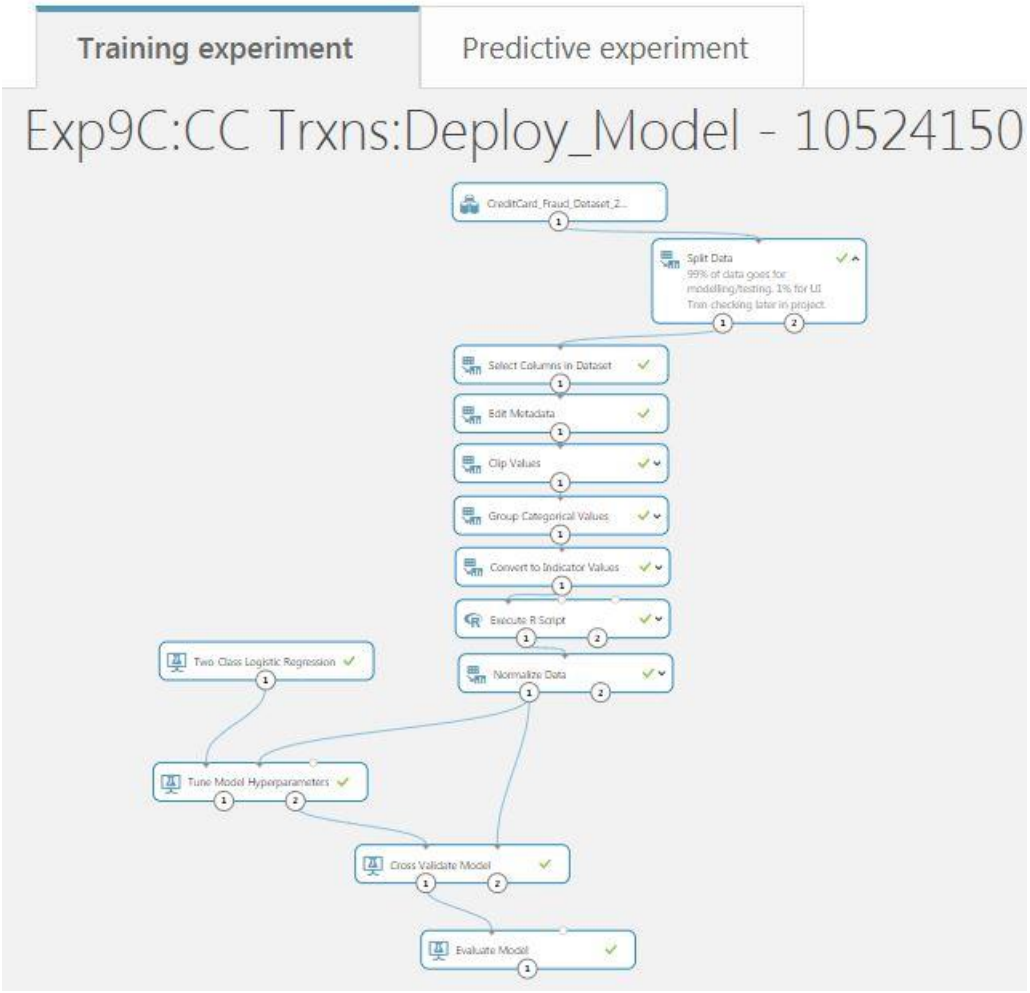
The larger dataset is being introduced at this point in the project to provide a greater volume of data for the training process, and thus ideally increase the reliability of the predictive fraud model.

4.5.2. Experiment 9: Creation of Predictive Fraud Model for Deployment

Experiment 9 is drawn from the results and conclusions from earlier experiments.

The 'Training Experiment' in the illustration below represents the 'final' model creation approach.

Figure: Experiment 9: Training Model



(The '9C' numbering convention is the result of various iterations on this experiment).

The purpose of this Experiment is to create a training model which will then be converted into a 'Predictive' model.

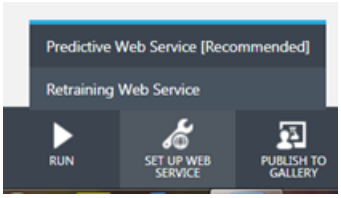
The 'Predictive' model is the basis for the deployment of a Web Service to allow external access (from my Shiny R application) to the scoring model for credit card fraud prediction.

The experiment above is under the 'Training' tab.

Generating a Predictive Experiment

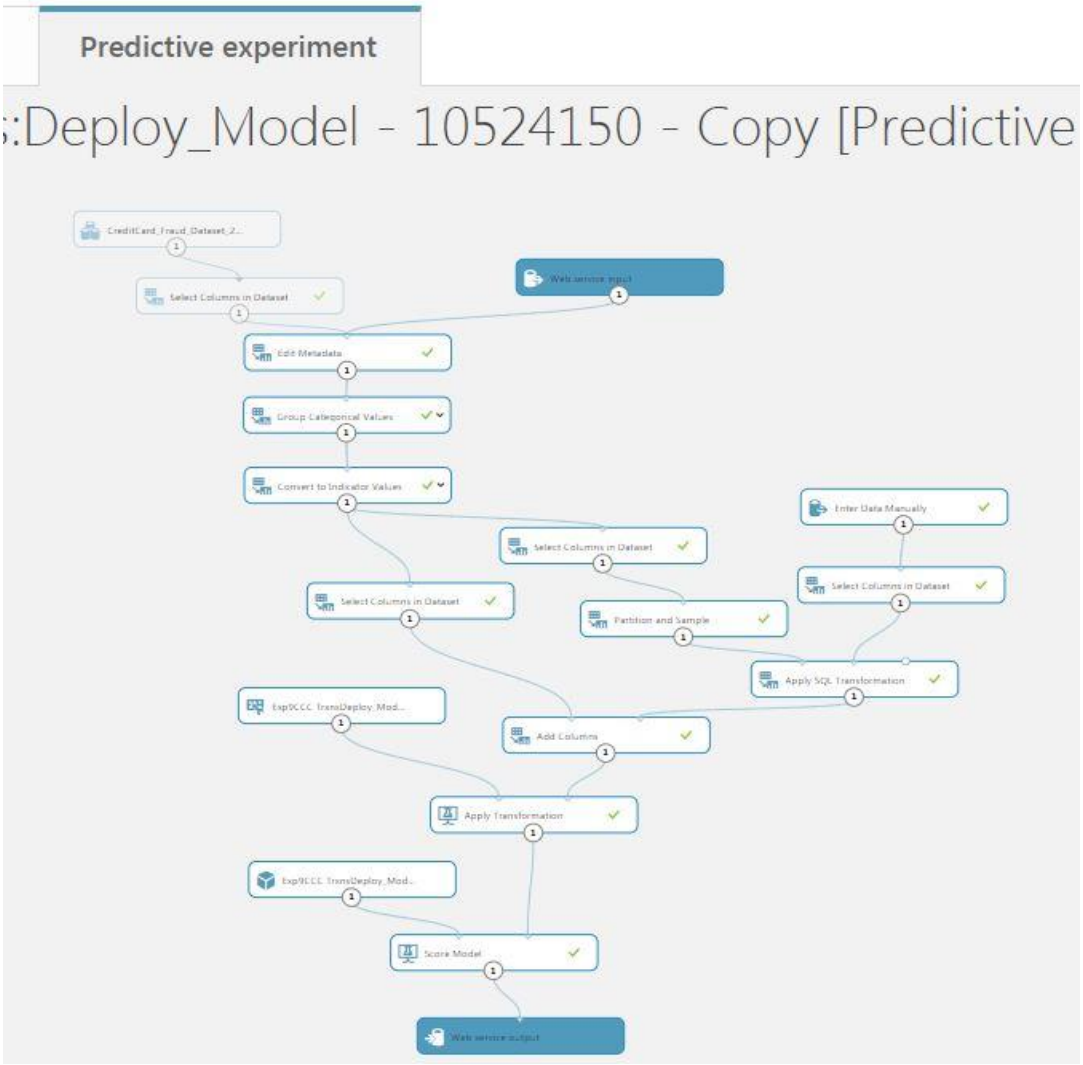
The Azure Machine Learning Studio (classic) provides an option for any experiment with a trained model to be deployed as a Web Service.

Figure: Option to generate Predictive experiment



This creates a 'stripped down' version of the Training experiment called the 'Predictive experiment'.

Figure: Experiment 9: Predictive experiment



Web Service Inputs / Outputs

As some of the remaining modules will be redundant for the real time 'one-by-one' scoring of transactions, which is a key requirement of my credit card fraud prediction project, I have removed other elements of the 'Training experiment' that were brought across.

Key features of the Predictive experiment, as show in the illustration, are:

- Azure Machine Learning Studio (classic) introduces Web Service input and output modules. These determine the interface points to the model in deployment.

- I moved the Web Service input to a point after the Feature Selection module. This is done so that the API code in my Shiny R application will only need to pass the sub-set of 28 features directly required by the module, and not the much larger (post Feature Engineering) dataset.

- I have created additional modules to generate a manual one-hot encoding. This is required because tests failed during verification of the deployed model when transactions were being processed 'one-by-one'. Single credit card transactions would not generate the additional non-numeric features created during the modelling process with the larger dataset.

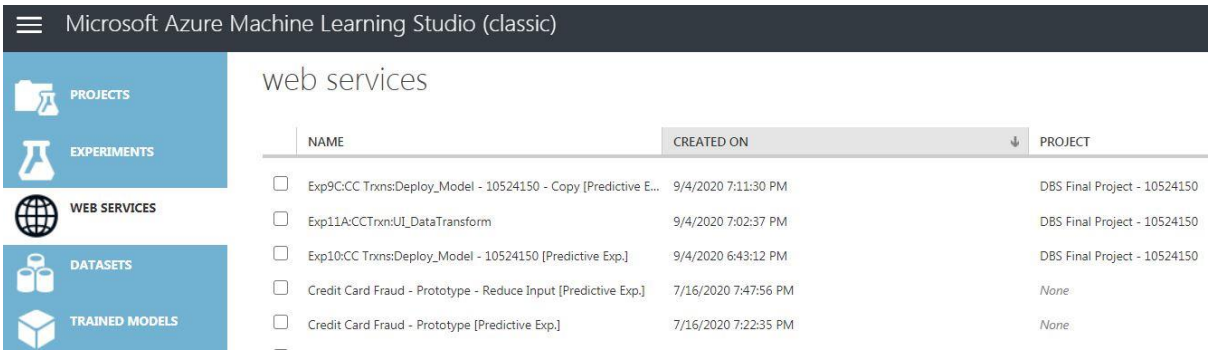
Once validated, the Predictive experiment can be deployed as a Web Services, hosted within Azure.

Figure: Option to Deploy Web Service

4.5.3. Deployment and Validation of Web Service for Predictive Fraud Model

Azure Machine Learning Studio (classic) maintains a list of generated Web Services, which can be accessed through the Studio interface.

Figure: Web Services List

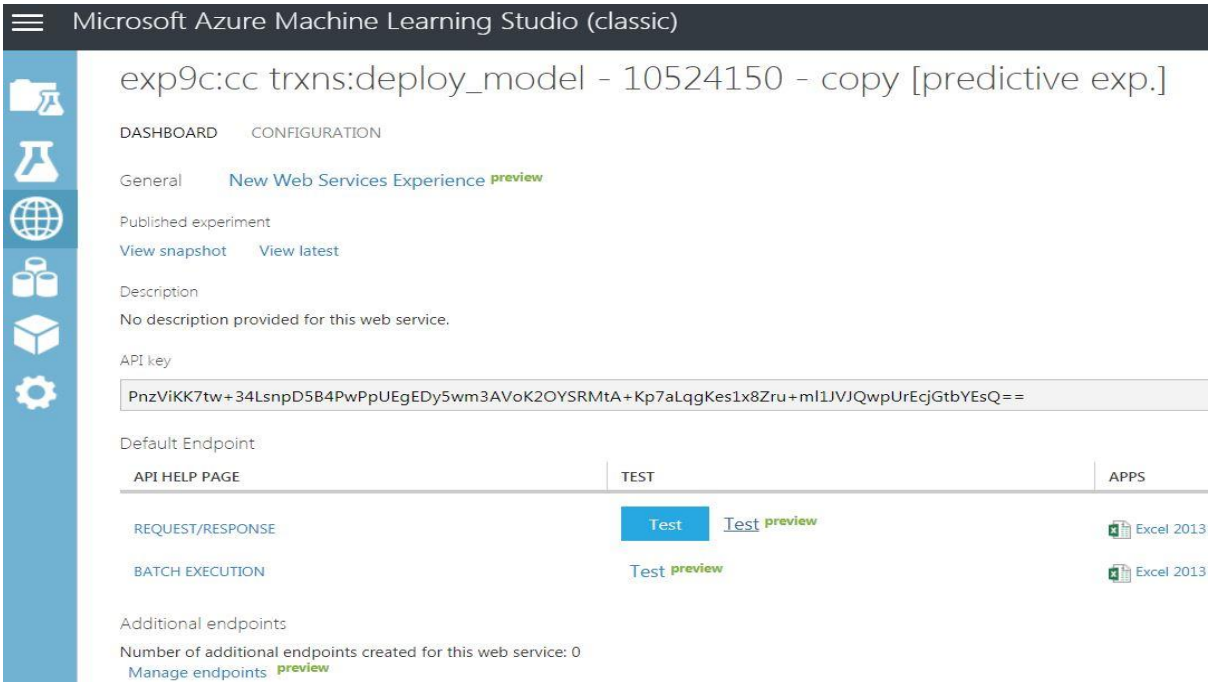


The Web Services that I have generated in my project as part of ongoing research, for the Interim Prototype, and for the final predictive credit card fraud model can be seen in this illustration above.

How to validate the Web Service through Azure?

The generation process for a Web Service, if successful, brings the user to a dashboard screen. The illustration below shows the dashboard screen for my final production model for credit card fraud prediction.

Figure: Web Services Dashboard



Section 5 of this document will explain more about how this Web Service is consumed but a key element on this page is the API key through which the model can be invoked externally.

API key

PnzViKK7tw+34LsnpD5B4PwPpUEgEDy5wm3AVoK2OYSRMtA+Kp7aLqgKes1x8Zru+ml1JVJQwpUrEcjGtbYEsQ==

Before I began writing R code to access the API for the fraud model, I needed to verify that the Web Service was working as expected and returning a score for predicting fraud on my credit card transaction.

Azure Machine Learning Studio provides a separate Web Services harness to test and manage these hosted endpoints.

Using this portal, shown in the illustrations below, I was able to validate my credit card fraud predictive model was working correctly.

Figure: Azure ML Studio Web Services Portal – Main Screen

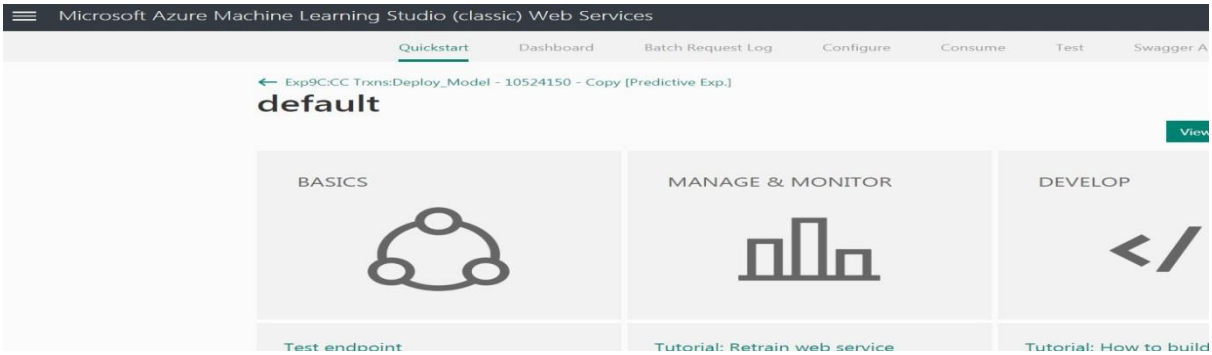
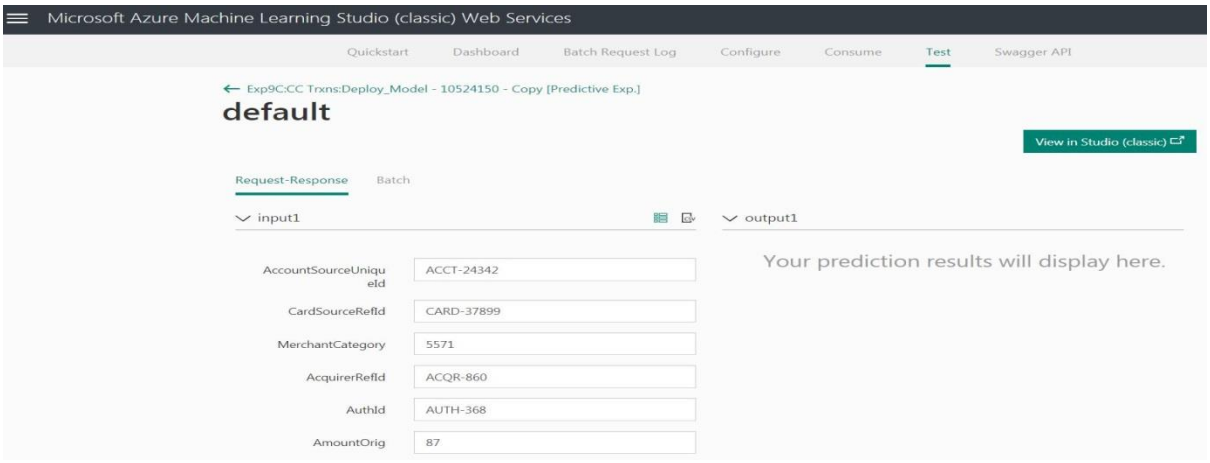


Figure: Azure ML Studio Web Services Portal – Test Screen



Test results return a 'Score Label' – '1' for Fraud, '0' for Non-Fraud. A Scored Probability value is also returned, which is a number between 0 and 1 (> 0.5 = Fraud, <0.5 = Non-Fraud).

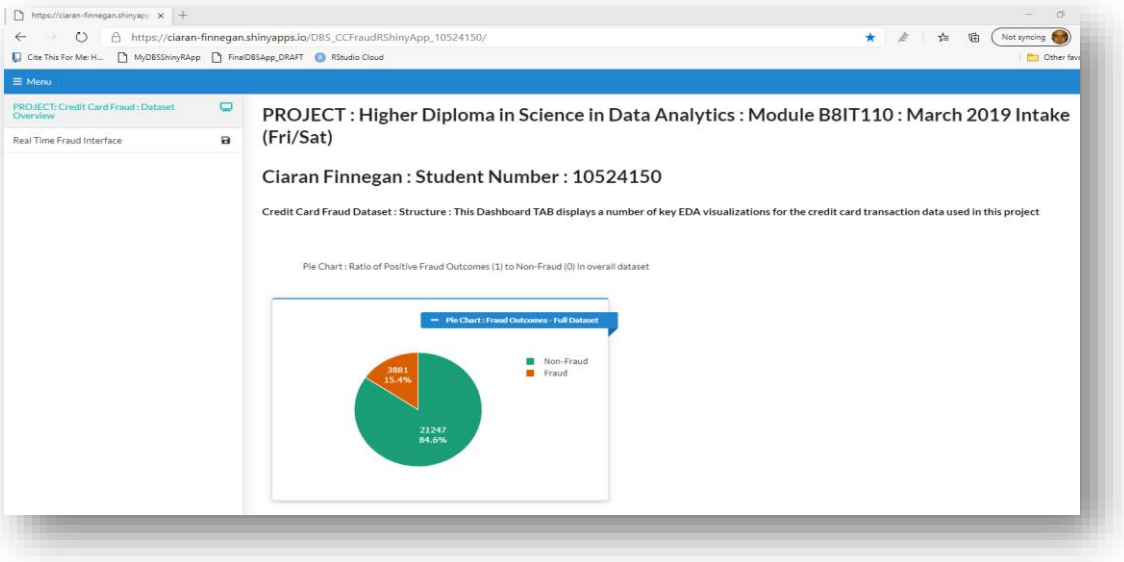
5. Project Implementation (2) – Shiny R Dashboard UI

5.1. Data Visualisations in a Shiny Dashboard

The project user interface is a Shiny R Dashboard application that launches from URL:

https://ciaran-finnegan.shinyapps.io/DBS_CCFraudRShinyApp_10524150/

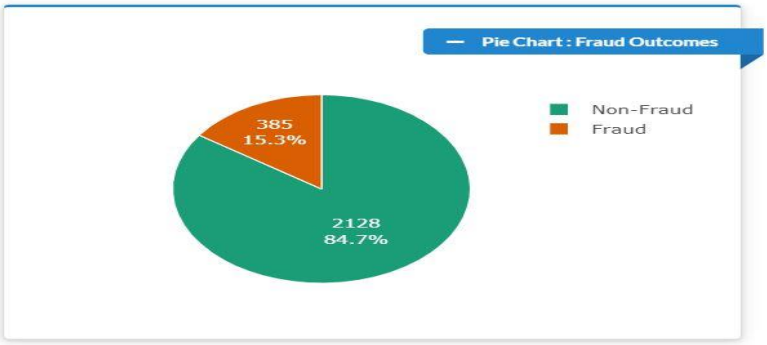
Figure: Initial View of project application for Credit Card Fraud Analysis / Detection



The first tab of the dashboard contains graphical analysis of key features of the credit card dataset used to train the predictive fraud model invoked by the application.

5.1.1. Graph 1: Balance of Fraud in Dataset

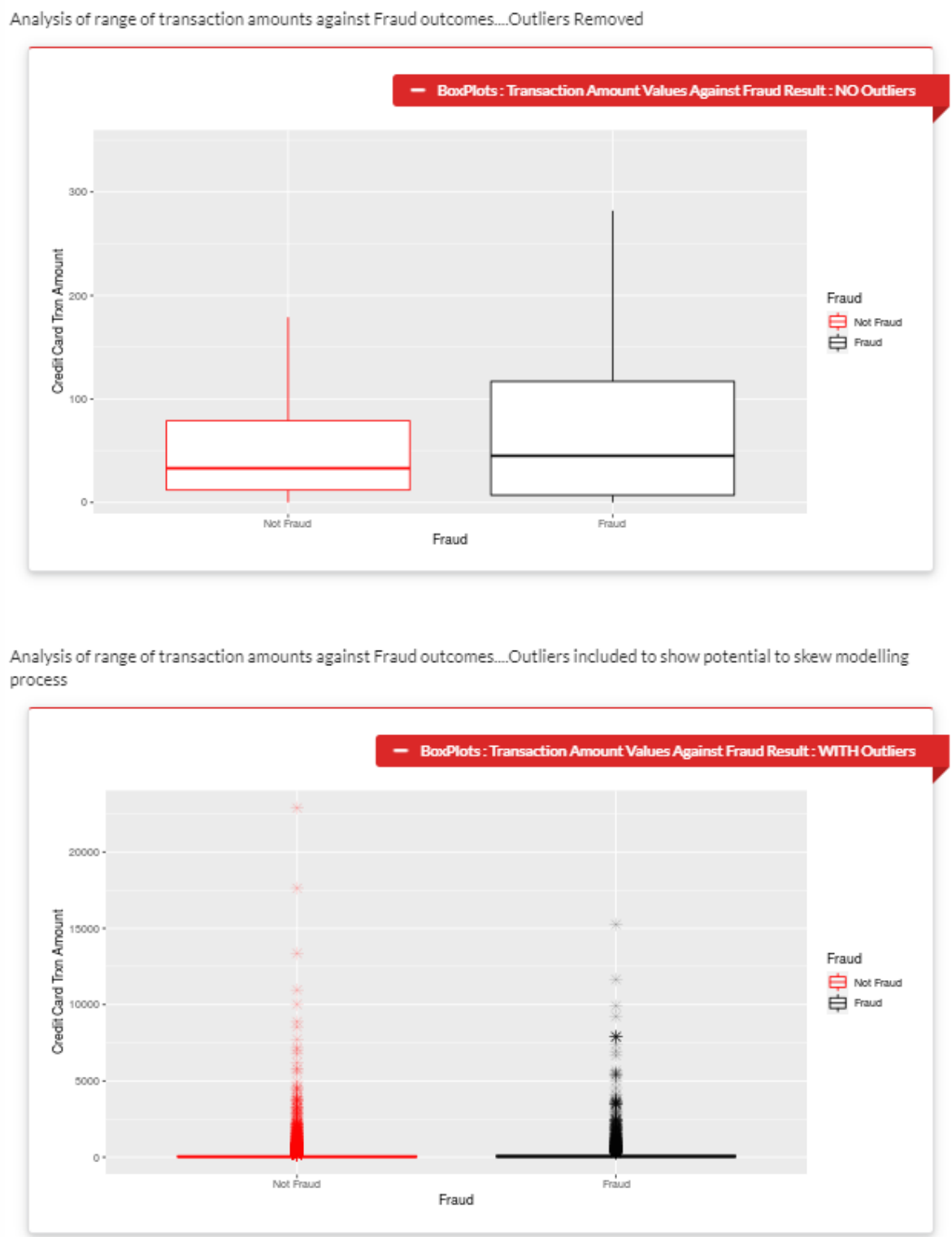
Figure: Pie Chart showing proportion of records in dataset that represent fraud.



Just over 15% of the rows in the credit card dataset represent transactions that were shown to be fraudulent.

5.1.2. Graph 2: Box Plot Analysis of the Dollar Amount of CC Transactions

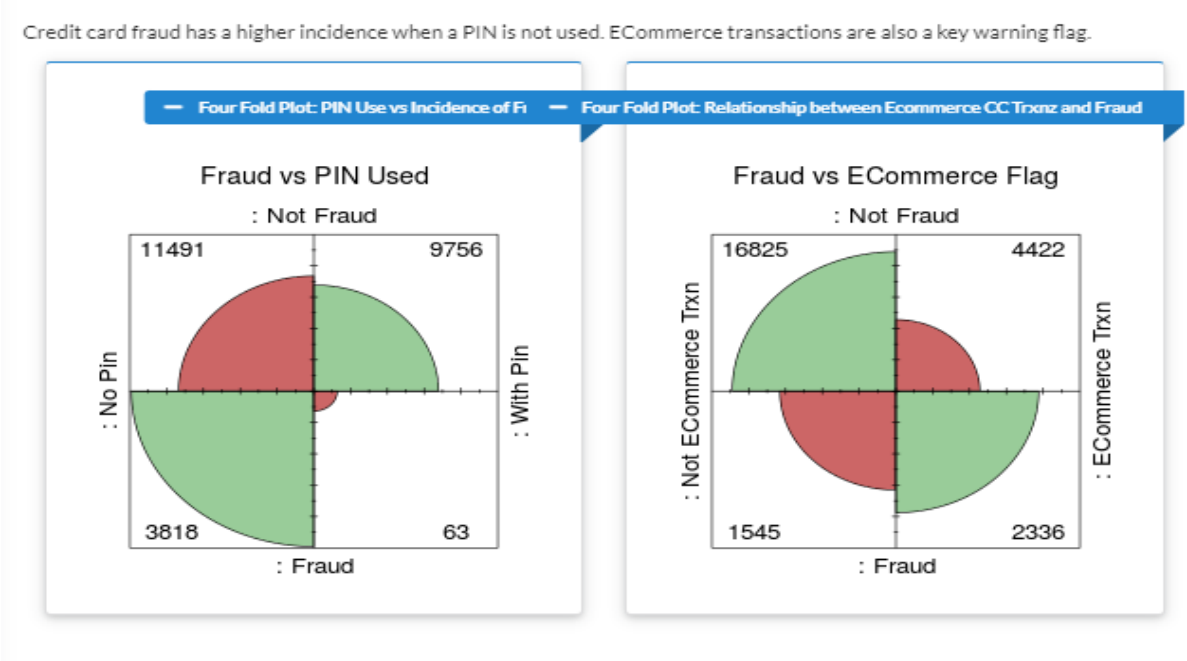
Figure: Box Plots of Transaction Amounts



The box plots show that the majority of transaction amounts ($) in the dataset are in the early 100s. There are also some outlier values in the thousands.

5.1.3. Graph 4: PIN Used and Ecommerce Flag

Figure: Four-Fold Plot Showing Influence of 'PIN Used' and 'Ecommerce Flag' on Fraud.



Not surprisingly the use of a PIN code is less likely to be associated with a fraudulent credit card transaction.

5.1.4. Graph 5: Customer Not Present Table

Figure: Table Showing Relationship of Customer Presence to Fraud.

The physical present of a customer at the point of transaction had an influence on the potential for credit card fraud

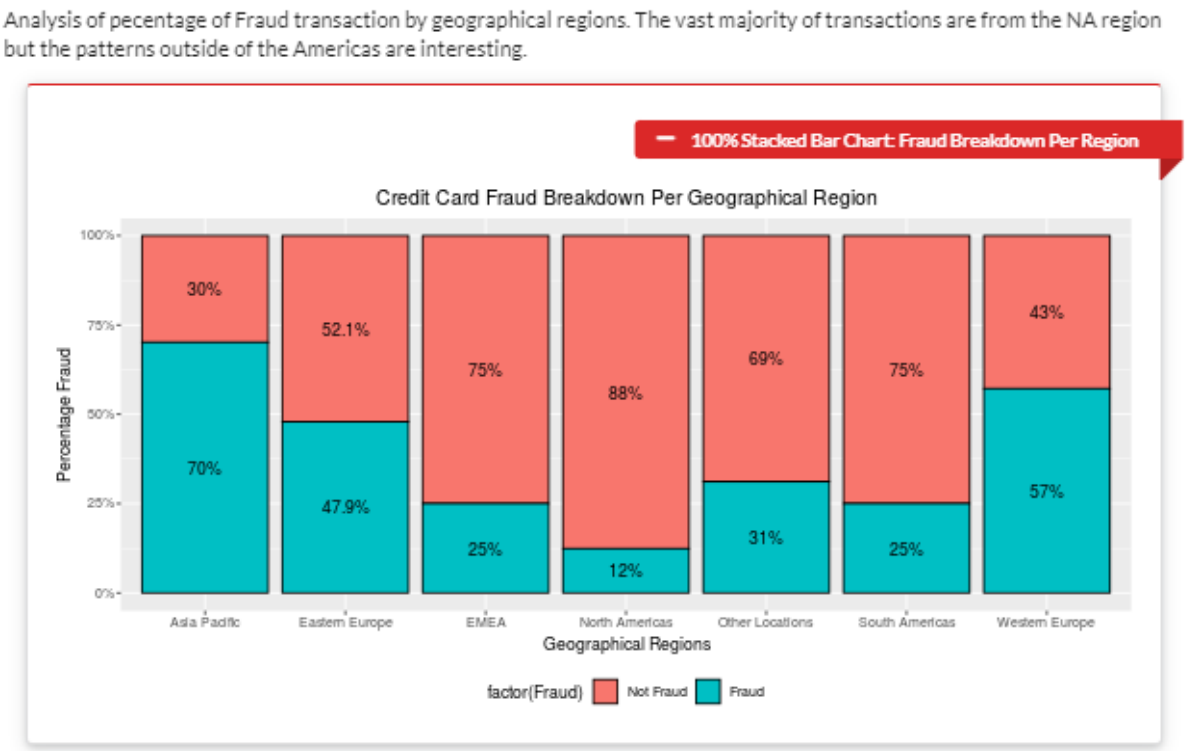
Table: Relationship of Fraud to Customer Presence at CC Transaction

Fraud?	Is Customer Present	No of Trxnz
Not Fraud	No Customer	5349
Not Fraud	Unknown	2375
Not Fraud	Customer Present	13523
Fraud	No Customer	2456
Fraud	Unknown	629
Fraud	Customer Present	796

Customer Presence is less likely to be associated with a fraudulent credit card transaction.

5.1.5. Graph 5: Balance of Fraud in Dataset

Figure: 100% Stacked Bar Chart Showing Proportions of Fraud Transactions Regionally



In the dataset the vast majority of transactions are from the North American region but the patterns outside of the Americas are still interesting.

5.1.6. How are the Shiny Dashboard Graphs Created?

The user interface for this application is a Shiny R application that uses several libraries to render the dashboard and the embedded graphs.

A full breakdown of the R source code is provided in Section 9.1 of the Appendices to this document.

Taking the 100% Stacked Bar Chart as an example, the implementation steps can be briefly explained as:

- The '*app.r*' file is the main body of the application. It contains a separate 'ui' and 'server' function that provide the framework for most of the dashboard.

Figure: Sample Code Snippets

```
# Shiny UI Function

ui <- dashboardPage(


  dashboardHeader(color = "blue", title = "PROJECT : Data Analytics : B8IT110", inverted = TRUE),

  dashboardSidebar(
    size = "wide", color = "teal",
    sidebarMenu(
      menuItem(tabName = "dataviz", text = "PROJECT: Credit Card Fraud : Dataset Overview", icon = icon("tv")),
      menuItem(tabName = "fraud_interface", text = "Real Time Fraud Interface", icon = icon("save"))
  )),

  dashboardBody(
    tabItems(
      selected = 1,
```

```
# Shiny Server Function

server <- function(input, output, session) {

  ## Call function to activate file selection dialog box on 2nd Tab
  ## This allows the user to select a given file with a set of 'new' transactions
  fCCTrxn_datafromfile <- csvFileServer("NewCC_Trxn_datafile", stringsAsFactors = FALSE)


  ## Set up dataselect routines for user interactions on 2nd tab
  v <- reactiveValues()
  v$s <- NULL



  ##############################################################################################
  ## ---    1st Tab Functions
  ##############################################################################################

  # Generate Pie Chart to show proportion of Fraud and non-Fraud transactions in the dataset
  output$plot1 <- renderPlotly({
```

- The 100% Stacked Bar Chart is positioned on the dashboard within the Shiny 'ui' function code for the 1st tab.

```
fluidRow(
  htmlOutput("text.8"),
  box(
    width = 16,
    title = "100% Stacked Bar Chart: Fraud Breakdown Per Region",
    color = "red",
    ribbon = TRUE,
    title_side = "top right",
    column(
      14,
      plotOutput("plot6", height = 350)
    )
  )
)
```

- The 'server' code for the dashboard generates the actual graph details, which are passed to the user interface using the '*plot6*' identifier. (The initial code segments below are functions that prepare the dataset to be passed as a parameter to the graph building code).

```
# Read Azure Hosted Credit Card Fraud Dataset for Geographical Analysis of Fraud in dataset
frdCntReport = api_call$AZURE_CreditCardFraud_Download$downloadDevCntryCdCCTxns()
##############################################################################################
```

```
# Generate 100% Stacked Bar Chart with Proportion of Fraud Per Region
output$plot6 <- renderPlot({

  ## Call function and return plot to UI section
  frdGeoBars <- fGetPlot_frdGeo(frdCntReport)
  frdGeoBars

})
```

- The graph is generated using the appropriate R library and passed back to the Shiny *app.r* code to be rendered in the dashboard.

```
## Generate and return a 100% stacked bar chart that shows the percentage of Fraud/non-Fraud
## across the different geographical regions in the Credit Card dataset
fGetPlot_frdGeo = function(ds) {


  # Label the fraud column values to improve the graph display
  ds$Fraud <- as.factor(ds$Fraud)
  levels(ds$Fraud) <- c("Not Fraud", "Fraud")   # Fraud


  # Group the data by geographical region and set up the percentage information
  frdPercentData <- ds  %>% group_by(DeviceCountryCode) %>% count(Fraud) %>%
    mutate(ratio=scales::percent(n/sum(n)))


  # Generate graph
  p1 <- ggplot(ds, aes(x=DeviceCountryCode, fill = factor(Fraud))) +
    geom_bar(color = "black", position = "fill") +
    geom_text(data = frdPercentData, aes(y=n, label=ratio),
              position = position_fill(vjust = 0.5)) +
    scale_y_continuous(labels = percent_format()) +
    labs(y="Percentage Fraud") +
    scale_x_discrete("Geographical Regions") +
    ggtitle("Credit Card Fraud Breakdown Per Geographical Region") +
    theme(plot.title = element_text(hjust = 0.5),
          legend.position = "bottom", legend.direction = "horizontal")

  p1

}
```

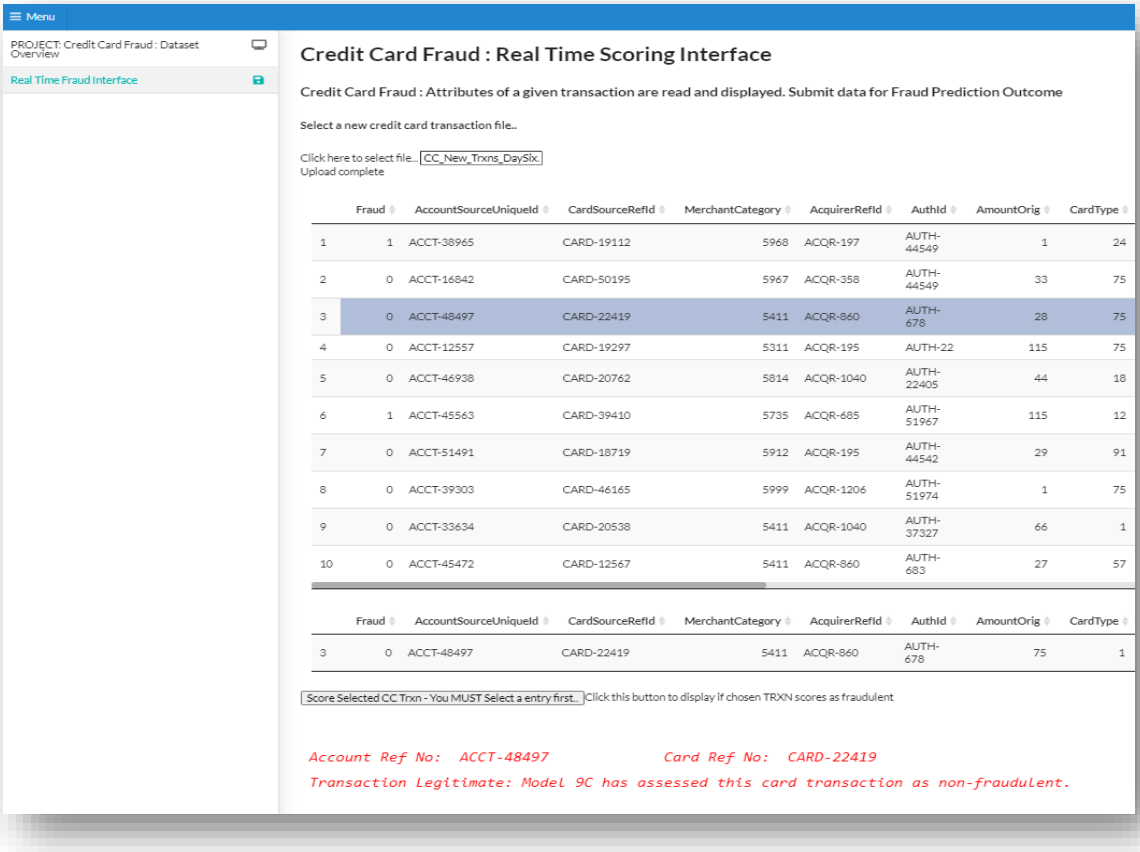
5.2. Credit Card Fraud – UI to Check Fraud Predictions

5.3.1. The User Interface for Fraud Detection

The second tab on the application is the interface that allows the user to:

- Load .csv files that contain 'new' credit card transactions.
- Display the individual records in these csv files.
- Select a single record and invoke the API to the Azure hosted predictive fraud model.
- Obtain a result in real time from the production model that indicates if the chosen transaction is likely to be fraudulent.

Figure: Fraud Detection UI



	Fraud	AccountSourceUniqueId	CardSourceRefId	MerchantCategory	AcquirerRefId	AuthId	AmountOrig	CardType
1	1	ACCT-38965	CARD-19112	5968	ACQR-197	AUTH-44549	1	24
2	0	ACCT-16842	CARD-50195	5967	ACQR-358	AUTH-44549	33	75
3	0	ACCT-48497	CARD-22419	5411	ACQR-860	AUTH-678	28	75
4	0	ACCT-12557	CARD-19297	5311	ACQR-195	AUTH-22	115	75
5	0	ACCT-46938	CARD-20762	5814	ACQR-1040	AUTH-22405	44	18
6	1	ACCT-45563	CARD-39410	5735	ACQR-685	AUTH-51967	115	12
7	0	ACCT-51491	CARD-18719	5912	ACQR-195	AUTH-44542	29	91
8	0	ACCT-39303	CARD-46165	5999	ACQR-1206	AUTH-51974	1	75
9	0	ACCT-33634	CARD-20538	5411	ACQR-1040	AUTH-37327	66	1
10	0	ACCT-45472	CARD-12567	5411	ACQR-860	AUTH-683	27	57

	Fraud	AccountSourceUniqueId	CardSourceRefId	MerchantCategory	AcquirerRefId	AuthId	AmountOrig	CardType
3	0	ACCT-48497	CARD-22419	5411	ACQR-860	AUTH-678	75	1

The User Guide that accompanies this report provides a visual guide to fraud detection process.

5.3.2. How does the application access the production model?

The following code snippets describe the high-level implementation of how the Shiny R dashboard communicates with the predictive model hosted on Azure.

A full breakdown of the R source code is provided in Section 9.1 of the Appendices to this document.

The User Guide that accompanies this report provides a visual guide to fraud detection process.

1 - Loading the transaction files

```
## Call function to activate file selection dialog box on 2nd Tab
## This allows the user to select a given file with a set of
## 'new' transactions
fCCTrxn_datafromfile <- csvFileServer("NewCC_Trxn_datafile",
                                      stringsAsFactors = FALSE)
```

Call the file open dialog from the 'server' function in *app.r*.

2 – Populate the data table on the second tab on the Shiny dashboard

```
## Populate user interface with the transaction read
## from a chosen csv file
output$credit_card_file_txns <- DT::renderDataTable({
  datatable(fCCTrxn_datafromfile(),
            selection = "single",
            extensions = 'FixedColumns',
            options = list(
              autoWidth = TRUE,
              dom = 't',
              scrollX = TRUE,
              fixedColumns = TRUE
            )
  )
})
```

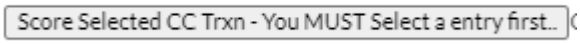
The content of the CSV file is loaded and displayed on the dashboard.

3 – Populated the 'Selected Transaction' row

```
## When a transaction in the table list is clicked update
## the single row 'selected transaction' table
observe({
  if(!is.null(input$credit_card_file_txns_rows_selected)){

    v$s <- input$credit_card_file_txns_rows_selected

  }
})

# Update the single row 'selected transaction' table
output$selected_cc_trxn <- DT::renderDataTable({
  datatable({
              dataTableProxy(outputId = 'selected_cc_trxn') %>%
                hideCols(hide = columns2hide)
                fCCTrxn_datafromfile()[v$s,]
            },
            options = list(dom = 't')
           )
})
```

When the user clicks on an entry in the first data table a separate single line table is populated to confirm the transaction the user wishes to assess for fraud.

4 – Click the button to check the transaction for fraud




```
## Display a description for the user if the selected transaction
## if possibly fraudulent or not
idText <- eventReactive(input$apiTxnRow, {


  # Call function to prepare list of attributes for the API call
  # to the predictive Fraud model
  api.arg.list.score <- get.arg.list.from.cctrxn()

  # Indicate that only the model score is required
  api.arg.list.score <- append(api.arg.list.score, "Score_Message")

  # Pass Parameters for API Call - Full Production Model
  do.call(api_call$AZURE_9C_CCFraud_APICall$Print9CFraudModelResult,
          api.arg.list.score)


})

# Display CC TXN Score
output$cctxn_id <- renderPrint({
  idText()
})
```

```
## Function to parse CC Trxn Row into parameters for API call
get.arg.list.from.cctrxn <- function(){

  # Prepare parameter list of given credit card transaction
  chosen_cc_trxns <- fCCTrxn_datafromfile()[v$s,]
  len_list <- length(chosen_cc_trxns)
  arg.list <- list()

  # Start reading cc trxn record on first entry in row
  i <- 1
  while(i<(len_list+1)){  # This Reflects the number of attribute
    # entries in the cc_trxn record

    arg.list <- append(arg.list, chosen_cc_trxns[1,i])
    i <- i + 1

  }

  return(arg.list)

}
```

When the button is clicked the features of the transaction are read and passed as parameters to the function calling the API to the fraud model.

5 – Invoke the model through the API for the Azure hosted Web Service

(These are code snippets from the R code base. ***Some of the API code has been autogenerated by Microsoft Azure Machine Learning Studio (classic) but requires rework to support dynamic submission of new parameters.***)

```
## The response output is parsed based on a control flag.
## One button in the UI only requires a outcome description.
## One button in the UI only requires a model score
## Another button option returns the full model output.
Print9CFraudModelResult = function(
                                   Fraud,       #  'Fraud',
                                   AccSrcUID,   #  'AccountSourceUniqueId',
                                   CrdSrcRID,   #  'CardSourceRefId',
                                   MerchCat,    #  'MerchantCategory',
                                   AcqRID,      #  'AcquirerRefId',
                                   AuthId,      #  'AuthId',
                                   Amount,      #  'AmountOrig',
                                   CardType,    #  'CardType',
                                   DvcVrfCap,   #  'DvcVerificationCap',
                                   CustPresInd,#  'CustomerPresentIndicator',
                                   PosTrmAttd,  #  'PosTerminalAttended',
                                   DeviceZone,  #  'DeviceZone',
                                   DeviceCntCd,#  'DeviceCountryCode',
                                   ECommerceFlag,  #  'ECommerceFlag',
                                   PinIndicator,   #  'PinIndicator',
                                   HRkPOSCtHPres,  #  'HighRiskPOSCnt.cnt.hour.present',
                                   FPmpCntDtTot,   #  'FuelPumpCount.cnt.day.total',
                                   FPmpCntDtPres,  #  'FuelPumpCount.cnt.day.present',
                                   NtECommAtAmtP3,#  'NotECommerceAuthAmount.acc.day.past3',
                                   NEMVTrxnCntDy29, #  'NonEMVTransactionsCount.cnt.day.past29',
                                   PosTrmAttdACntDy3, #'POSTerminalAttendedAuthCount.cnt.day.past3',
                                   DomAthCnt1,      #  'DomesticAuthCount.cnt.hour1',
                                   NtECommAtAmtPres,#  'NotECommerceAuthCount.cnt.day.present',
                                   EMVTxCntPres,    #  'EMVTransactionsCount.cnt.day.present',
                                   EMVTxCntDy3,     #  'EMVTransactionsCount.cnt.day.past3',
                                   POSCntPres,      #  'POS_Count.cnt.day.present',
                                   EMVTxCntDy1,     #  'EMVTransactionsAcc.acc.day.past1',
                                   AltCntSpCntDy1,  # 'AlternatingCountrySwapCounter.cnt.day.past1'
                                   ReqResp          # Control flag to parse output
                                   )
```

```
## The key features from the on screen credit card transaction are formatted into a list to be passed to the Web Service
## hosting the production model.
req =  list(
  Inputs = list(
    "input1"= list(
      list(
        'AccountSourceUniqueId' = AccSrcUID,
        'CardSourceRefId' = CrdSrcRID,
        'MerchantCategory' = MerchCat,
        'AcquirerRefId' = AcqRID,
        'AuthId' = AuthId,
        'AmountOrig' = Amount,
        'CardType' = CardType,
        'DvcVerificationCap' = DvcVrfCap,
        'CustomerPresentIndicator' = CustPresInd,
        'PosTerminalAttended' = PosTrmAttd,
        'DeviceZone' = DeviceZone,
        'DeviceCountryCode' = DeviceCntCd,
        'ECommerceFlag' = ECommerceFlag,
        'PinIndicator' = PinIndicator,
        'Fraud' = "0", # Not used in modeling process
        'HighRiskPOSCnt.cnt.hour.present' = HRkPOSCtHPres,
        'FuelPumpCount.cnt.day.total' = FPmpCntDtTot,
        'FuelPumpCount.cnt.day.present' = FPmpCntDtPres,
        'NotECommerceAuthAmount.acc.day.past3' = NtECommAtAmtP3,
        'NonEMVTransactionsCount.cnt.day.past29' = NEMVTrxnCntDy29,
        'POSTerminalAttendedAuthCount.cnt.day.past3' = PosTrmAttdACntDy3,
        'DomesticAuthCount.cnt.hour1' = DomAthCnt1,
        'NotECommerceAuthCount.cnt.day.present' = NtECommAtAmtPres,
        'EMVTransactionsCount.cnt.day.present' = EMVTxCntPres,
        'EMVTransactionsCount.cnt.day.past3' = EMVTxCntDy3,
        'POS_Count.cnt.day.present' = POSCntPres,
        'EMVTransactionsAcc.acc.day.past1' = EMVTxCntDy1,
        'AlternatingCountrySwapCounter.cnt.day.past1' = AltCntSpCntDy1
      )
    )
  ),
  GlobalParameters = setNames(fromJSON('{}'), character(0))
)

body = enc2utf8(toJSON(req))

## Provide the unique API key for the credit card fraud web service
api_key = "PnzViKK7tw+34LsnpO5B4PwPpUEgEDy5wm3AVoK2OYSRMtA+Kp7aLqgKes1x8Zru+ml1JVJQwpUrEcjGtbYEsQ=="
authz_hdr = paste('Bearer', api_key, sep=' ')

## Provide the location of the Azure Workspace in which the model is deployed
response=POST(url = "https://europewest.services.azureml.net/workspaces/7375a43f744940748c691a78945b2588/services/979195811821437096 62bcdb1fb4eeb3/execute?api-version=2.0&format=swagger",
              add_headers('Content-Type' = "application/json", 'Authorization' = authz_hdr),
              body=body)

result = content(response, type="text", encoding="UTF-8")
```

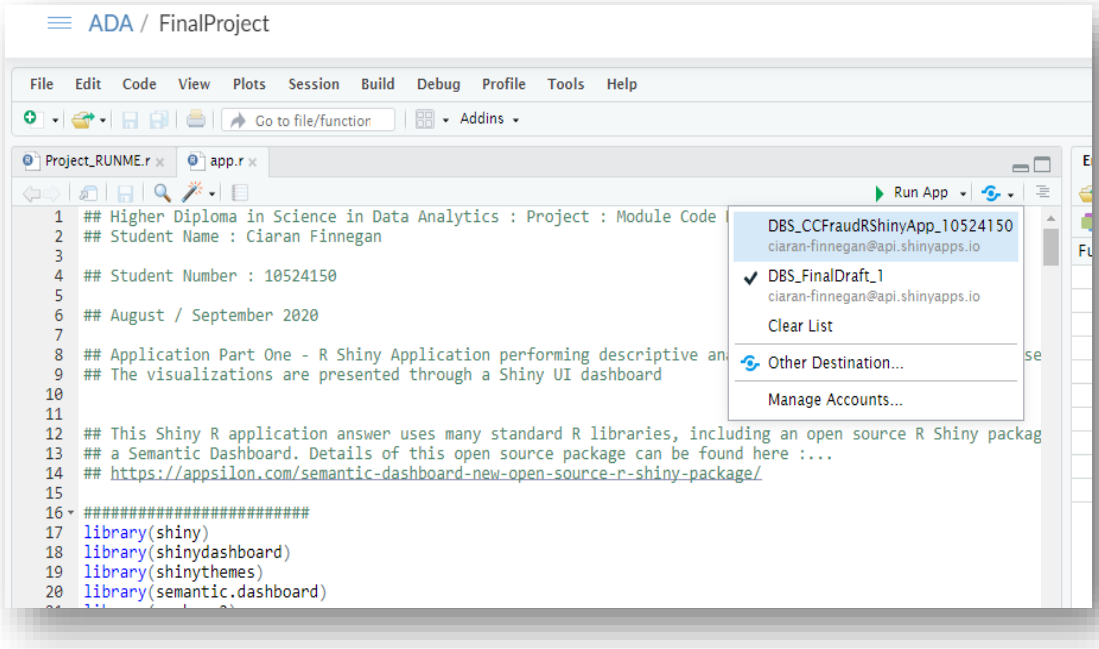
The response from the Web Service is parsed and returned to be rendered on the dashboard.

5.3. Shiny UI – Hosted Application

The application was built as a RStudio Cloud project and then hosted online as a ShinyIO application.

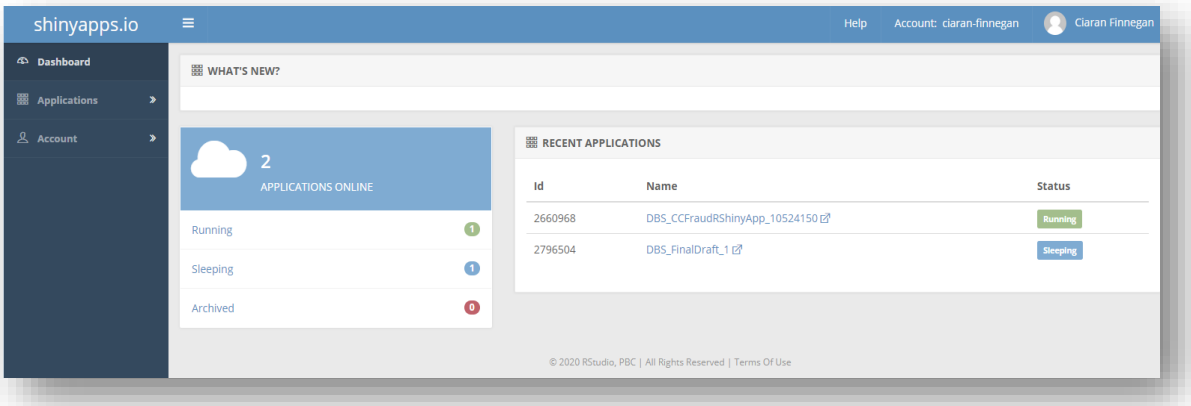
When the *app.r* file is open in RStudio Cloud, an option to 'publish' the application as a ShinyIO application is available by selecting the icon:

Figure: Publishing the application to ShinyIO



This allows the application to be uploaded and hosted on the ShinyIO online service.

Figure: The ShinyIO Dashboard



6. Testing and Results

6.1. User Story 'Demos' – Test Results and 'Feedback'

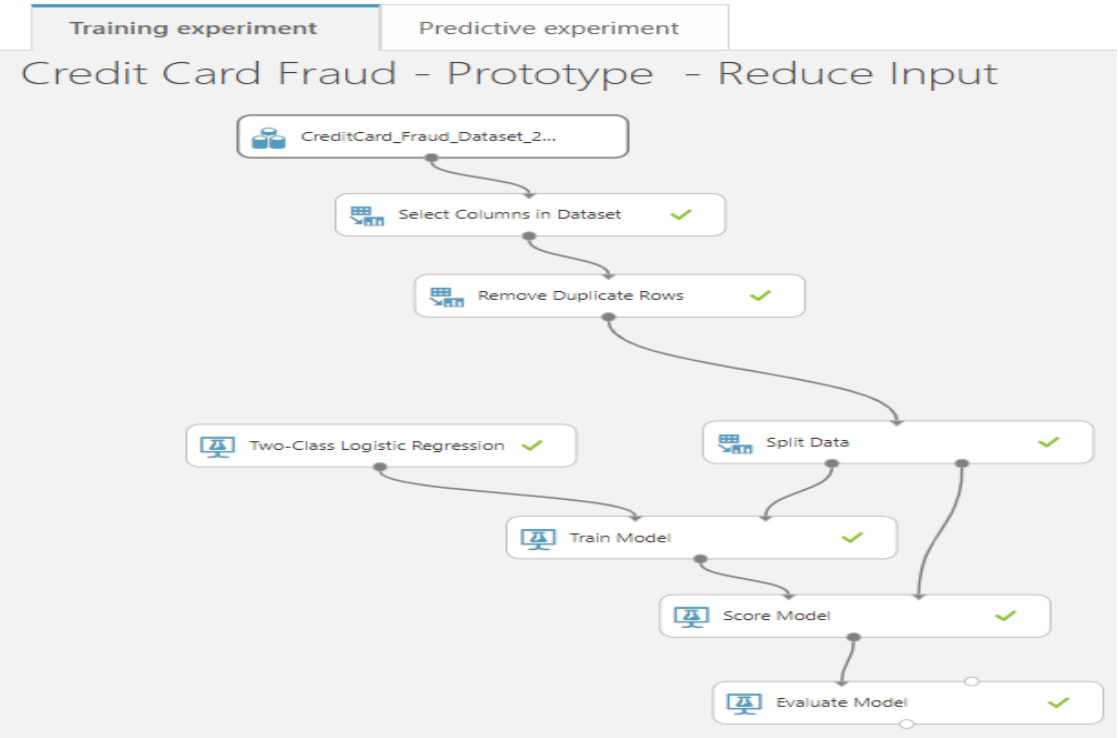
6.1.1. User Story 4: Initial Data Modelling – Review and Evaluation

Goal: *Build a basic credit card fraud predictive model in Azure ML Studio (classic) based on a small subset of transactions dataset*.

Assessment of robustness of code and functionality delivered:

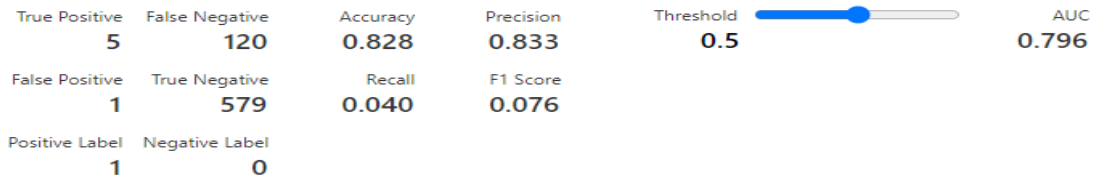
1. Goal Achieved – August 1st 2020. User Stories 1 – 3 provided enough research and background to set up Azure ML workspace for ML Studio (classic).

Figure: User Story 4 demonstration



2. Model generated with manual selection of features and basic modeling. Tests with 'Evaluate Model' module displayed Accuracy results of ~82%. 'Recall' value extremely poor but model acceptable for prototype.

Figure: User Story 4 Test Model Results



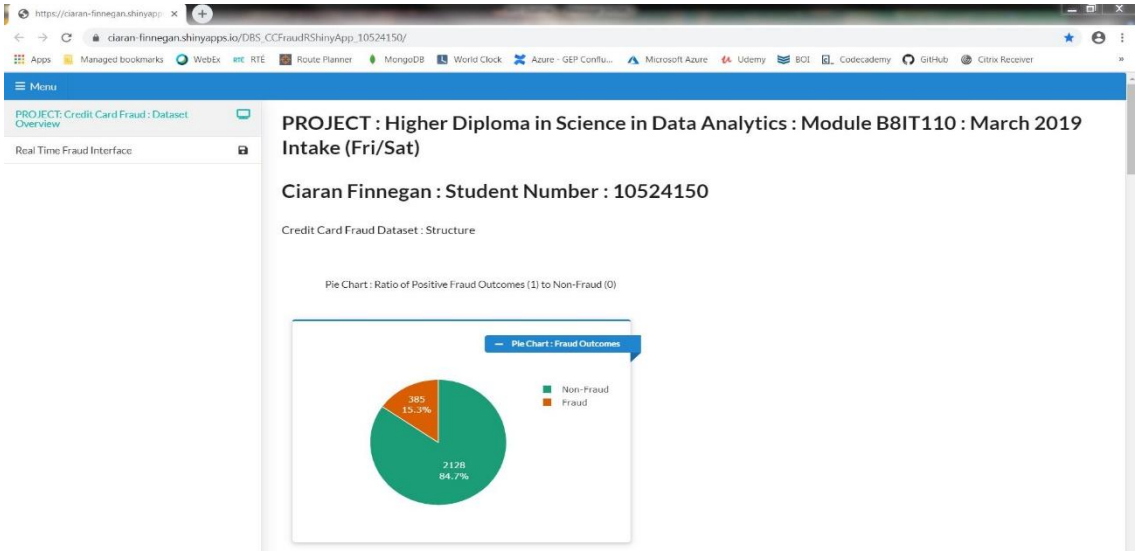
6.1.2. User Story 5: Basic Shiny App – Review and Evaluation

Goal: *Build a basic Shiny R dashboard app that displays basic EDA of my credit card dataset and has a placeholder screen for fraud detection interface.*

Assessment of robustness of code and functionality delivered:

1. Goal Achieved – August 7th, 2020.

Figure: User Story 5 demonstration



2. Quick Turnaround from User Story 1. Basic Shiny Dashboard App running without error from within RStudio environment.

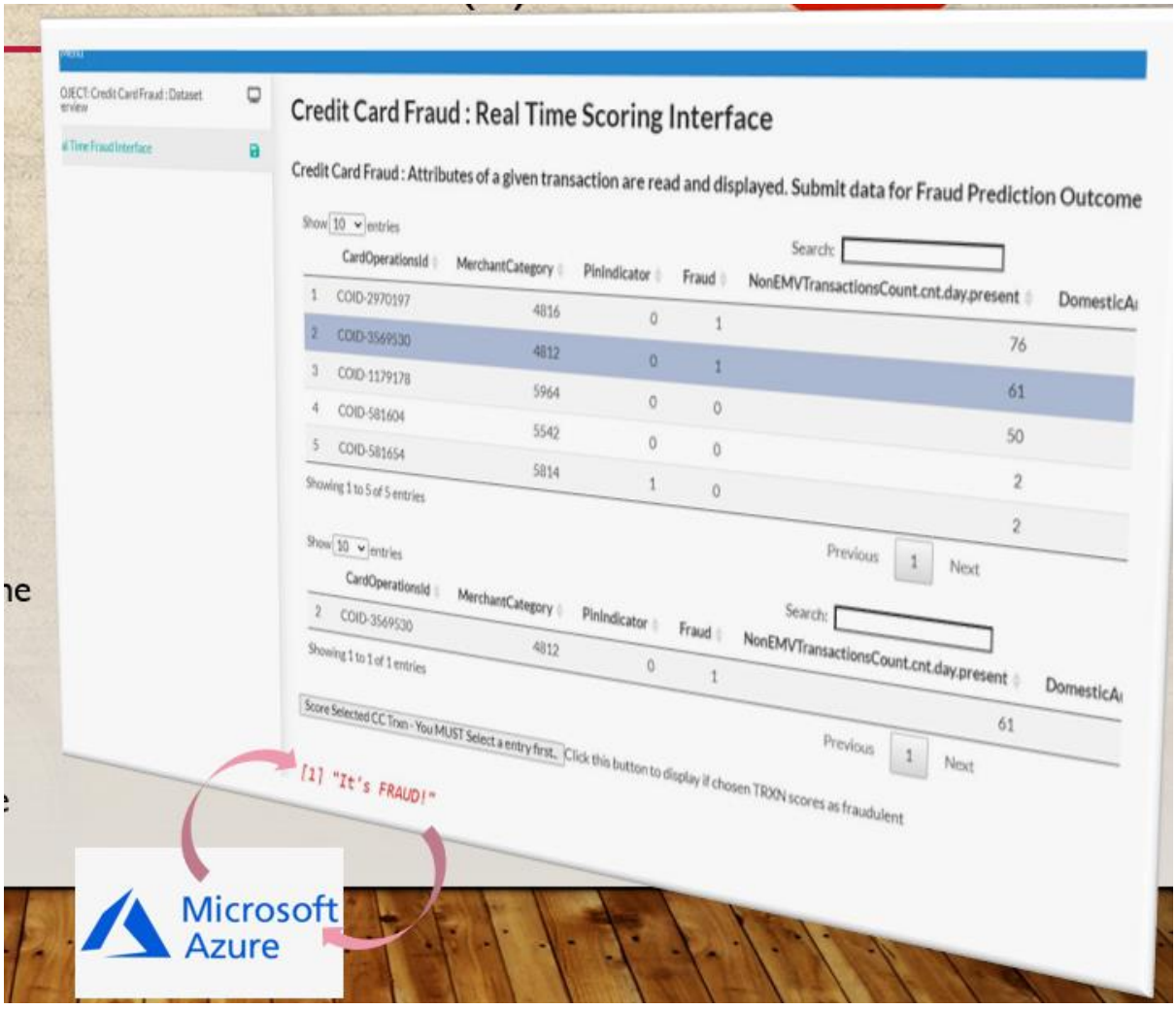
6.1.3. User Story 6: Integrated Prototype – Review and Evaluation

Goal: *Add R code to R Shiny Dashboard to invoke basic card fraud model with fixed data inputs. Host working Shiny App online.*

Assessment of robustness of code and functionality delivered:

1. Goal Achieved – August 14th, 2020. This working prototype was released online with a basic user guide as part of the Interim Report for the project.

Figure: User Story 6 demonstration



6.1.4. User Story 7: Enhanced Modelling – Review and Evaluation

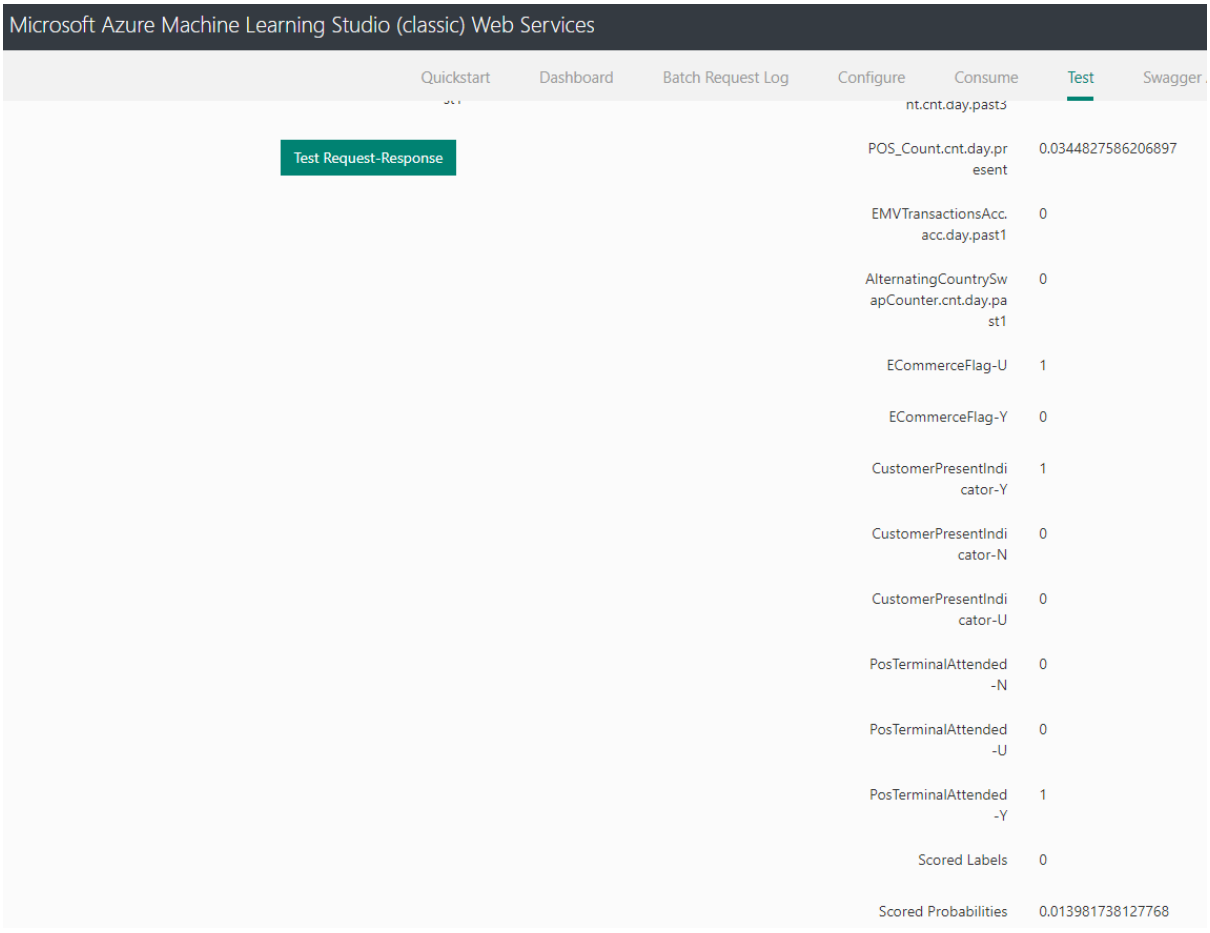
Goal: *Refine credit card model with full ML workflow processes. Enhance UI to select ad-hoc credit card transactions.*

Assessment of robustness of code and functionality delivered:

1. Goal Partially Achieved – September 5th, 2020.

2. Full end-to-end ML workflow applied to create a production ready model for credit card fraud prediction. Tested and validated in the Azure ML Studio Web Services portal.

Figure: Web Services Portal Testing of 'final' predictive model.



3. Shiny App UI only partially updated. Complexity of rebuilding model left no time to complete this section of the User Story. The UI is reading in new fixed files but there is no option to select a transaction file at random by the user. Carried over to User Story 8.

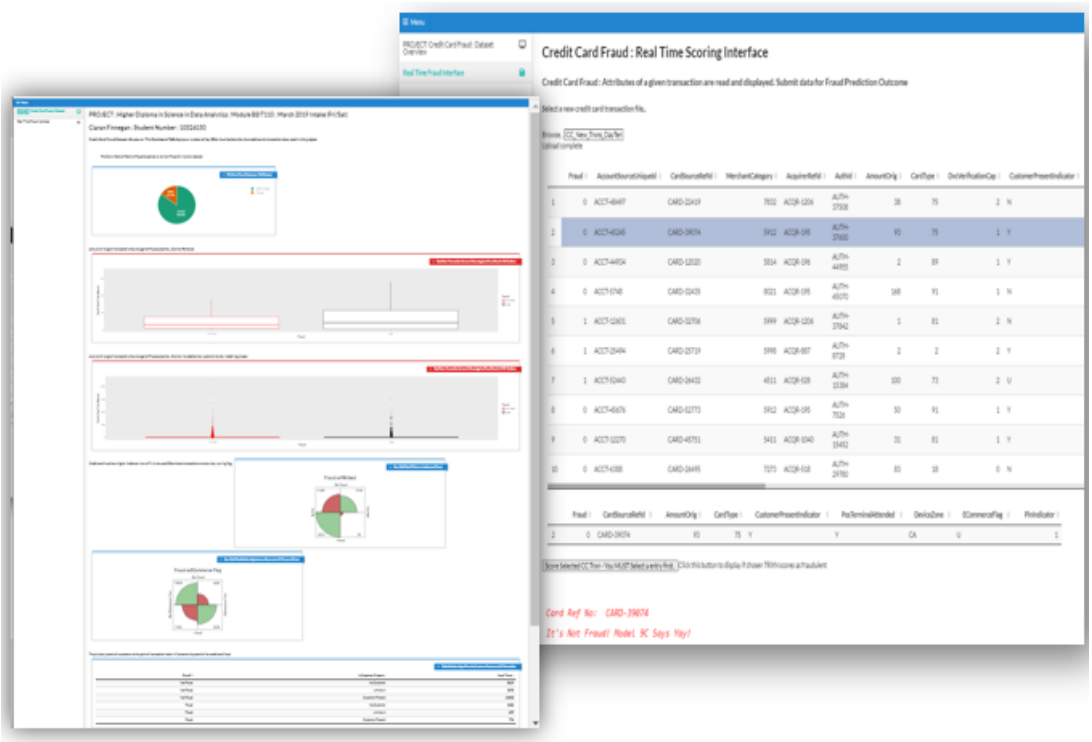
6.1.5. User Story 8: Enhanced UI – Review and Evaluation

Goal: ***(Revised)*** *Redeploy new credit card fraud model in Azure. Update code in Shiny R Dashboard to:*

- *Invoke new API*
- *Allow for ad-hoc selection of 'new' transactions to submit to predictive fraud model*
- *Display improved data visualisation graphs on UI based on credit card dataset*

Assessment of robustness of code and functionality delivered:

1. Goal Achieved – September 17th, 2020.
2. The user can select from multiple files and submit any given transaction for fraud assessment.
3. The Dashboard tab containing data visualizations of the original credit card dataset has been enhanced with additional graphs.

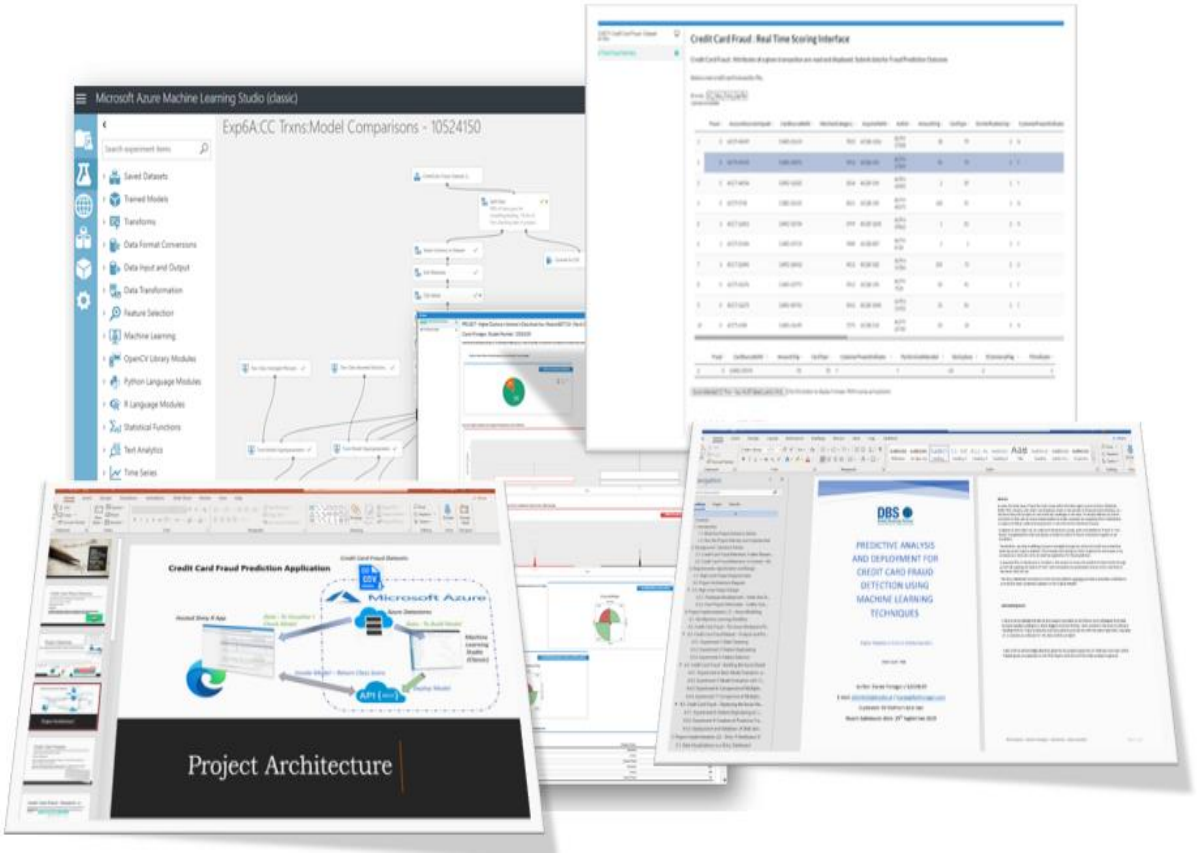


6.1.6. User Story 9: Presentation Preparation – Review and Evaluation

Goal: *Refine UI in preparation for Final project demonstration.*

Assessment of robustness of code and functionality delivered:

1. Goal Achieved – September 25th, 2020.
2. Code complete, including all code refactoring and commenting.
3. All tests completed and passed.
4. Documentation completed and proofread.
5. Project presentation completed.
6. Project submitted.



6.2. Final Project Assessment: A Critical Evaluation

The project is intended to demonstrate an end-to-end solution for credit card fraud detection, using established and comprehensive Machine Learning techniques.

Taking a checkpoint at the end of the project, I feel I have been largely successful in the goals I set out for myself. However, not all aspects of the implementation have gone as planned, and there are some inherent limitations with the approach I took.

On balance, these are my primary evaluations and observations on the project:

1. In section 2.2 of the Interim Report, I describe how a 'real-world' fraud detection system would almost certainly process credit card transactions in a large-scale batch mode. Those transaction marked as possible fraud would then be sent, usually though an internal company workflow process, into some kind of 'Case Management' system for a Fraud Investigator to review. Despite this, I feel the experience of building a real time one-by-one fraud detection interface has been an excellent learning experience for me, and a great academic challenge.

2. The Microsoft Azure Machine Learning Studio (classic) was interesting tool to learn to use, and to build and deploy the predictive mode for this project. For the most part I used the visual designer aspect of the tool. There were elements of embedded R and Python code it the experiments for Feature Engineering, but these were for relatively atomic tasks. This was a challenging project to implement but my next iteration of this project would raise the academic stakes by writing much more code in Python, within frameworks such as Jupyter Notebooks.

3. The Shiny R dashboard is reasonably aesthetically pleasing, thanks largely to the Semantic libraries used within the Shiny R development. However, there is room for improvement. The visual graphs on the first tab are only static representations of the dataset and do not take advantage of any of the interactive possibilities in Shiny. The predictive fraud interface could also use a little 'polish' when new credit card transactions are chosen but I found this difficult to implement.

6.3. Project Plan 2020: Final Status – 25th September 2020

(Produced using the Team Gantt online portal)

DBS Data Analytics Project -...	start	end	98%
User Story 1 - Submit Project Prop...	**22/06/20**	**26/06/20**	**100%**
User Story 2 - Pre-approval Resear...	**22/06/20**	**10/07/20**	**100%**
User Story 3 - Proposal Revisions	**02/07/20**	**10/07/20**	**100%**
User Story 4 - Initial Data Modelling	**10/07/20**	**03/08/20**	**100%**
Basic Dataset Preparation	10/07	13/07	100%
Azure ML Studio : Basic Card Fraud ...	14/07	16/07	100%
Create 'Skeleton' Interim Report	17/07	18/07	100%
Annual Leave	20/07	31/07	100%
Test/Fix and Document	01/08	03/08	100%
User Story 5 - Shiny App Prototype	**04/08/20**	**07/08/20**	**100%**
Shiny App UI - Local Prototype	04/08	05/08	100%
Test/Fix and Document	06/08	06/08	100%
Internal 'Sprint' Review of US 4 + 5	07/08	07/08	100%
User Story 6 - Integrated Prototype	**07/08/20**	**21/08/20**	**88%**
Deploy Azure Rest Endpoint	07/08	08/08	100%
Hosted Shiny UI Interface with API Ca...	10/08	10/08	100%
Demo Test Preparation	11/08	11/08	100%
Complete Interim Report	12/08	13/08	100%
Interim Submission Report	14/08	14/08	100%
Demonstration	21/08	21/08	0%
User Story 7 - Feature Enhanceme...	**19/08/20**	**05/09/20**	**100%**
Refine Azure Card Fraud Model	19/08	24/08	100%
Shiny R UI : Read and Score New Data	25/08	01/09	100%
Test/Fix and Document	02/09	04/09	100%
Internal 'Sprint' Demo of Prototype	05/09	05/09	100%
User Story 8 - Feature Enhanceme...	**07/09/20**	**17/09/20**	**100%**
Redeploy Azure Card Fraud Model	07/09	08/09	100%
Complete Full Shiny Data Visualisati...	09/09	11/09	100%
Test/Fix and Document	14/09	16/09	100%
Internal 'Sprint' Demo of Prototype	17/09	17/09	100%
Final Defect Resolution and Presen...	**18/09/20**	**02/10/20**	**89%**
Final Application Refinement	18/09	21/09	100%
Final Report - Completed Version	22/09	25/09	100%
Submit Final Report	25/09	25/09	100%
Final Presentation	02/10	02/10	0%

7. Project Location and User Guide

7.1. Credit Card Fraud Application: ShinyIO Location

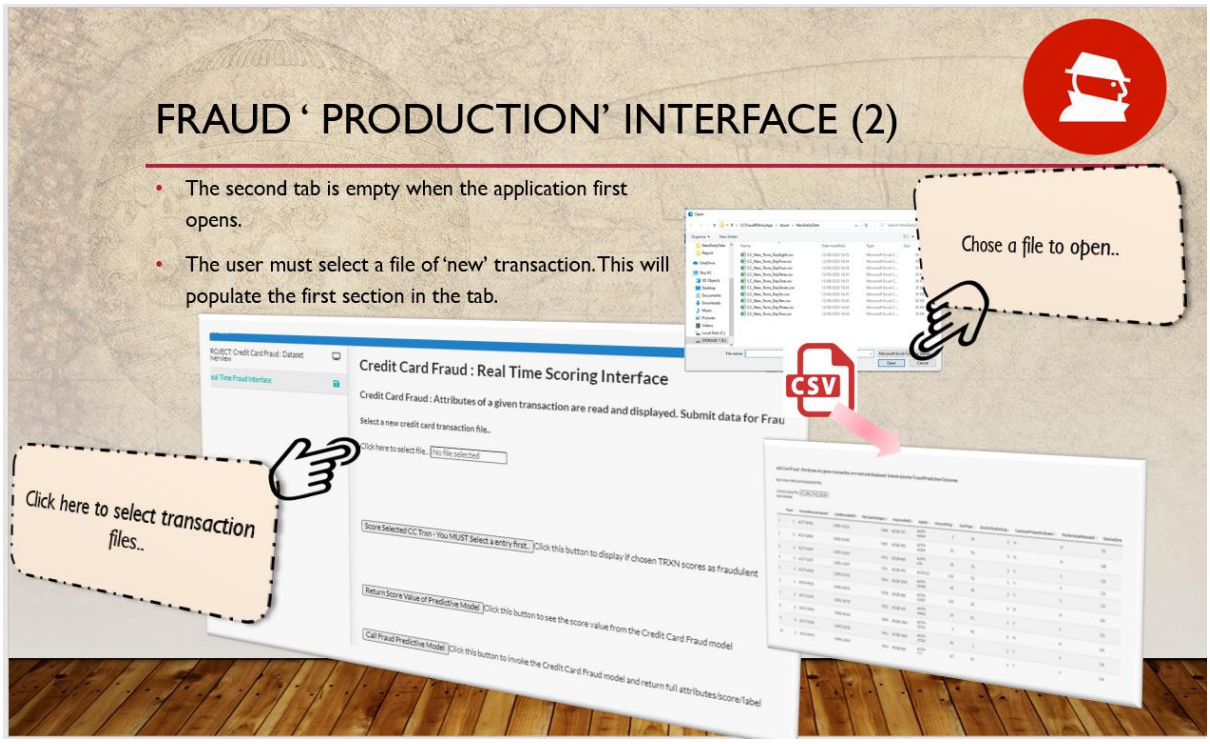
This project application is hosted on *shinyapps.io* and the UI can be accessed through this URL;

https://ciaran-finnegan.shinyapps.io/DBS_CCFraudRShinyApp_10524150/

7.2. Credit Card Fraud Application: User Guide (Final Project)

A User Guide, in Microsoft PowerPoint format, is embedded with this report, and has also been submitted separately.

Figure: Final Project User Guide



10524150_DBSFinal Project_UserGuide c

8. Project Conclusions

8.1. Where Project Goals Achieved?

Yes. Looking at the architecture diagram in Section 3.3 of the Interim Report, and reproduced in Section 3.2 of this document, I feel I built the application that I set out to create.

The Machine Learning process in the Microsoft Azure Machine Learning Studio (classic) platform built and deployed a model that performed with very satisfactory results.

The work to invoke the Web Services for the model through the Shiny R Dashboard encountered some challenges but I was pleased that the interface met the requirements I set out at the start of the project.

The overarching goal of using this project to cement the knowledge I learned throughout my Data Analytics course in DBS in 2019/2020 was most certainly achieved (IMHO).

8.2. Future Design/Deployment Considerations

Where did I deviate from the original design requirements, as documented in the Interim report?

- **Dataset size.** The prototype version of the project application, which was available in tandem with the submission of the Interim Report, worked off a subset dataset of 2.5K transactions. This was more than sufficient for the early phases of development. The final project was to use a larger dataset of 100K+ rows but issues with data formats and file type incompatibility meant that I had to settle for a final dataset size of 25K. This was still 10x times greater than the prototype, and produced an accurate/reliable model, but I would have preferred to work with more information.

- **Modelling Platform.** Although not necessarily a deviation from the original design, I had hoped to look at training and deployment options in the current Azure ML 'Services' option for the final version of the project. However, cost and complexity meant that I remained working (successfully) within the Azure ML Studio 'classic' version.

What were the learning experiences? Where will the lessons of this project lead for me?

- **R**. Building the UI reinforced my R development skills, learned during my time in DBS.

- **Data Mining**. The data manipulation and modelling techniques required for this project emphasised for me the practice and benefits of the Machine Learning workflow process.

- **Career options**. I work in a company that is making the steady evolution from rules-based applications for financial crime prevention to Machine Learning technologies. I may not be able to class myself as an expert in the field (yet) but this project, and the overall DBS course experience, has enabled me to understand the vocabulary of ML environments. My intention would be to exploit this knowledge and re-orient my career toward an ML Product development path.

What would be the suggestions for an evolution of this project with further development?

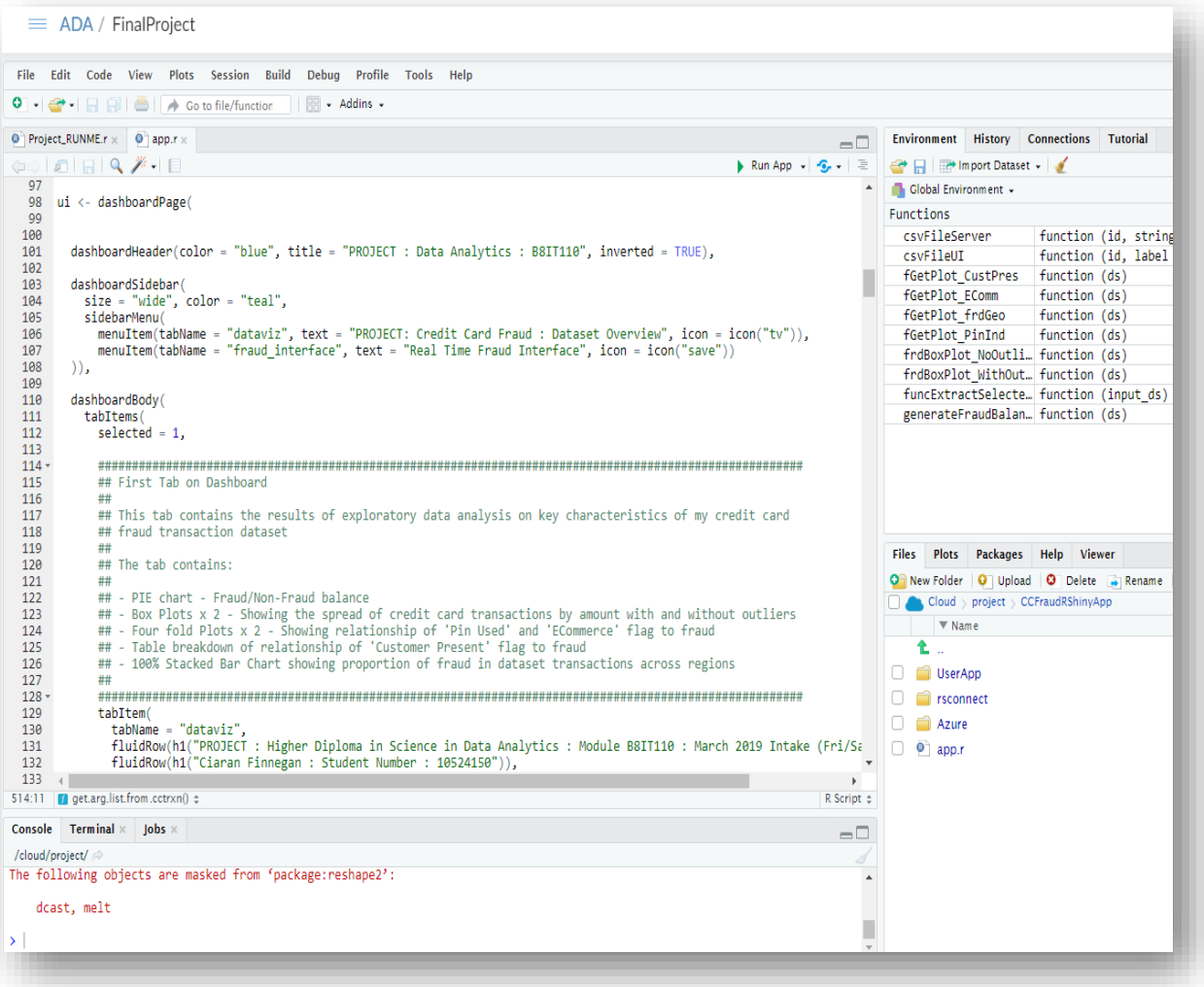
- **More data**. Although it is in a DSV file format that is not widely supported and contains 1600 columns of which the vast majority are redundant, I have access to a larger dataset than can provide 280K credit card transaction records. Working with that larger dataset on a new platform (such as AWS Sage Maker or Google Colaboratory) would be an interesting new challenge.

- **More modern ML development platform**. Even if future development chose to remain within the Microsoft Azure platform the classic studio is likely to be deprecated in the near future. Microsoft recommend users move to the newer Machine Learning *Services* platform, which integrates more seamlessly with technologies such as Jupyter Notebooks and has access to a greater range of Machine Learning algorithms.

9. Appendices

9.1. Shiny R Application Code Files

9.1.1. Diagram: The RStudio Cloud Environment



The R dashboard was developed as a Shiny application in an RStudio Cloud environment.

The R source code has been reproduced in this section of the document and is contained in this attached zip file:

rstudio-export_10524150_2020Capston

9.1.2. The Shiny UI Code – Data Visualisations – Source Code

This project interface is a Shiny R dashboard. The majority of the user interface code is contained in the 'ui' and 'server' functions in the ***app.r*** file.

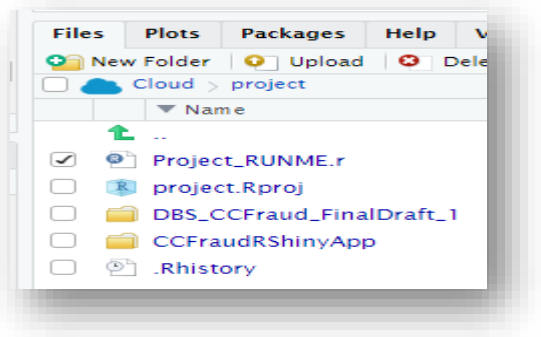
The shiny application is invoked from the ***'Project_RUNME.r'*** file in the root directory of the project.

The folder containing the *app.r* file is called *CCFraudRShinyApp*. Other functions for graphs, file access, and Azure APIs are stored files in sub directories under the *CCFraudRShinyApp* folder.

The R code for most of the graphs that appear on the first tab are contained in the *EDA_PlotDisplays.R* file in the */UserApp* sub-directory.

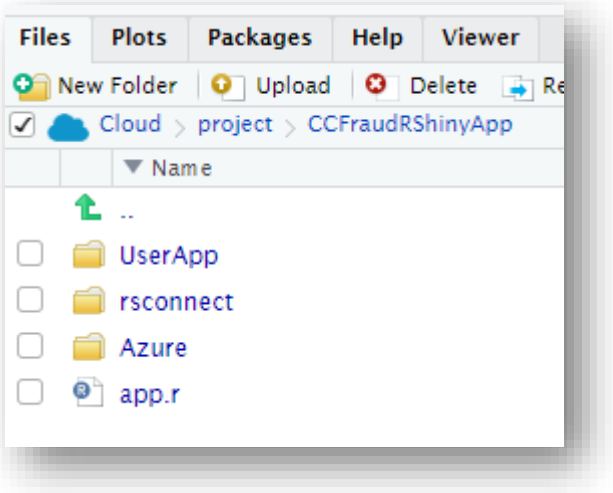
The source code for these aspects of the project are re-produced here:

1 – Project_RUNME.r




```
## Higher Diploma in Science in Data Analytics : Project : Module
Code B8IT110 : March 2019 Intake - Friday/Saturday Class
## Student Name : Ciaran Finnegan
## Student Number : 10524150
## July 2020
## This is the source code for the Shiny Semantic Dashboard App
that has two functions;
## 1 - To display a visual representation of my Credit Card Fraud
dataset
## 2 - Invoke an API call to access a Production deployment of an
Azure hosted model for fraud prediction, training on this dataset
## Set Working Directory Accordingly
# This install.packages line is only included to assist in the
first run. If these packages are already installed then it is not
required.
# install.packages(c("dplyr", "DT", "ggcorrplot", "ggplot2",
"plotly", "reshape2", "semantic.dashboard", "shinythemes"))
# If there are issues runing the Shiny App, navidate to the
\CCFraudRShinyApp folder and install packages from there.
# This command will launch the 'app.r' file in the
CCFraudRShinyApp sub folder.
runApp("CCFraudRShinyApp")
```

2 – app.r




```
## Higher Diploma in Science in Data Analytics : Project : Module
Code B8IT110
## Student Name : Ciaran Finnegan
## Student Number : 10524150
## August / September 2020
## Application Part One - R Shiny Application performing
descriptive analytics on Credit Card Fraud dataset
## The visualizations are presented through a Shiny UI dashboard
## This Shiny R application answer uses many standard R
libraries, including an open source R Shiny package for
## a Semantic Dashboard. Details of this open source package can
be found here :...
## https://appsilon.com/semantic-dashboard-new-open-source-r-
shiny-package/
#########################
library(shiny)
library(shinydashboard)
library(shinythemes)
library(semantic.dashboard)
library(reshape2)
library(ggplot2)
library(reshape2)
library(scales)
library(plotly)
library(ggcorrplot)
library(dplyr)
library(DT)
## Final Project Libraries
library(modules)
library(curl)
library(httr)
library(rjson)
library(stats)
library(readr)
## DT Formatting
library(magrittr)
library(data.table)
#########################
```

```
#####################################################################
###################################
#devtools::install_github("RevolutionAnalytics/AzureML") - Only
required during environment set up
library("AzureML")
# R Code for project split into modular files. Set up calls to
functions in R files in /Azure sub folder
api_call <- modules::use("Azure")
ui_data <- modules::use("UserApp")
source("UserApp/UI_FileSelection.R")
source("UserApp/EDA_PlotDisplays.R")
#####################################################################
###################################
#####################################################################
###################################
## - Load Azure hosted datasets for Dashboard graphs
#####################################################################
###################################
# Read Azure Hosted Credit Card Fraud Dataset for Final
Production version of the project
ds_cctxns =
api_call$AZURE_CreditCardFraud_Download$download25KProductionCCTx
ns()
# Read Azure Hosted Credit Card Fraud Dataset for Geographical
Analysis of Fraud in dataset
frdCntReport =
api_call$AZURE_CreditCardFraud_Download$downloadDevCntryCdCCTxns(
)
#####################################################################
###################################
#####################################################################
###################################
## Label fields in dataset to aid presentation of 2-dimensional
matrix graphs on UI
#####################################################################
###################################
ds_cctxns$Fraud <- as.factor(ds_cctxns$Fraud)
levels(ds_cctxns$Fraud) <- c("Not Fraud", "Fraud")   # Fraud
ds_cctxns$PinIndicator <- as.factor(ds_cctxns$PinIndicator)
levels(ds_cctxns$PinIndicator) <- c("No Pin", "With Pin")  # Pin
Indicator
ds_cctxns$ECommerceFlag <- as.factor(ds_cctxns$ECommerceFlag)
levels(ds_cctxns$ECommerceFlag) <- c("Not ECommerce Trxn",
"ECommerce Trxn")  # ECommerce Flag
ds_cctxns$CustomerPresentIndicator <-
as.factor(ds_cctxns$CustomerPresentIndicator)
levels(ds_cctxns$CustomerPresentIndicator) <- c("No Customer",
"Unknown", "Customer Present")  # Customer Present Indicator
#####################################################################
###################################
## Limit the view of columns in the single selected cc trxn on
screen
## List the columns NOT to show on the selected cc transaction -
this is a parameter later used to
## control a DataTable display on screen
columns2hide <-
ui_data$DATA_FileSelection$SelectColumsToHide(ds_cctxns)
```

```
#####################################################################
###################################
# Shiny UI Function
ui <- dashboardPage(


  dashboardHeader(color = "blue", title = "PROJECT : Data
Analytics : B8IT110", inverted = TRUE),

  dashboardSidebar(
    size = "wide", color = "teal",
    sidebarMenu(
      menuItem(tabName = "dataviz", text = "PROJECT: Credit Card
Fraud : Dataset Overview", icon = icon("tv")),
      menuItem(tabName = "fraud_interface", text = "Real Time
Fraud Interface", icon = icon("save"))
  )),

  dashboardBody(
    tabItems(
      selected = 1,


#####################################################################
#######################################
      ## First Tab on Dashboard
      ##
      ## This tab contains the results of exploratory data
analysis on key characteristics of my credit card
      ## fraud transaction dataset
      ##
      ## The tab contains:
      ##
      ## - PIE chart - Fraud/Non-Fraud balance
      ## - Box Plots x 2 - Showing the spread of credit card
transactions by amount with and without outliers
      ## - Four fold Plots x 2 - Showing relationship of 'Pin
Used' and 'ECommerce' flag to fraud
      ## - Table breakdown of relationship of 'Customer Present'
flag to fraud
      ## - 100% Stacked Bar Chart showing proportion of fraud in
dataset transactions across regions
      ##

#####################################################################
#######################################
      tabItem(
        tabName = "dataviz",
        fluidRow(h1("PROJECT : Higher Diploma in Science in Data
Analytics : Module B8IT110 : March 2019 Intake (Fri/Sat)")),
        fluidRow(h1("Ciaran Finnegan : Student Number :
10524150")),
        fluidRow(
          # Heading for dataset ----
          h2(htmlOutput("text.1"))
        ),
        fluidRow(
          # Heading for pie chart ----
```

```
          htmlOutput("text.2")
        ),
        fluidRow(
          # Output : Pie chart of Fraud Outcomes
          box(
            width = 7,
            title = "Pie Chart : Fraud Outcomes - Full Dataset",
            color = "blue",
            ribbon = TRUE,
            title_side = "top right",
            column(
              width = 7,
              plotlyOutput("plot1", height = 250)
            )
          )
        ),
        br(),
        br(),
        br(),
        fluidRow(
          htmlOutput("text.3"),
          box(
            width = 16,
            title = "BoxPlots : Transaction Amount Values Against
Fraud Result : NO Outliers",
            color = "red",
            ribbon = TRUE,
            title_side = "top right",
            column(
              14,
              plotOutput("plot2", height = 350)
            )
          )
        ),
        br(),
        br(),
        br(),
        fluidRow(
          htmlOutput("text.4"),
          box(
            width = 16,
            title = "BoxPlots : Transaction Amount Values Against
Fraud Result : WITH Outliers",
            color = "red",
            ribbon = TRUE,
            title_side = "top right",
            column(
              14,
              plotOutput("plot3", height = 350)
            )
          )
        ),
        fluidRow(
          htmlOutput("text.6"),
          box(
              width = 8,
              title = "Four Fold Plot: PIN Use vs Incidence of
Fraud",
```

```
              color = "blue",
              ribbon = TRUE,
              title_side = "top right",
              column(
                8,
                plotOutput("plot4", height = 350)
              )
            ),
          box(
              width = 8,
              title = "Four Fold Plot: Relationship between
Ecommerce CC Trxnz and Fraud",
              color = "blue",
              ribbon = TRUE,
              title_side = "top right",
              column(
                8,
                plotOutput("plot5", height = 350)
              )
          ),
        ),
        br(),
        br(),
        fluidRow(
          htmlOutput("text.7"),
          box(
            width = 16,
            title = "Table: Relationship of Fraud to Customer
Presence at CC Transaction",
            color = "blue",
            ribbon = TRUE,
            title_side = "top right",
            column(
              16,
              dataTableOutput("custPresent", width = "100%",
height = "auto")
            )
          )
        ),
        br(),
        br(),
        br(),
        br(),
        fluidRow(
          htmlOutput("text.8"),
          box(
            width = 16,
            title = "100% Stacked Bar Chart: Fraud Breakdown Per
Region",
            color = "red",
            ribbon = TRUE,
            title_side = "top right",
            column(
              14,
              plotOutput("plot6", height = 350)
            )
          )
        )
```

```
      ),


########################################################################
#####################################
      ## Second Tab on Dashboard
      ##
      ## This tab contains the interface to new 'unseen' credit
card transaction records which can then be
      ## submitted (vai API) to the Azure hosted predictive fraud
model I created in Azure ML Studio (classic)
      ##

########################################################################
#####################################
      tabItem(
        tabName = "fraud_interface",

        fluidRow(
          h1("Credit Card Fraud : Real Time Scoring Interface")
        ),

        br(),
        fluidRow(
          h3("Credit Card Fraud : Attributes of a given
transaction are read and displayed. Submit data for Fraud
Prediction Outcome")
        ),

        br(),
        br(),
        fluidRow(
          h4("Select a new credit card transaction file..")
        ),
        fluidRow(csvFileUI("NewCC_Trxn_datafile")),  ## Set up UI
elements for file selection dialog

        ## Display table of the transactions loaded from the
chosen csv file
        fluidRow(
          column(16,
                 DT::dataTableOutput("credit_card_file_txns")
          )
        ),

        ## Display details of the specific transaction chosen
from those displayed from the file
        br(),
        fluidRow(
          column(12,DT::dataTableOutput("selected_cc_trxn"))
        ),

        ## Action buttons to invoke APIs and return information
from the predicitive fraud model
        ## hosted in Azure
        br(),
        br(),
```

```
        br(),
        br(),
        fluidRow(
          actionButton("apiTxnRow"," Score Selected CC Trxn - You
MUST Select a entry first.. "),
          p(" Click this button to display if chosen TRXN scores
as fraudulent")
        ),

        ## Display on screen if the transaction is predicted to
be fraudulent or not
        fluidRow(
          verbatimTextOutput("cctxn_id"),
          tags$head(tags$style("#cctxn_id{color: red;
                               font-size: 20px;
                               font-style: italic;
                               }"
          )
          )
        ),

        br(),
        br(),

        br(),
        br(),

        ## Give option to display the actual score returned by
the Azure hosted
        ## predictive model
        fluidRow(
          actionButton("apiScoreBtn","Return Score Value of
Predictive Model"),
          p(" Click this button to see the score value from the
Credit Card Fraud model")
        ),

        fluidRow(
          verbatimTextOutput("score_result"),
          tags$head(tags$style("#score_result{color: blue;
                               font-size: 15px;
                               font-style: bold;
                               }"
          )
          )
        ),

        br(),
        br(),

        ## Give option to display the full contents the response
returned by the Azure hosted
        ## predictive model
        fluidRow(
          actionButton("apiModelRtnBtn","Call Fraud Predictive
Model"),
          p(" Click this button to invoke the Credit Card Fraud
model and return full attributes/score/label")
```

```
        ),

        fluidRow(
          verbatimTextOutput("model_result")
        )
      )

    )
  )

)
# Shiny Server Function
server <- function(input, output, session) {

  ## Call function to activate file selection dialog box on 2nd
Tab
  ## This allows the user to select a given file with a set of
  ## 'new' transactions
  fCCTrxn_datafromfile <- csvFileServer("NewCC_Trxn_datafile",
                                        stringsAsFactors = FALSE)


  ## Set up dataselect routines for user interactions on 2nd tab
  v <- reactiveValues()
  v$s <- NULL




##################################################################
#########################
  ## ---    1st Tab Functions

##################################################################
#########################

  # Generate Pie Chart to show proportion of Fraud and non-Fraud
transactions in the dataset
  output$plot1 <- renderPlotly({


    frdPlot <- generateFraudBalanceChart(ds_cctxns)
    frdPlot

  })


  # Generate Credit Card Trxn Amount BoxPlot against Fraud
Outcome and remove outliers  ##Fraud' is output variable
  ## The two sets of Box Plot graphs are shown to indicate how
outlier values in the transaction
  ## amounts could skew the modeling process
  output$plot2 <- renderPlot({

    fBxPlt1 <- frdBoxPlot_NoOutliers(ds_cctxns)
    fBxPlt1
  })
```

```
  # Generate Credit Card Trxn Amount BoxPlot against Fraud
Outcome and include outliers  ##Fraud' is output variable
  ## The two sets of Box Plot graphs are shown to indicate how
outlier values in the transaction
  ## amounts could skew the modeling process
  output$plot3 <- renderPlot({

    fBxPlt2 <- frdBoxPlot_WithOutliers(ds_cctxns)
    fBxPlt2

  })



  # Generate Comparison Matrix for Fraud occurrences vs number of
times a PIN was used in the transaction
  output$plot4 <- renderPlot({

    ## Call function and return plot to UI section
    pinPlot <- fGetPlot_PinInd(ds_cctxns)


  })

  # Generate Comparison Matrix for Fraud occurrences vs number of
times the transaction was flagged as an
  # ECommerce transaction
  output$plot5 <- renderPlot({

    ## Call function and return plot to UI section
    eCommPlot <- fGetPlot_EComm(ds_cctxns)


  })

  ## The table will be sorted on Fraud/non-Fraud that shows a
breakdown of fraud based
  ## on whether the customer was physically present at the credit
card transaction
  output$custPresent <- renderDataTable({

    # Call function the get datatable and return plot to UI
section
    # after formating the datatable output
    datatable(fGetPlot_CustPres(ds_cctxns),
              options = list(dom = 't',
                             order = list(c(0 , 'asc'))),
              rownames = FALSE,
              colnames = c('Fraud?','Is Customer Present','No
of Trxnz'),
              filter = "none")

  })

  # Generate 100% Stacked Bar Chart with Proportion of Fraud Per
Region
  output$plot6 <- renderPlot({
```

```
    ## Call function and return plot to UI section
    frdGeoBars <- fGetPlot_frdGeo(frdCntReport)
    frdGeoBars

  })



##################################################################
#########################
  ## ---    2nd Tab Functions

##################################################################
#########################
  ## Populate user interface with the transaction read
  ## from a chosen csv file
  output$credit_card_file_txns <- DT::renderDataTable({
    datatable(fCCTrxn_datafromfile(),
              selection = "single",
              extensions = 'FixedColumns',
              options = list(
                autoWidth = TRUE,
                dom = 't',
                scrollX = TRUE,
                fixedColumns = TRUE
              )
    )
  })
  ## When a transaction in the table list is clicked update
  ## the single row 'selected transaction' table
  observe({
    if(!is.null(input$credit_card_file_txns_rows_selected)){

      v$s <- input$credit_card_file_txns_rows_selected

    }
  })

  # Update the single row 'selected transaction' table
  output$selected_cc_trxn <- DT::renderDataTable({
    datatable({
                dataTableProxy(outputId = 'selected_cc_trxn') %>%
                  hideCols(hide = columns2hide)
                  fCCTrxn_datafromfile()[v$s,]
              },
              options = list(dom = 't')
             )
  })



  ## Function to parse CC Trxn Row into parameters for API call
  get.arg.list.from.cctrxn <- function(){

    # Prepare parameter list of given credit card transaction
    chosen_cc_trxns <- fCCTrxn_datafromfile()[v$s,]
    len_list <- length(chosen_cc_trxns)
```

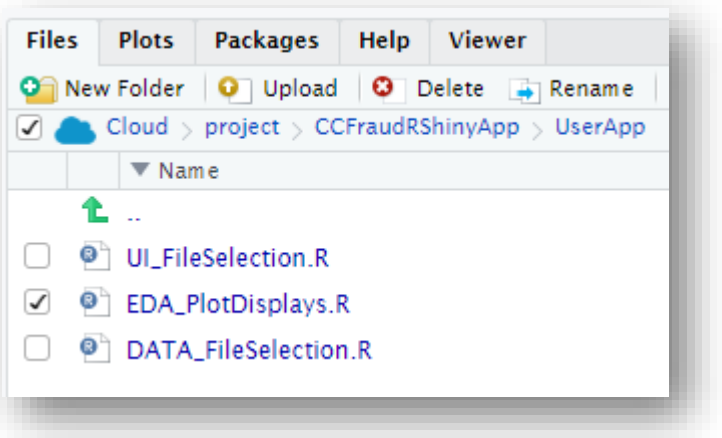
```
    arg.list <- list()

    # Start reading cc trxn record on first entry in row
    i <- 1
    while(i<(len_list+1)){  # This Reflects the number of
attribute
      # entries in the cc_trxn record

      arg.list <- append(arg.list, chosen_cc_trxns[1,i])
      i <- i + 1

    }

    return(arg.list)

  }



  ## Display a description for the user if the selected
transaction
  ## if possibly fraudulent or not
  idText <- eventReactive(input$apiTxnRow, {
    # Call function to prepare list of attributes for the API
call
    # to the predictive Fraud model
    api.arg.list.score <- get.arg.list.from.cctrxn()

    # Indicate that only the model score is required
    api.arg.list.score <- append(api.arg.list.score,
"Score_Message")

    # Pass Parameters for API Call - Full Production Model

do.call(api_call$AZURE_9C_CCFraud_APICall$Print9CFraudModelResult
,
            api.arg.list.score)


  })

  # Display CC TXN Score
  output$cctxn_id <- renderPrint({
    idText()
  })



  ## Display just the score from the Predictive Model
  scoreText <- eventReactive(input$apiScoreBtn, {

    # Call function to prepare list of attributes for the API
call to the predictive Fraud model
    api.arg.list <- get.arg.list.from.cctrxn()

    # Indicate that the full model output is required
    api.arg.list <- append(api.arg.list, "Score_Only")
```

```
    # Pass Parameters for API Call - Full Production Model

do.call(api_call$AZURE_9C_CCFraud_APICall$Print9CFraudModelResult
, api.arg.list)

  })



  # Call Fraud Model API and display result
  output$score_result <- renderPrint({
    scoreText()
  })



  ## Display Full Output of Score Predictive Model
  modelText <- eventReactive(input$apiModelRtnBtn, {

    # Call function to prepare list of attributes for the API
call to the predictive Fraud model
    api.arg.list <- get.arg.list.from.cctrxn()

    # Indicate that the full model output is required
    api.arg.list <- append(api.arg.list, "Full_Output")

    # Pass Parameters for API Call - Full Production Model

do.call(api_call$AZURE_9C_CCFraud_APICall$Print9CFraudModelResult
, api.arg.list)

  })



  # Call Fraud Model API and display result
  output$model_result <- renderPrint({
    modelText()
  })







##################################################################
##############################
  # -------  Text Ouput - Used for on screen explanation of the
visualization graphs  ---------

##################################################################
##############################

  # text output for dataset structure  description
  sHTML_for_Dataset_structure_desc=
    '<p style="color:black; font-size: 12pt">
```

```
        Credit Card Fraud Dataset : Structure : This Dashboard
TAB displays a number of key EDA visualizations
        for the credit card transaction data used in this
project</p>
      <p> </p>
      <p> </p>'
   output$text.1 <- renderUI({
    tags$div(
      HTML(sHTML_for_Dataset_structure_desc)
    )
  })

  # text output for Pie-chart
  sHTML_for_PieChart_desc_header=
    '<p style="color:black; font-size: 12pt">
        <p>    
        Pie Chart : Ratio of Positive Fraud Outcomes (1) to Non-
Fraud (0) in overall dataset</p>
        <p></p>'
  output$text.2 <- renderUI({
    tags$div(
      HTML(sHTML_for_PieChart_desc_header)
    )
  })

  # text output for Fraud Trxn Amount BotPlot against Fraud
outcome - Outliers Removed
  sHTML_for_FrdAmtBoxPlots_desc_BoxPlot_NoOut=
    '<p style="color:black; font-size: 12pt">
        <p></p><p></p><p></p><p></p>
        Analysis of range of transaction amounts against Fraud
outcomes....Outliers Removed
        <p></p>
        <p></p><p></p><p></p><p></p>
        <p></p>'
  output$text.3 <- renderUI({
    tags$div(
      HTML(sHTML_for_FrdAmtBoxPlots_desc_BoxPlot_NoOut)
    )
  })


  # Text output for Transaction Amount BoxPlot against Fraud
outcome - Outliers Included
  sHTML_for_FrdAmtBoxPlots_desc_BoxPlot_Out=
    '<p style="color:black; font-size: 12pt">
        <p></p><p></p><p></p><p></p>
        Analysis of range of transaction amounts against Fraud
outcomes....Outliers included to show potential to skew modeling
process
        <p></p>
        <p></p><p></p><p></p><p></p>
        <p></p>'
  output$text.4 <- renderUI({
    tags$div(
      HTML(sHTML_for_FrdAmtBoxPlots_desc_BoxPlot_Out)
    )
  })
```

```
  # text output for Four Plots on PIN and ECommerce Flags
  sHTML_for_FrdPinEComm_desc_BoxPlot=
    '<p style="color:black; font-size: 12pt">
        <p></p><p></p><p></p><p></p>
        Credit card fraud has a higher incidence when a PIN is
not used. ECommerce transactions are also a key
        warning flag.
        <p></p>
        <p></p><p></p><p></p><p></p>
        <p></p>'
  output$text.6 <- renderUI({
    tags$div(
      HTML(sHTML_for_FrdPinEComm_desc_BoxPlot)
    )
  })

  # text output for Table of Fraud vs Customer Present at
Transactions
  sHTML_for_FrdCustPres_desc_BoxPlot=
    '<p style="color:black; font-size: 12pt">
        <p></p><p></p><p></p><p></p>
        The physical present of a customer at the point of
transaction had an influence on the potential
        for credit card fraud
        <p></p>
        <p></p><p></p><p></p><p></p>
        <p></p>'
  output$text.7 <- renderUI({
    tags$div(
      HTML(sHTML_for_FrdCustPres_desc_BoxPlot)
    )
  })

  # text output for Fraud % BotPlot against Geographical Location
  sHTML_for_FrdGeoVars_desc_BoxPlot=
    '<p style="color:black; font-size: 12pt">
        <p></p><p></p><p></p><p></p>
        Analysis of pecentage of Fraud transaction by
geographical regions. The vast majority of transactions are from
the
        NA region but the patterns outside of the Americas are
interesting.
        <p></p>
        <p></p><p></p><p></p><p></p>
        <p></p>'
  output$text.8 <- renderUI({
    tags$div(
      HTML(sHTML_for_FrdGeoVars_desc_BoxPlot)
    )
  })


}
shinyApp(ui, server)
```

3 – EDA_PlotDisplays.R




```
## Higher Diploma in Science in Data Analytics : Project : Module
Code B8IT110
## Student Name : Ciaran Finnegan
## Student Number : 10524150
## September 2020
# Functions to generate graphs for display on screen for dataset
visualizations
# This code is being written in a separate R file to make the
overall project
# code more readable.
# The 'app.r' file for the Shiny R application is a large file
and these functions
# are being written here and invoked directly from the code in
'app.r'.
## Generate PIE Chart to show balance of volume of Fraud vs Non-
Fraud transactions in
## the project dataset
generateFraudBalanceChart = function(ds) {

  # Generate a count by Fraud Outcome in dataset   ##'Fraud' is
output variable
  values <- ds %>%
    group_by(Fraud) %>%
    summarize(count = n())

  # Set Colours
  colours <- c('rgb(27, 158, 119)', 'rgb(217, 95, 2)') #"#1B9E77"
"#D95F02"

  labels = c('Non-Fraud','Fraud')

  # Generate graph
  p1 <- plot_ly(data = values,type='pie',
          labels=labels,
          values=~count,
          text = ~count,
          marker = list(colors = colours,
```

```
                          line = list(color = '#FFFFFF', width =
1)))

  p1

}
# Generate Credit Card Trxn Amount BoxPlot against Fraud Outcome
and remove outliers  ##Fraud' is output variable
frdBoxPlot_NoOutliers = function(ds) {

  p0 = ggplot(ds, aes(x = Fraud, y = AmountOrig, col = Fraud)) +
    geom_boxplot(outlier.shape = NA) +
    ylab("Credit Card Trxn Amount") +
    scale_color_manual(values = c("red", "black")) +
    scale_fill_manual(values = c("red", "black"))

  # Rescale the Box Plot to Remove the outliers - only focus on
quartiles and whiskers
  sts <- boxplot.stats(ds$AmountOrig)$stats
  p1 = p0 + coord_cartesian(ylim = c(sts[2]/2,max(sts)*1.75))
  p1

}
# Generate Credit Card Trxn Amount BoxPlot against Fraud Outcome
and include outliers  ##Fraud' is output variable
frdBoxPlot_WithOutliers = function(ds) {

  pl1 <- ggplot(ds, aes(x = Fraud, y = AmountOrig, col = Fraud))
+
    geom_boxplot(alpha = 0.2,
                 outlier.shape=8,
                 outlier.size=3) +
    ylab("Credit Card Trxn Amount") +
    scale_color_manual(values = c("red", "black")) +
    scale_fill_manual(values = c("red", "black"))

  pl1

}
## Return a Four Fold Style 2-Dimensional Matrix Table showing
ratio of Fraud broken
## down by those transactions where a PIN was or was not used.
fGetPlot_PinInd = function(ds) {

  frd_pin <- table(ds$Fraud, ds$PinIndicator)

  fourfoldplot(frd_pin, color = c("#CC6666", "#99CC99"),
               conf.level = 0, margin = 1, main = "Fraud vs PIN
Used")

}
## Return a Four Fold Style 2-Dimensional Matrix Table showing
ratio of Fraud broken
## down by those transactions where a PIN was or was not used.
fGetPlot_EComm = function(ds) {


  frd_eComm <- table(ds$Fraud, ds$ECommerceFlag)
```

```
  fourfoldplot(frd_eComm, color = c("#CC6666", "#99CC99"),
               conf.level = 0, margin = 1, main = "Fraud vs
ECommerce Flag")


}
## Return a datatable based on Fraud and
CustomerPresentIndicator.
## The output will be sorted on Fraud/non-Fraud that shows a
breakdown of fraud based
## on whether the customer was physically present at the credit
card transaction
fGetPlot_CustPres = function(ds) {

  frd_custPres <- table(ds$Fraud, ds$CustomerPresentIndicator)
  frd_custPres

}
## Generate and return a 100% stacked bar chart that shows the
percentage of Fraud/non-Fraud
## across the different geographical regions in the Credit Card
dataset
fGetPlot_frdGeo = function(ds) {

  # Label the fraud column values to improve the graph display
  ds$Fraud <- as.factor(ds$Fraud)
  levels(ds$Fraud) <- c("Not Fraud", "Fraud")   # Fraud


  # Group the data by geographical region and set up the
percentage information
  frdPercentData <- ds  %>% group_by(DeviceCountryCode) %>%
count(Fraud) %>%
    mutate(ratio=scales::percent(n/sum(n)))


  # Generate graph
  p1 <- ggplot(ds, aes(x=DeviceCountryCode, fill =
factor(Fraud))) +
    geom_bar(color = "black", position = "fill") +
    geom_text(data = frdPercentData, aes(y=n, label=ratio),
              position = position_fill(vjust = 0.5)) +
    scale_y_continuous(labels = percent_format()) +
    labs(y="Percentage Fraud") +
    scale_x_discrete("Geographical Regions") +
    ggtitle("Credit Card Fraud Breakdown Per Geographical
Region") +
    theme(plot.title = element_text(hjust = 0.5),
          legend.position = "bottom", legend.direction =
"horizontal")

  p1

}
```

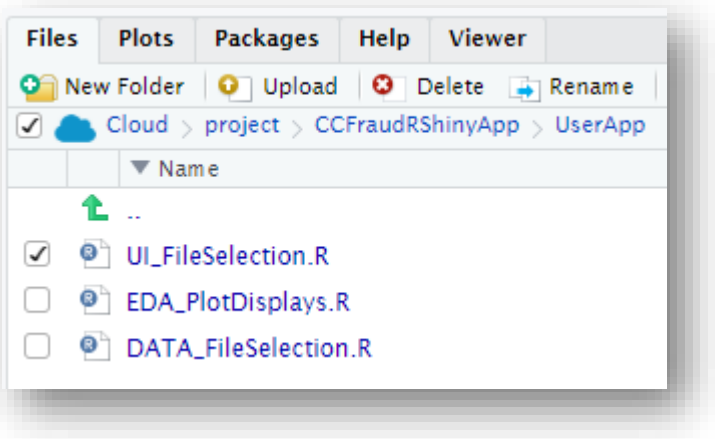
9.1.3. The R Source Code – Transaction Fraud Detection

Most of the R source code for the project that manages the user interface for fraud detection of 'new' loaded transactions in contained in the *app.r* file, described in Section 9.1.2 of these Appendices.

Two files in the ***/UserApp*** sub-directory contain code for the file dialog and the filtering of information for 'selected transactions', *UI_FileSelection.R* and *DATA_FileSelection.R* respectively.

The central function in the application is contained in the **/Azure** sub-directory in a file named ***AZURE_9C_CCFraud_APICall.r***. This contains the source code for the function to invoke an API to access the Web Service for the predictive credit card fraud model.

1 – UI_FileSelection.R

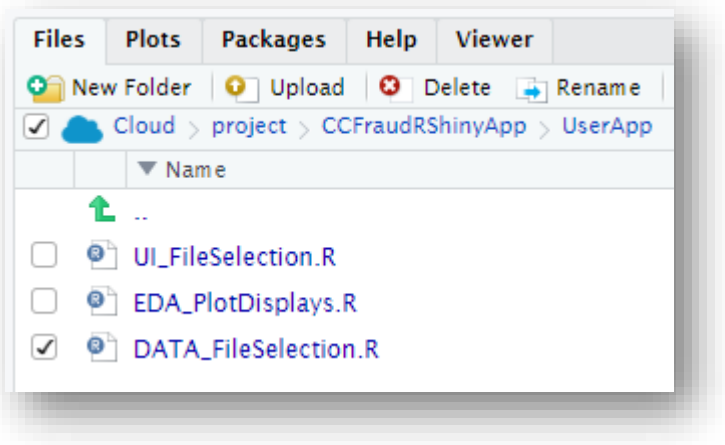



```
## Higher Diploma in Science in Data Analytics : Project : Module
Code B8IT110
## Student Name : Ciaran Finnegan
## Student Number : 10524150
## September 2020
## This file contains the Shiny code for File Select on the
dashboard.
## It is separated out from the main 'app.R' file because the
project
## has split source code into modules to avoid the main Shiny
application
## file from growing too large. (This attempts to make the code
more
## readable).
## This function is called within the main 'csvFileServer'
function
```

```
## It is a wrapper function to filter the contents of the loaded
csv file
## The function works of a list of feature names and only these
values
## are passed into a data frame for use in the main application.
## These features are the 28 features selected for use by the
trained
## predictive fraud model. Any other data in the csv file is
unnecessary
funcExtractSelectedFeaturesfromFile = function(input_ds) {
  # Set up the columns that are intended for display
  # and to be passed to the Fraud API
  myvars <- c(
        'Fraud',
        'AccountSourceUniqueId',
        'CardSourceRefId',
        'MerchantCategory',
        'AcquirerRefId',
        'AuthId',
        'AmountOrig',
        'CardType',
        'DvcVerificationCap',
        'CustomerPresentIndicator',
        'PosTerminalAttended',
        'DeviceZone',
        'DeviceCountryCode',
        'ECommerceFlag',
        'PinIndicator',
        'HighRiskPOSCnt.cnt.hour.present',
        'FuelPumpCount.cnt.day.total',
        'FuelPumpCount.cnt.day.present',
        'NotECommerceAuthAmount.acc.day.past3',
        'NonEMVTransactionsCount.cnt.day.past29',
        'POSTerminalAttendedAuthCount.cnt.day.past3',
        'DomesticAuthCount.cnt.hour1',
        'NotECommerceAuthCount.cnt.day.present',
        'EMVTransactionsCount.cnt.day.present',
        'EMVTransactionsCount.cnt.day.past3',
        'POS_Count.cnt.day.present',
        'EMVTransactionsAcc.acc.day.past1',
        'AlternatingCountrySwapCounter.cnt.day.past1'
      )
  newdata <- input_ds[myvars]
  newdata
}
# Module UI function for File Select on Dashboard tab
csvFileUI <- function(id, label = "\n") {
  # `NS(id)` returns a namespace function, which was save as `ns`
and will
  # invoke later.
  ns <- NS(id)

  tagList(
    fileInput(ns("file"), label, accept = ".csv",  width =
'400px',
              buttonLabel = "Click here to select file...")
  )
}
```

```
# Module server function for File Select on Dashboard tab
csvFileServer <- function(id, stringsAsFactors) {
  moduleServer(
    id,
    ## Below is the module function
    function(input, output, session) {
      # The selected file, if any
      userFile <- reactive({
        # If no file is selected, don't do anything
        validate(need(input$file, message = FALSE))
        input$file
      })

      # The user's data, parsed into a data frame
      # This is wrapped within another function that filters only
the required
      # credit card transaction features from the file
      dataframe <- reactive({
        funcExtractSelectedFeaturesfromFile(
                read.csv(userFile()$datapath,
                stringsAsFactors = stringsAsFactors)
        )
      })

      # We can run observers in here if we want to
      observe({
        msg <- sprintf("\n\nFile %s was uploaded",
userFile()$name)
        cat(msg, "\n")
      })

      #cc_trxn_file_ds <-
funcExtractSelectedFeaturesfromFile(dataframe)
      cc_trxn_file_ds <- dataframe

      # Return the reactive that yields the data frame
      return(cc_trxn_file_ds)  #return(dataframe)
    }
  )
}
```

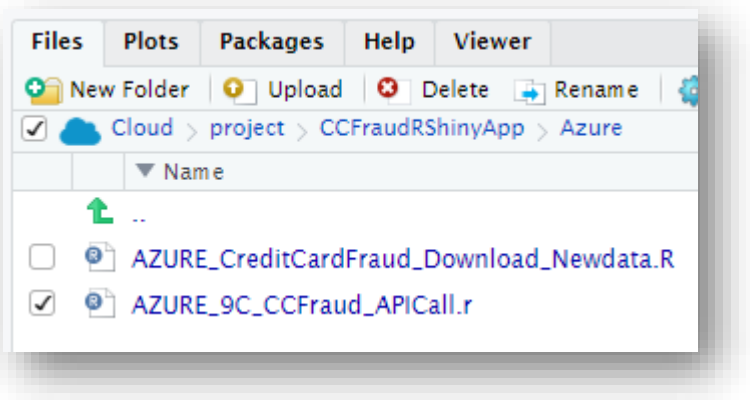
2 – DATA_FileSelection.R




```
## Higher Diploma in Science in Data Analytics : Project : Module
Code B8IT110
## Student Name : Ciaran Finnegan
## Student Number : 10524150
## September 2020
## This file contains the Shiny code to limit display values for
a 'selected
## transaction'.
## It is separated out from the main 'app.R' file because the
project
## has split source code into modules to avoid the main Shiny
application
## file from growing too large. (This attempts to make the code
more
## readable).
## This function is a further restriction on the feature list
displayed on screen
## It is used when a credit card transaction is selected from the
main list on the
## Fraud UI tab and the details of that selection are reproduced
in the single line
## 'selected transaction' table.
## The feature list is reduced so that information for the
transction fits on screen
## without the need for a scroll bar.
SelectColumsToHide = function(cc_Txns){

  # Restrict Display on single view of a selected credit card
transaction
  # Done to improve on screen display and navigation


  # Choose Columns NOT to Display
  select_cols = c(
    'AccountSourceUniqueId',
    'MerchantCategory',
    'AcquirerRefId',
    'AuthId',
    'DvcVerificationCap',
    'DeviceCountryCode',
```

```
    'HighRiskPOSCnt.cnt.hour.present',
    'FuelPumpCount.cnt.day.total',
    'FuelPumpCount.cnt.day.present',
    'NotECommerceAuthAmount.acc.day.past3',
    'NonEMVTransactionsCount.cnt.day.past29',
    'POSTerminalAttendedAuthCount.cnt.day.past3',
    'DomesticAuthCount.cnt.hour1',
    'NotECommerceAuthCount.cnt.day.present',
    'EMVTransactionsCount.cnt.day.present',
    'EMVTransactionsCount.cnt.day.past3',
    'POS_Count.cnt.day.present',
    'EMVTransactionsAcc.acc.day.past1',
    'AlternatingCountrySwapCounter.cnt.day.past1')


  # Passed back list of columns are then used as a parameter to a
datatableproxy function
  columns2hide <- match(select_cols, colnames(cc_Txns))



}
```

3 - AZURE_9C_CCFraud_APICall.r




```
## Higher Diploma in Science in Data Analytics : Project : Module
Code B8IT110
## Student Name : Ciaran Finnegan
## Student Number : 10524150
## September 2020
## This file contains the R code to invoke the API to the Azure
hosted
## predictive fraud model.
## It is separated out from the main 'app.R' file because the
project
## has split source code into modules to avoid the main Shiny
application
## file from growing too large. (This attempts to make the code
more
## readable).
## This is the main function that invokes the Web Service hosted
in Azure for the
## predictive fraud model that was built in Microsoft Azure
Machine Learning Studio (classic)
## then deployed as a REST Endpoint within Azure.
## The calling 'app.R' Shiny R code has extracted the key
features from a credit card transaction
## selected through the user interface and passes these as
parameters.
## This function invokes the API with the transaction parameters
and returns a probability score
## based on the likelihood that the credit card transaction is
fraudulent.
## The response output is parsed based on a control flag.
## One button in the UI only requires a outcome description.
## One button in the UI only requires a model score
## Another button option returns the full model output.
Print9CFraudModelResult = function(
                                   Fraud,       #  'Fraud',
                                   AccSrcUID,   #
'AccountSourceUniqueId',
                                   CrdSrcRID,   #
'CardSourceRefId',
                                   MerchCat,    #
'MerchantCategory',
                                   AcqRID,      #  'AcquirerRefId',
```

```
                                    AuthId,     #  'AuthId',
                                    Amount,     #  'AmountOrig',
                                    CardType,   #  'CardType',
                                    DvcVrfCap,  #
'DvcVerificationCap',
                                    CustPresInd,#
'CustomerPresentIndicator',
                                    PosTrmAttd, #
'PosTerminalAttended',
                                    DeviceZone, #  'DeviceZone',
                                    DeviceCntCd,#
'DeviceCountryCode',
                                    ECommerceFlag, #
'ECommerceFlag',
                                    PinIndicator,  #
'PinIndicator',
                                    HRkPOSCtHPres, #
'HighRiskPOSCnt.cnt.hour.present',
                                    FPmpCntDtTot,  #
'FuelPumpCount.cnt.day.total',
                                    FPmpCntDtPres, #
'FuelPumpCount.cnt.day.present',
                                    NtECommAtAmtP3,#
'NotECommerceAuthAmount.acc.day.past3',
                                    NEMVTrxnCntDy29, #
'NonEMVTransactionsCount.cnt.day.past29',
                                    PosTrmAttdACntDy3,
#'POSTerminalAttendedAuthCount.cnt.day.past3',
                                    DomAthCnt1,      #
'DomesticAuthCount.cnt.hour1',
                                    NtECommAtAmtPres,#
'NotECommerceAuthCount.cnt.day.present',
                                    EMVTxCntPres,    #
'EMVTransactionsCount.cnt.day.present',
                                    EMVTxCntDy3,     #
'EMVTransactionsCount.cnt.day.past3',
                                    POSCntPres,      #
'POS_Count.cnt.day.present',
                                    EMVTxCntDy1,     #
'EMVTransactionsAcc.acc.day.past1',
                                    AltCntSpCntDy1,  #
'AlternatingCountrySwapCounter.cnt.day.past1'
                                    ReqResp          # Control flag
to parse output
                                    )
{
    import(stats)
    import(curl)
    import(rjson)
    import(httr)

    requestFailed = function(response) {
      return (response$status_code >= 400)
    }

    printHttpResult = function(response, result) {
      if (requestFailed(response)) {
```

```
        print(paste("The request failed with status code:",
response$status_code, sep=" "))

        # Print the headers - they include the request ID and the
timestamp, which are useful for debugging the failure
        print(response$headers)
      }

      print("Result:")
      print(fromJSON(result))
    }

    ## The key features from the on screen credit card
transaction are formatted into a list to be passed to the Web
Service
    ## hosting the production model.
    req =  list(
      Inputs = list(
        "input1"= list(
          list(
            'AccountSourceUniqueId' = AccSrcUID,
            'CardSourceRefId' = CrdSrcRID,
            'MerchantCategory' = MerchCat,
            'AcquirerRefId' = AcqRID,
            'AuthId' = AuthId,
            'AmountOrig' = Amount,
            'CardType' = CardType,
            'DvcVerificationCap' = DvcVrfCap,
            'CustomerPresentIndicator' = CustPresInd,
            'PosTerminalAttended' = PosTrmAttd,
            'DeviceZone' = DeviceZone,
            'DeviceCountryCode' = DeviceCntCd,
            'ECommerceFlag' = ECommerceFlag,
            'PinIndicator' = PinIndicator,
            'Fraud' = "0", # Not used in modeling process
            'HighRiskPOSCnt.cnt.hour.present' = HRkPOSCtHPres,
            'FuelPumpCount.cnt.day.total' = FPmpCntDtTot,
            'FuelPumpCount.cnt.day.present' = FPmpCntDtPres,
            'NotECommerceAuthAmount.acc.day.past3' =
NtECommAtAmtP3,
            'NonEMVTransactionsCount.cnt.day.past29' =
NEMVTrxnCntDy29,
            'POSTerminalAttendedAuthCount.cnt.day.past3' =
PosTrmAttdACntDy3,
            'DomesticAuthCount.cnt.hour1' = DomAthCnt1,
            'NotECommerceAuthCount.cnt.day.present' =
NtECommAtAmtPres,
            'EMVTransactionsCount.cnt.day.present' =
EMVTxCntPres,
            'EMVTransactionsCount.cnt.day.past3' = EMVTxCntDy3,
            'POS_Count.cnt.day.present' = POSCntPres,
            'EMVTransactionsAcc.acc.day.past1' = EMVTxCntDy1,
            'AlternatingCountrySwapCounter.cnt.day.past1' =
AltCntSpCntDy1
          )
        )
      ),
      GlobalParameters = setNames(fromJSON('{}'), character(0))
```

```
    )

    body = enc2utf8(toJSON(req))

    ## Provide the unique API key for the credit card fraud web
service
    api_key =
"PnzViKK7tw+34LsnpD5B4PwPpUEgEDy5wm3AVoK2OYSRMtA+Kp7aLqgKes1x8Zru
+ml1JVJQwpUrEcjGtbYEsQ=="
    authz_hdr = paste('Bearer', api_key, sep=' ')

    ## Provide the location of the Azure Workspace in which the
model is deployed
    response=POST(url =
"https://europewest.services.azureml.net/workspaces/7375a43f74494
0748c691a78945b2588/services/979195811821437096662bcdb1fb4eeb3/exe
cute?api-version=2.0&format=swagger",
                  add_headers('Content-Type' =
"application/json", 'Authorization' = authz_hdr),
                  body=body)

    result = content(response, type="text", encoding="UTF-8")

    ## Further Parsing Response from API call to
decompose/extract the score result
    get2json<- content(response, as = "parsed")
    parse2json<- (toJSON(get2json))
    data1 <- fromJSON(parse2json)

    # Obtain binary result from Fraud Scoring Model - the value 1
= 'fraud', 0 = 'non-fraud'
    scoreResult <- data1$Results$output1[[1]]$`Scored Labels`

    # Obtain Account and Card References from Fraud Scoring Model
- just used to enrichen the display
    AccRef <- data1$Results$output1[[1]]$AccountSourceUniqueId
    cardRef <- data1$Results$output1[[1]]$CardSourceRefId

    # Obtain the Score Probability from the Model Scoring Output
and
    # format the value for display by reducing decimal points
    scoreProbability_String <- data1$Results$output1[[1]]$`Scored
Probabilities`
    scoreProbability_Numeric <-
as.numeric(scoreProbability_String)
    scoreProbability <- format(scoreProbability_Numeric,
digits=4,nsmall=5)


    # Vary the text response based on calling parameter -
different on screen buttons return separate levels of detail
    if (ReqResp == "Score_Message") {

      # Flag if the model is predicting fraud based on the
generated 'score'
      if (scoreResult == "1") {
```

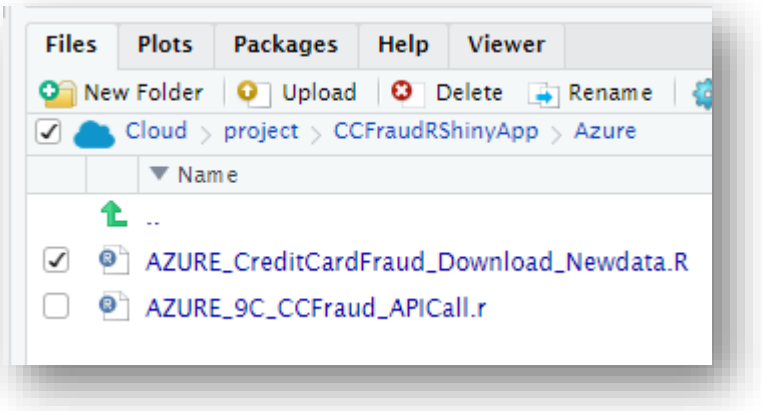
```
        testResult <- "Suspected FRAUD: Model 9C has assessed
this card transaction as fraudulent. Please Investigate."

      } else {

        testResult <- "Transaction Legitimate: Model 9C has
assessed this card transaction as non-fraudulent."

      }

      # Return Message Text to indicate if Fraud predicted or
not.
      cat("\n","Account Ref No: ", AccRef, "\t","\t", "Card Ref
No: ",cardRef,"\n","\n", testResult, "\n", "\n")
    }

    else if (ReqResp == "Score_Only"){

      # # Return Message Text to indicate if Fraud predicted or
not.
      cat("\n", "Score for Probability of Fraud is : ",
scoreProbability)

    }

    else {



      # Print the entire response message
      printHttpResult(response, result)

    }


}
```

9.1.4. The R Source Code – Loading Data for Visualisations

The data visualizations on the first tab on the dashboard use datasets hosted within the same Azure workspace that is used to train and deploy the model.

The file *AZURE_CreditCardFraud_Download_Newdata.R* contains the functions that invoke APIs to load the dataset contents from the Azure workspace.

1- AZURE_CreditCardFraud_Download_Newdata.R




```
## Higher Diploma in Science in Data Analytics : Project : Module
Code B8IT110
## Student Name : Ciaran Finnegan
## Student Number : 10524150
## August / September 2020
## This file contains the R code to invoke the API to load Azure
hosted
## credit card datasets
## It is separated out from the main 'app.R' file because the
project
## has split source code into modules to avoid the main Shiny
application
## file from growing too large. (This attempts to make the code
more
## readable).
## These routines directly read credit card transaction datasets
that are stored
## in Azure. This allows for greater ease of deployment of the
application and
## more effective automatic access to the datasets
##  --  These libraries were required during RStudio environment
set up -- ##
##devtools::install_github("RevolutionAnalytics/AzureML")
##install.packages("AzureML")
##library(AzureML)
## ----------------------------------------------------------------
--------- ##
```

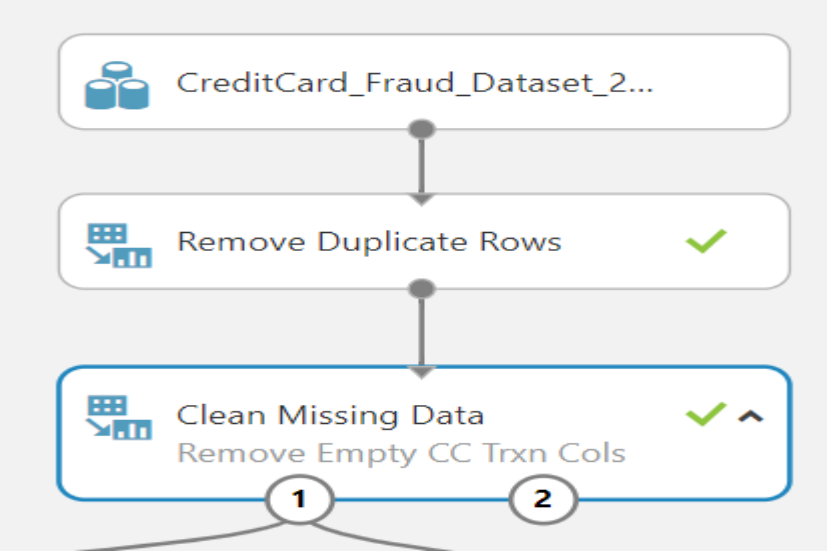
```
## This function returns the credit card transaction dataset that
was used for the
## creation of the final trained predictive model for fraudulent
credit card transactions
## This function drives all of the data visualization graphs on
the 1st tab with the
## exception of the 100% Stacked Bar chart.
download25KProductionCCTxns= function(){

  import("AzureML")

  ws <- workspace(
    id = "7375a43f744940748c691a78945b2588",
    auth =
"hItFXmJqT2qV4rYUREXnnfy5ZfleIiXBiQB8q1bA1zfF53dcJJP4CwqXvgy2Wls/
QpAhX0jEqzaLNWqjMfKDTQ==",
    api_endpoint = "https://europewest.studioapi.azureml.net"
  )
  ds <- download.datasets(
    dataset = ws,
    name =
"CreditCard_Fraud_Cleaned_Dataset_EDA_25KRows_Sept_v1_2020.csv",
    fill = TRUE
  )

  return(ds)


}
## This function loads data that has been manipulated to enhance
the 25K row dataset with an grouping
## for geographical locations
## The data is used for the 100% Stacked Bar Chart breakdown for
Fraud across differnt regions.
downloadDevCntryCdCCTxns = function() {

  import("AzureML")


  ws <- workspace(
    id = "7375a43f744940748c691a78945b2588",
    auth =
"hItFXmJqT2qV4rYUREXnnfy5ZfleIiXBiQB8q1bA1zfF53dcJJP4CwqXvgy2Wls/
QpAhX0jEqzaLNWqjMfKDTQ==",
    api_endpoint = "https://europewest.studioapi.azureml.net"
  )
  ds <- download.datasets(
    dataset = ws,
    name =
"CreditCard_Fraud_CountryCode_Cleaned_Dataset_EDA_25KRows_Sept_v1
_2020.csv"
  )


  return(ds)

}
```

9.2. Azure Machine Learning Classic Studio Experiments

9.2.1. Experiment 1: Breakdown

Breakdown of Experiment

Exp1: Step 1. Remove duplicate rows. Remove columns with missing data



Columns with missing data were seen to have a lot of empty cells. Removal was the best/most straightforward option.

The original dataset started with **380** columns. This transformation reduced the dataset to **362** columns.

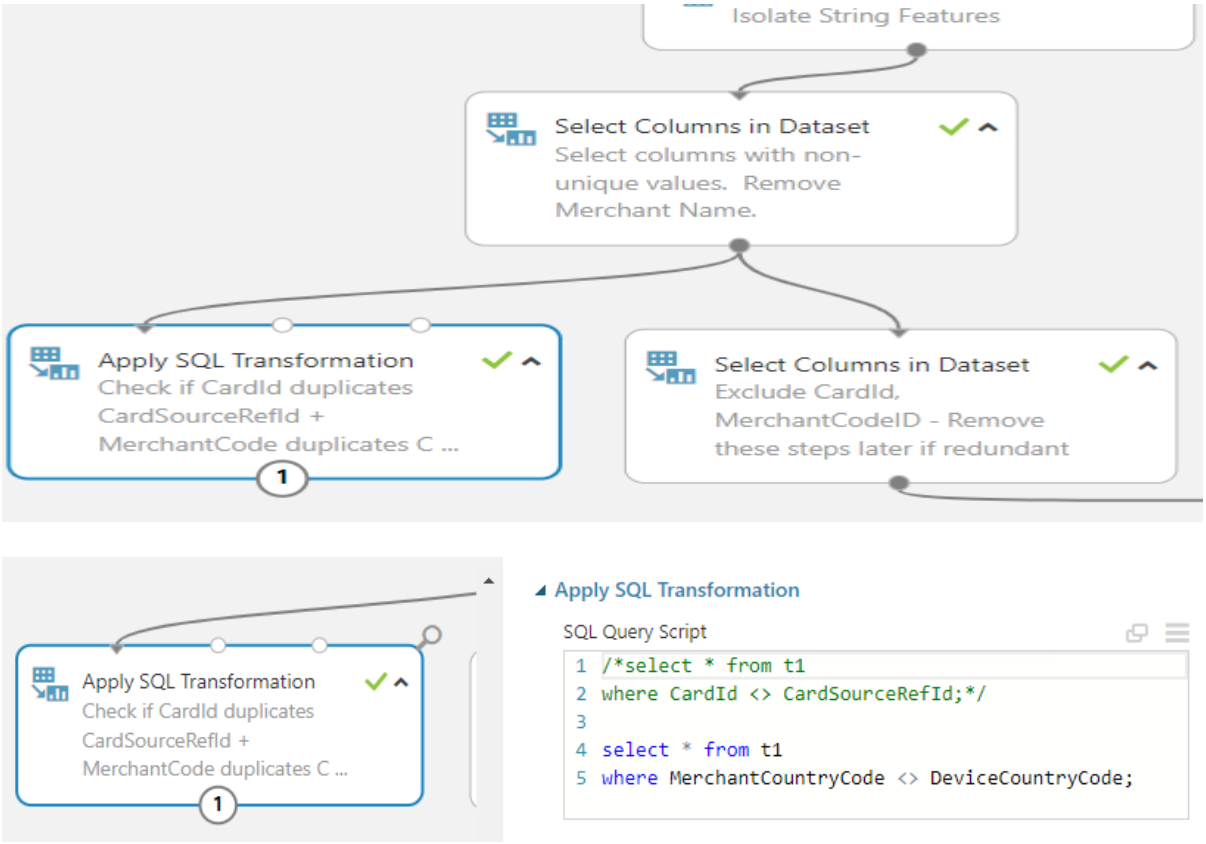
Exp1:CC Trxns: Data Cleansing - 10524150 › Clean Missing Data › Cleaned dataset

rows	columns
2513	362

Exp1: Step 2. Spilt Numeric/Non-Numeric Features

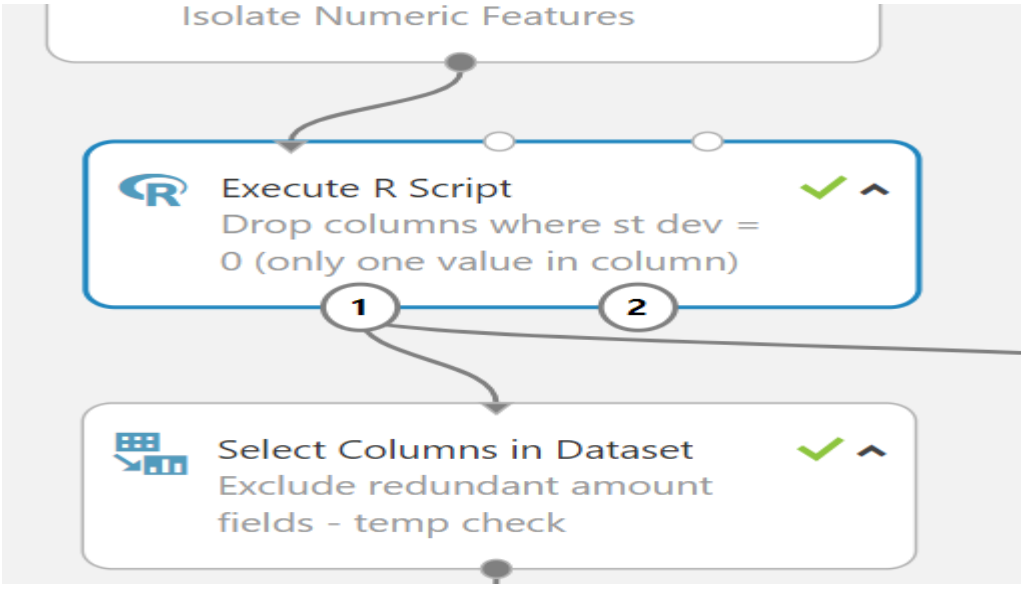
Split numeric/non-numeric features for subsequent processing.

Exp1: Step 3. Remove String Columns with Duplicate Data



Use SQL output values to check for columns with duplicate data. This requires a number of iterations and feeds into next 'Select Columns' module.

Exp1: Step 4. Remove Numeric Columns with Only One Value

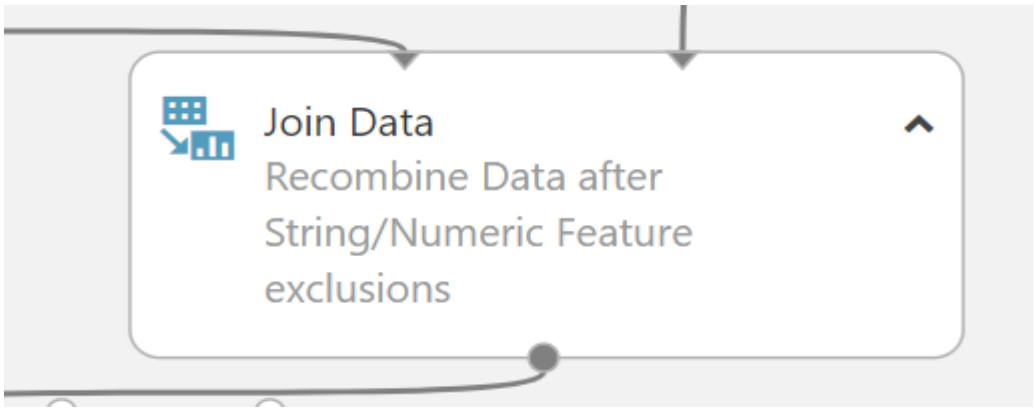


R Script

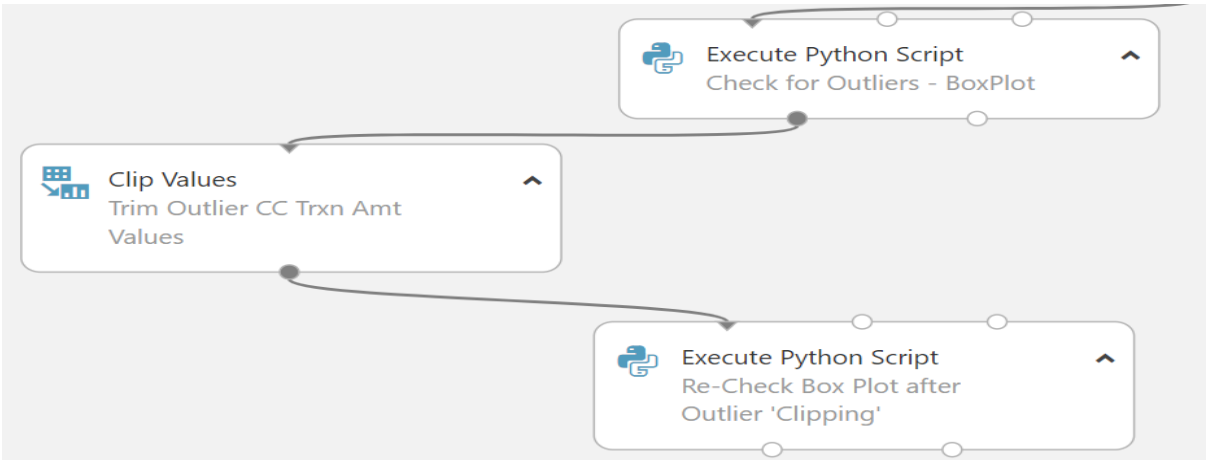
```r
## Ciaran Finnegan = 10524150
## Final Project
## R Code for Embedded Module  used in data cleansing


library(dplyr)

# Map 1- Read input dataset - all numeric values in CC trxn dataset
dataset1 <- maml.mapInputPort(1) # class: data.frame

# Prepare output dataset
data.set <- dataset1;
data.set[1] <- NULL;

# Loop through dataset - Check Std Deviation on all columns
# If it zero then the column only contains one unique values
# Remove these columns
for (i in colnames(data.set)){
  if ((sd(data.set[,i])) == 0) {
    data.set[[i]] <- NULL
  }
}


# Select data.frame to be sent to the output Dataset port
data.set <- bind_cols(dataset1[1], data.set)
maml.mapOutputPort("data.set");
```

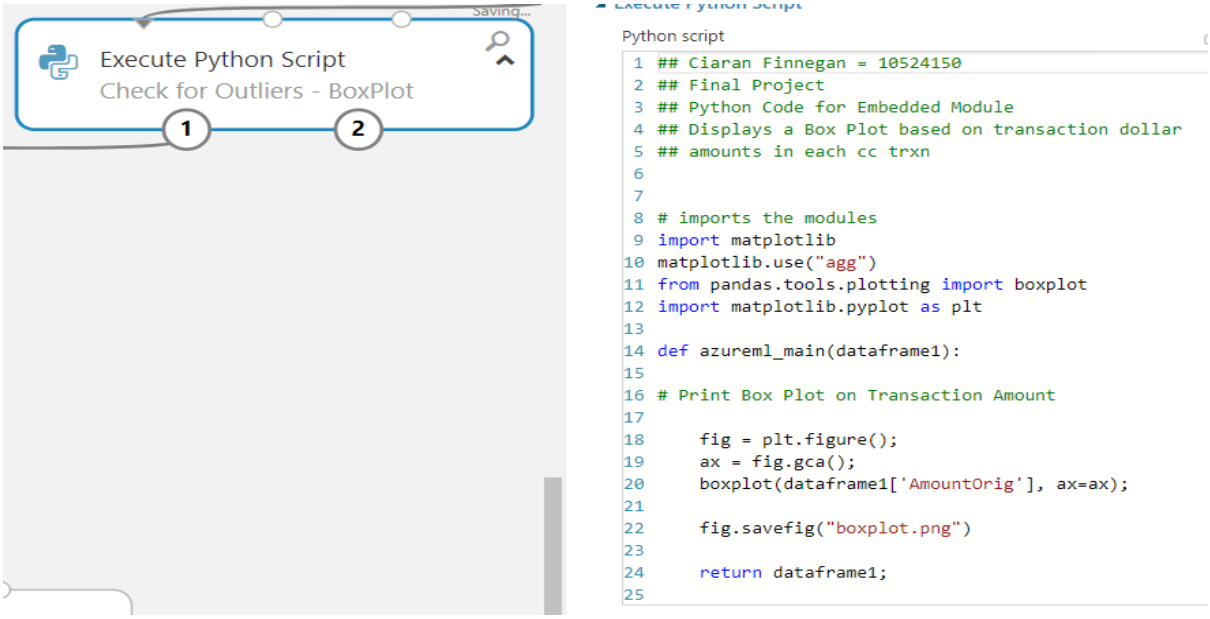
Use embedded R code module to isolate columns that only contain one value. feeds into next 'Select Columns' module.

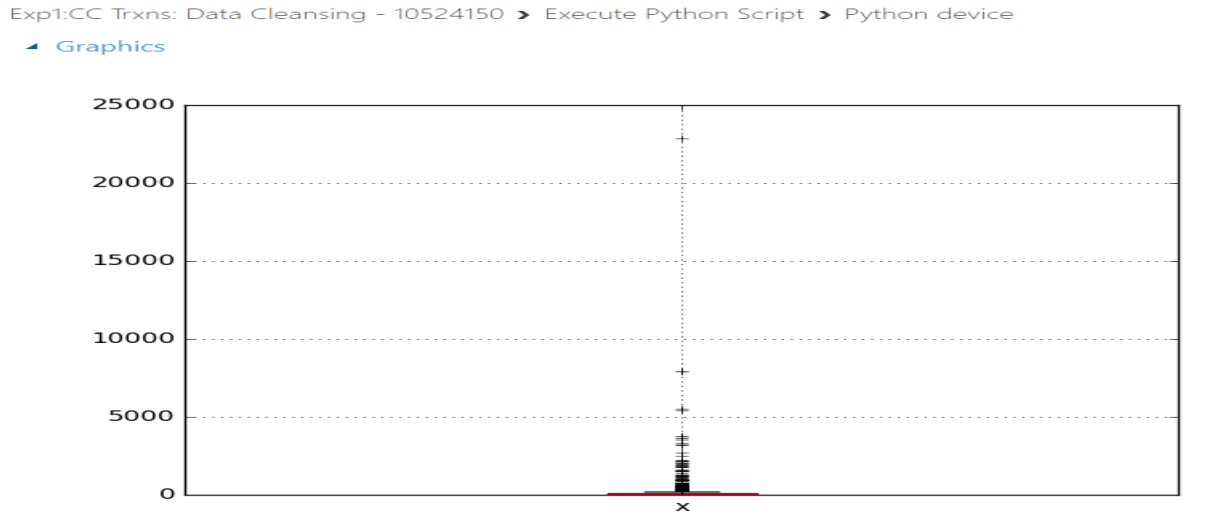
Exp1: Step 5. Recombine Non-Numeric/Numeric Features



Exp1: Step 6. Check for and Remove Outliers

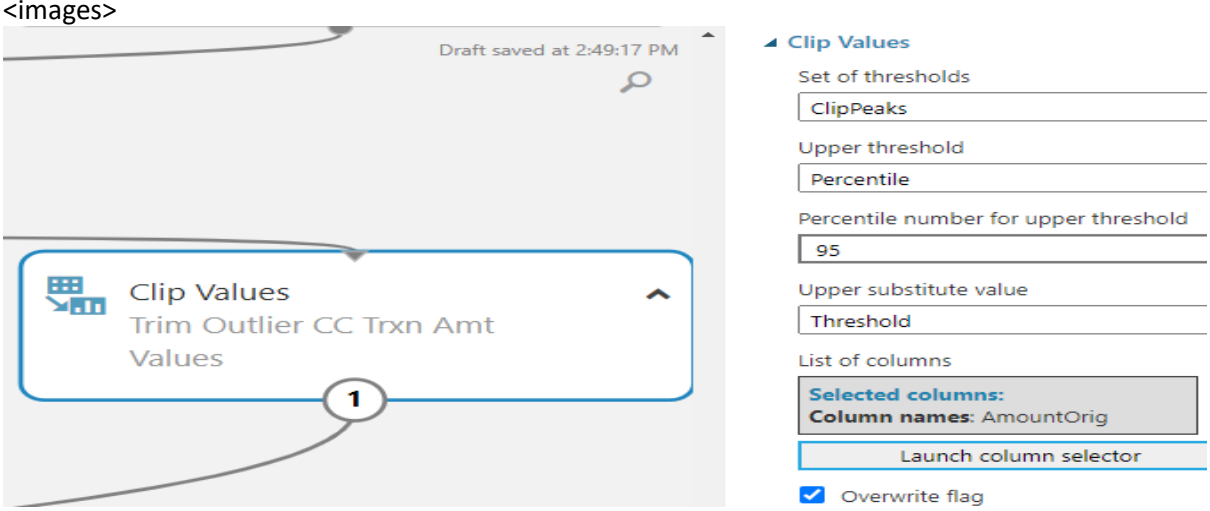




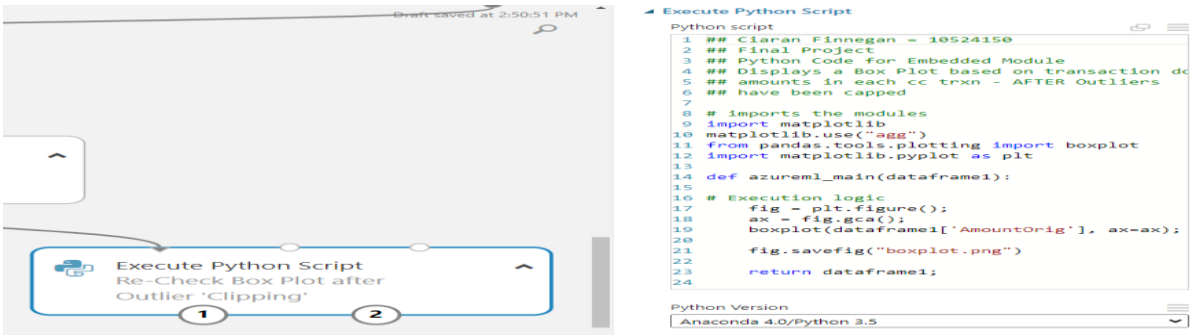


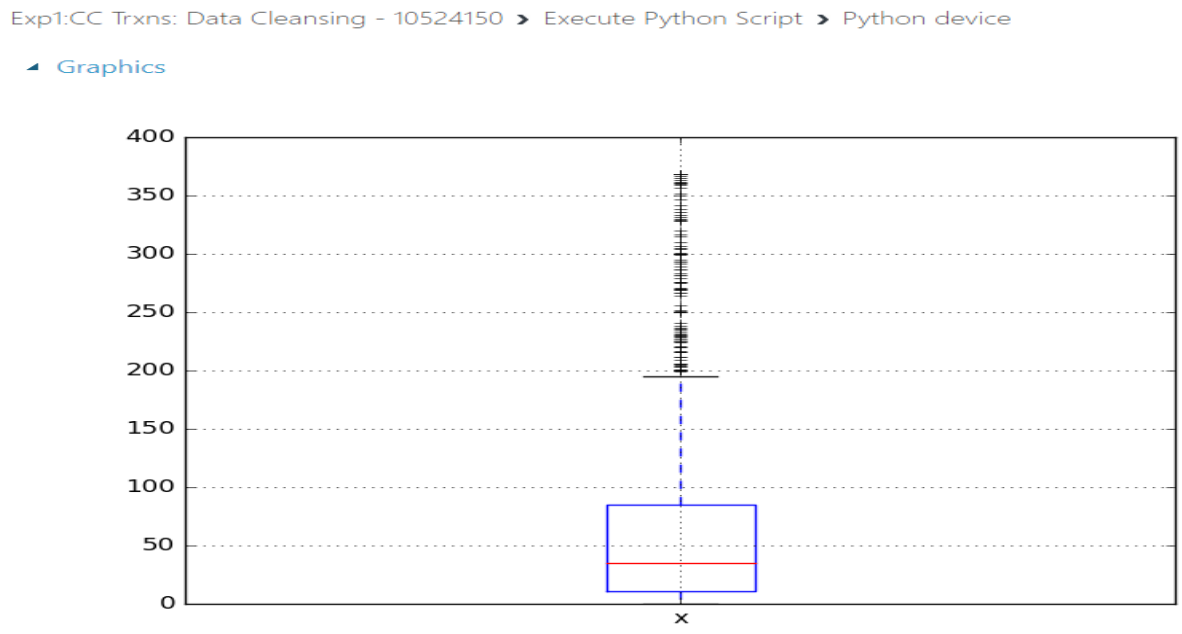
An embedded Python routine checks for outliers on cc transaction amounts and outputs a Box Plot representation, which is heavily skewed by transaction amount outliers.

<images>



The 'Clip Values' module caps the top 5% of amount values to reduce the distortion that outliers on transaction amounts would introduce to the predictive credit card model.



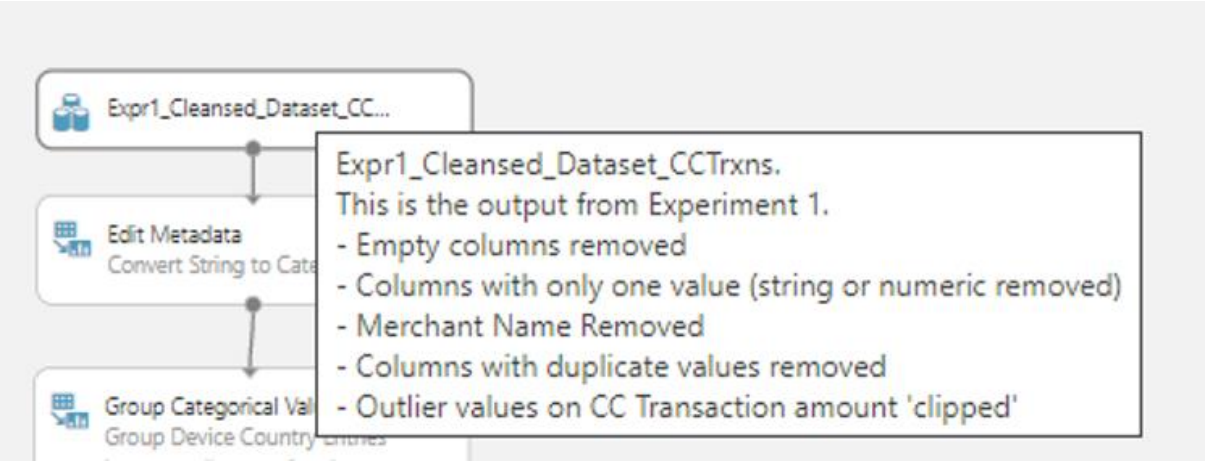


Running another Python routine shows how the impact of outliers is being reduced in the dataset in terms of the transaction amounts for each credit card row.

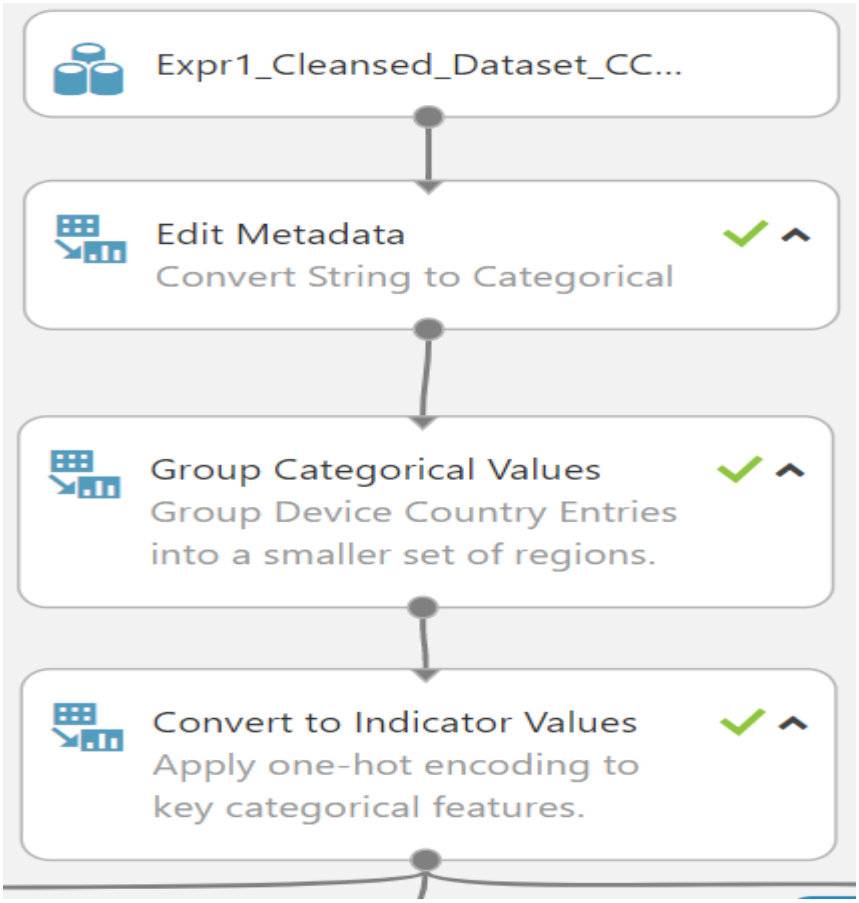
9.2.2. Experiment 2: Breakdown

Breakdown of Experiment

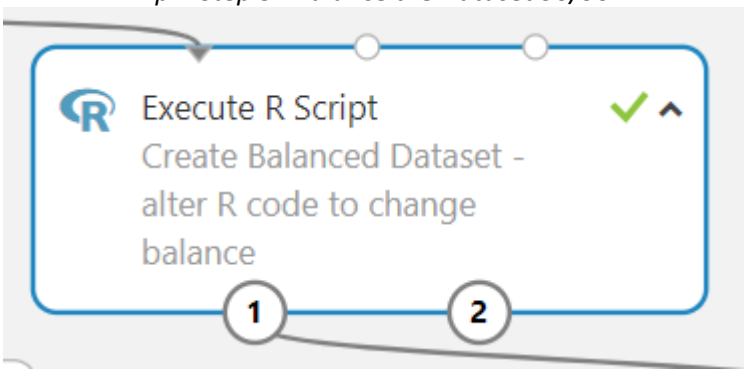
Exp2: Step 1. Take Input from Experiment 1



Exp2: Step 2. Convert Strings to Categorical Values / Encode.



Exp2: Step 3. Balance the Dataset 50/50.



Properties Project

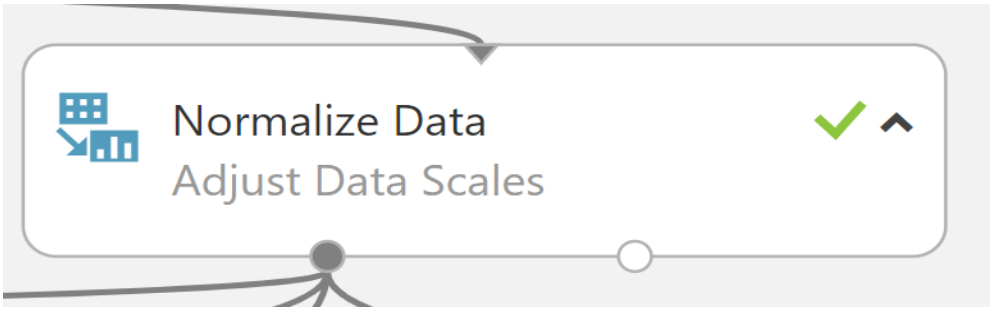
Execute R Script

R Script

```r
# C.Finnegan - August 2020 - Data Selection Code (Balancing) - 10524150

# Map 1-based optional input ports to variables
dataset1 <- maml.mapInputPort(1) # class: data.frame


# Set up two datasets split on Fraud label
dataset_Fraud <- dataset1[dataset1[,'Fraud']==1,];
dataset_nonFraud <- dataset1[dataset1[,'Fraud']==0,];


# Set sample size for non-Fraud dataset based on size of Fraud dataset
# Apply adjustment factor as required; default is '1' - equal balance
iBalanceAdj = 1
sample_size <- (nrow(dataset_Fraud) * iBalanceAdj);


# Extract a random sample of non-Fraud entries from full Fraud dataset
# A random selection of values is prefered to avoid possible bias with the
# sequence of entries in original dataset
dataset_nonFraud <- dataset_nonFraud[sample(1:nrow(dataset_nonFraud), sample_size,
   replace=FALSE),]

# Combine datasets to create balanced output dataset
data.set <- rbind(dataset_nonFraud, dataset_Fraud)

# Select data.frame to be sent to the output Dataset port
maml.mapOutputPort("data.set");
```

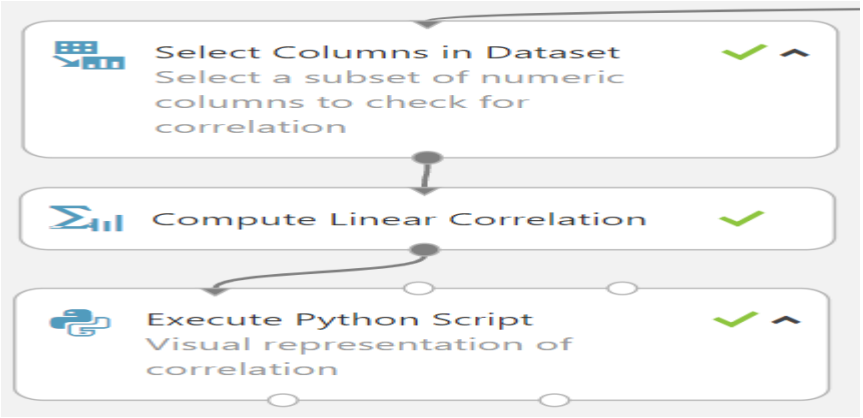
This embedded R code routine takes all the fraud data and extracts a random sample of the same size from the larger sub-set of non-Fraud transactions. The resultant combined dataset is balanced 50/50 and output to the next module.

Exp2: Step 4. Normalize Data



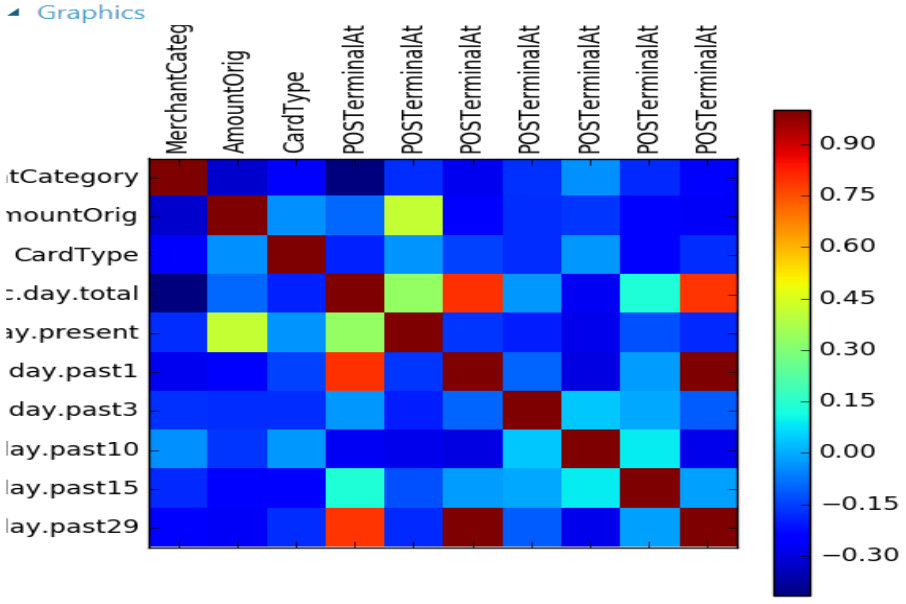
Exp2: Step 5. Check Numeric Columns for high Correlation

<Image>



Exp2:CC Trxns: Feature Engineering - 10524150 › Execute Python Script › Python device

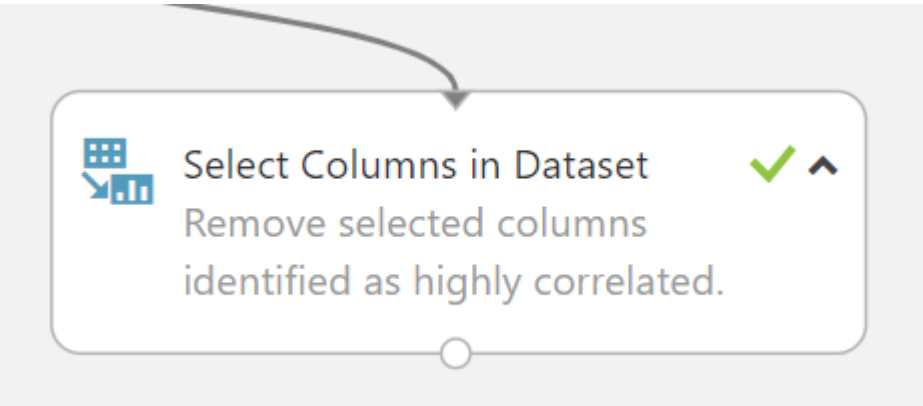
Graphics



Features that are highly correlated with other features are largely redundant and can be removed.

This task is iterated through a series of times and feeds into the final module that eliminates some of these highly correlated features.

Exp2: Step 5. Remove Certain Highly Correlated Columns



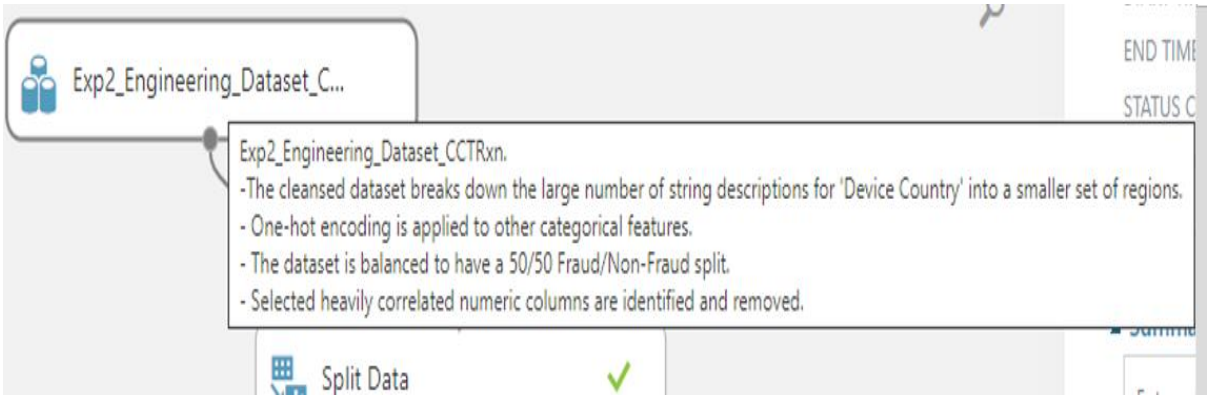
The Feature Engineering process brings the number of columns in the dataset up to 255, despite the removal of a sub-set of features.

Strings features have been converted into categorical values and further encoded into numbers so that the subsequent algorithms can build reliable models more efficiently.

9.2.3. Experiment 3: Breakdown

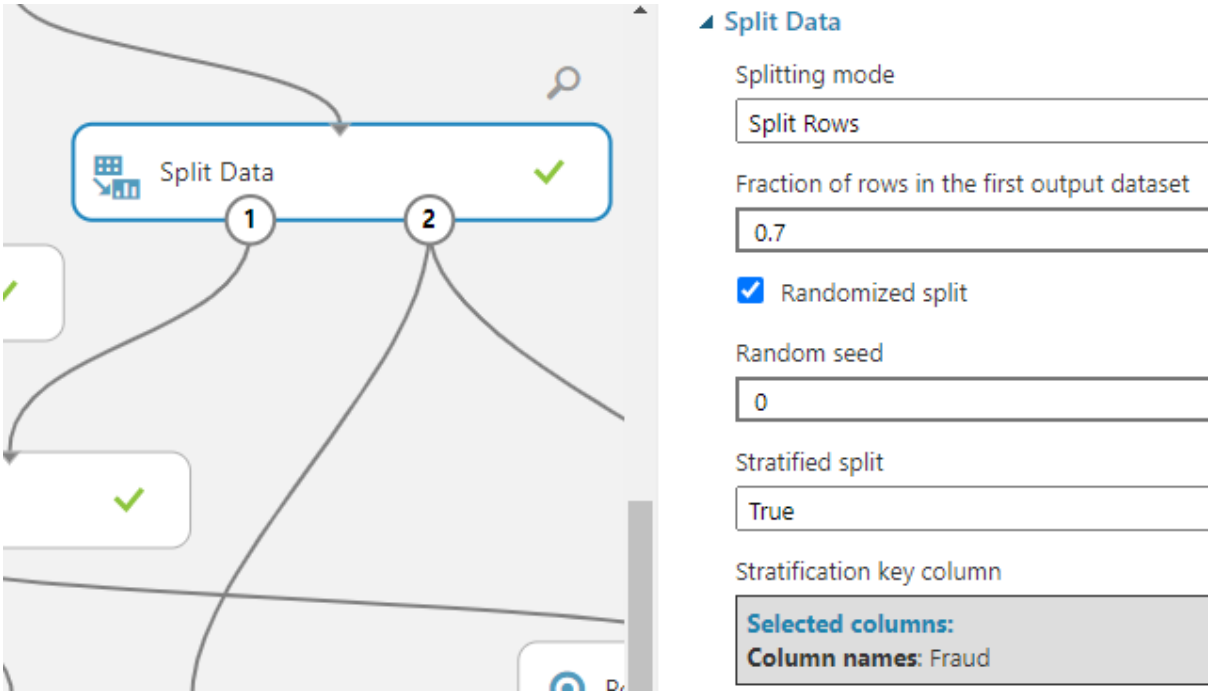
Breakdown of Experiment

Exp3: Step 1. Take Input from Experiment 2



Experiment 2 forms the basis for working out a feature selection list in Experiment 3.

Exp3: Step 2. Split Data into Training and Test Data

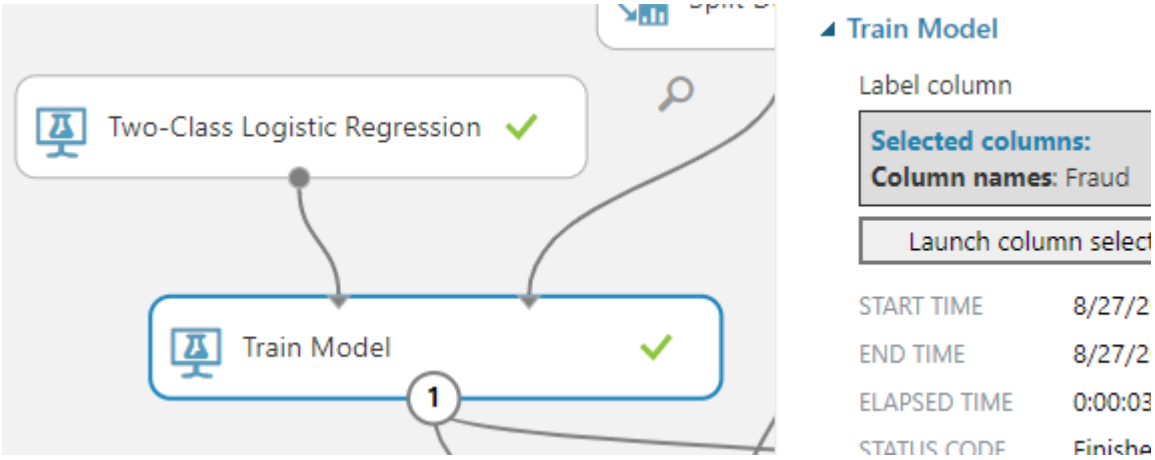


70% of the data is used to train the model. The remaining 30% is used to test the accuracy of the model. The Fraud data is split in proportion across the Train and Test dataset.

Exp3: Step 3. Set up Algorithm for Modelling

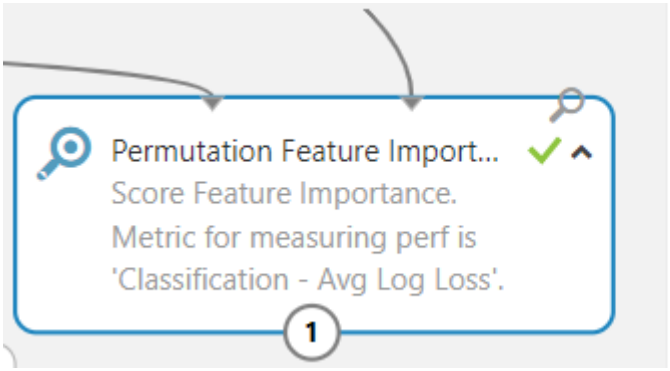
Two-Class Logistic Regression is the chosen algorithm. Parameters are left at the default assigned by Azure Machine Learning Studio.

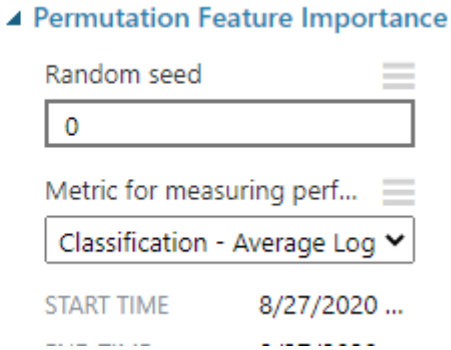
Exp3: Step 4. Set Up Training Module



The label to predict against is the 'Fraud' column.

Exp3: Step 5. Score Features by Importance





Exp3:CC Trxns: Feature Selection - 10524150 › Permutation Feature Importance › Feature importance

rows 254 columns 2

view as

Feature	Score
PinIndicator	0.102085
ECommerceFlag-Y	0.030885
ECommerceFlag-U	0.022542
AuthId	0.019177
DeviceZone	0.016133
EMVTransactionsCount.cnt.day.present	0.010882
HighRiskPOSCnt.cnt.hour.present	0.010417
CardSourceRefId	0.008589
AccountSourceUniqueId	0.008589
CustomerPresentIndicator-Y	0.008507
AcquirerRefId	0.005939

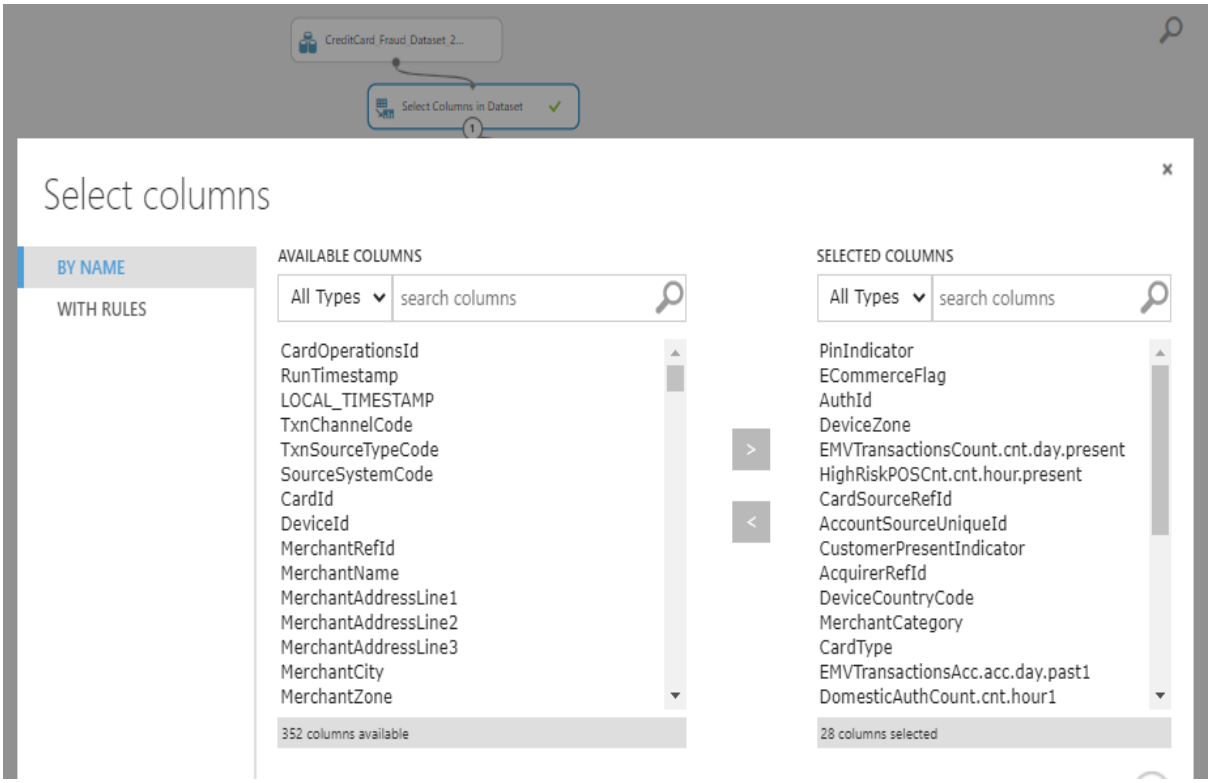
The output of the Permutation Feature Importance module provides an ordered list of the features ranked by their importance in determining the scored result of the training credit card fraud predictive model.

This list will influence the column selection process as later experiments build up for the final production model.

9.2.4. Experiment 4: Breakdown

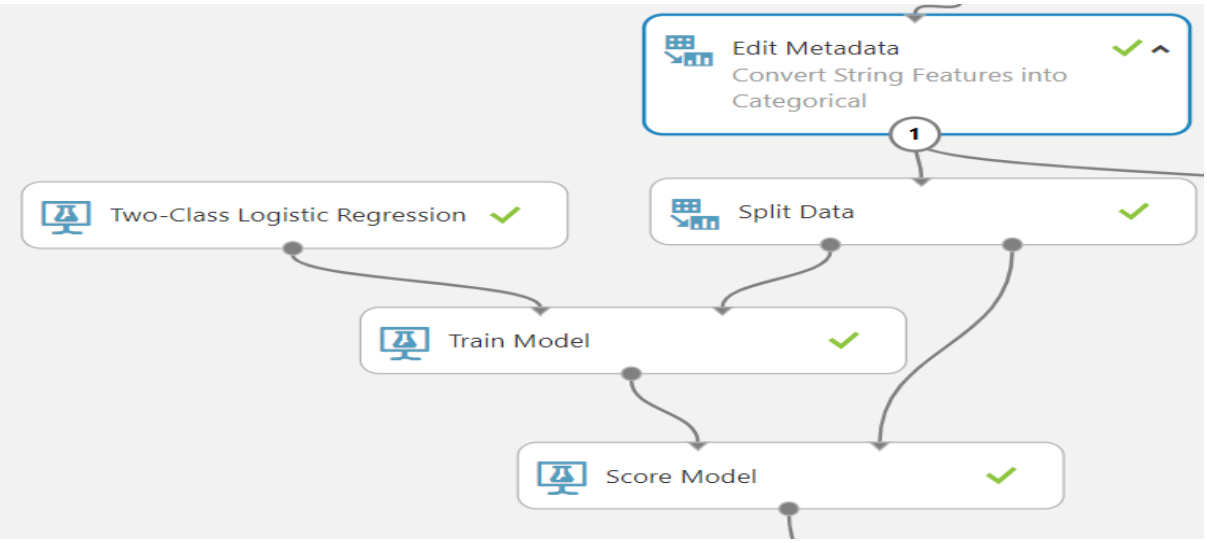
Breakdown of Experiment

Exp4: Step 1. Reload 25K Dataset – Apply Feature Selection

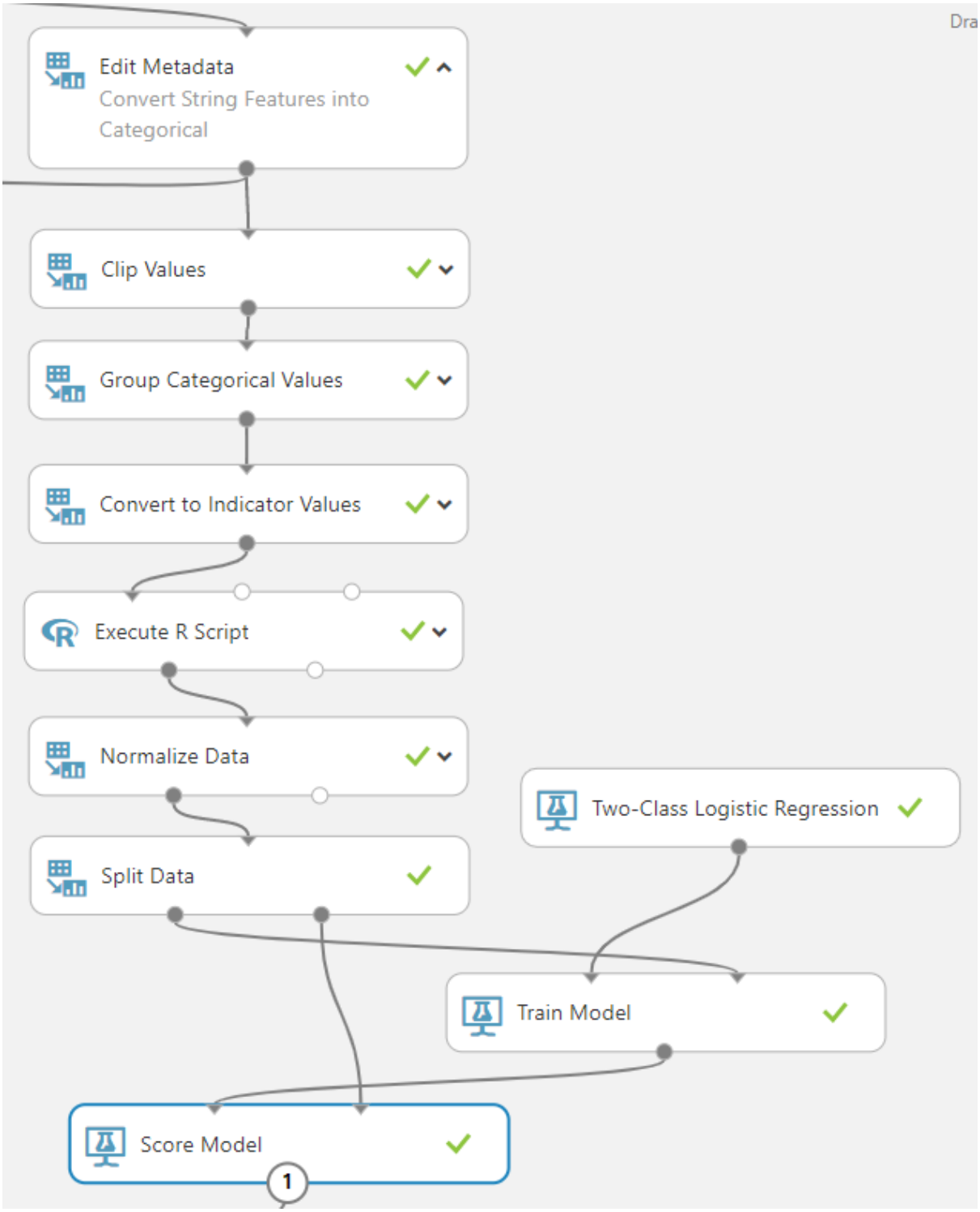


Out of the original 380 columns, we select 28 on which to train the predictive credit card fraud model.

Exp4: Step 2. LHS – Train Model with NO Feature Engineering



Exp4: Step 3. RHS – Train Model with Feature Engineering

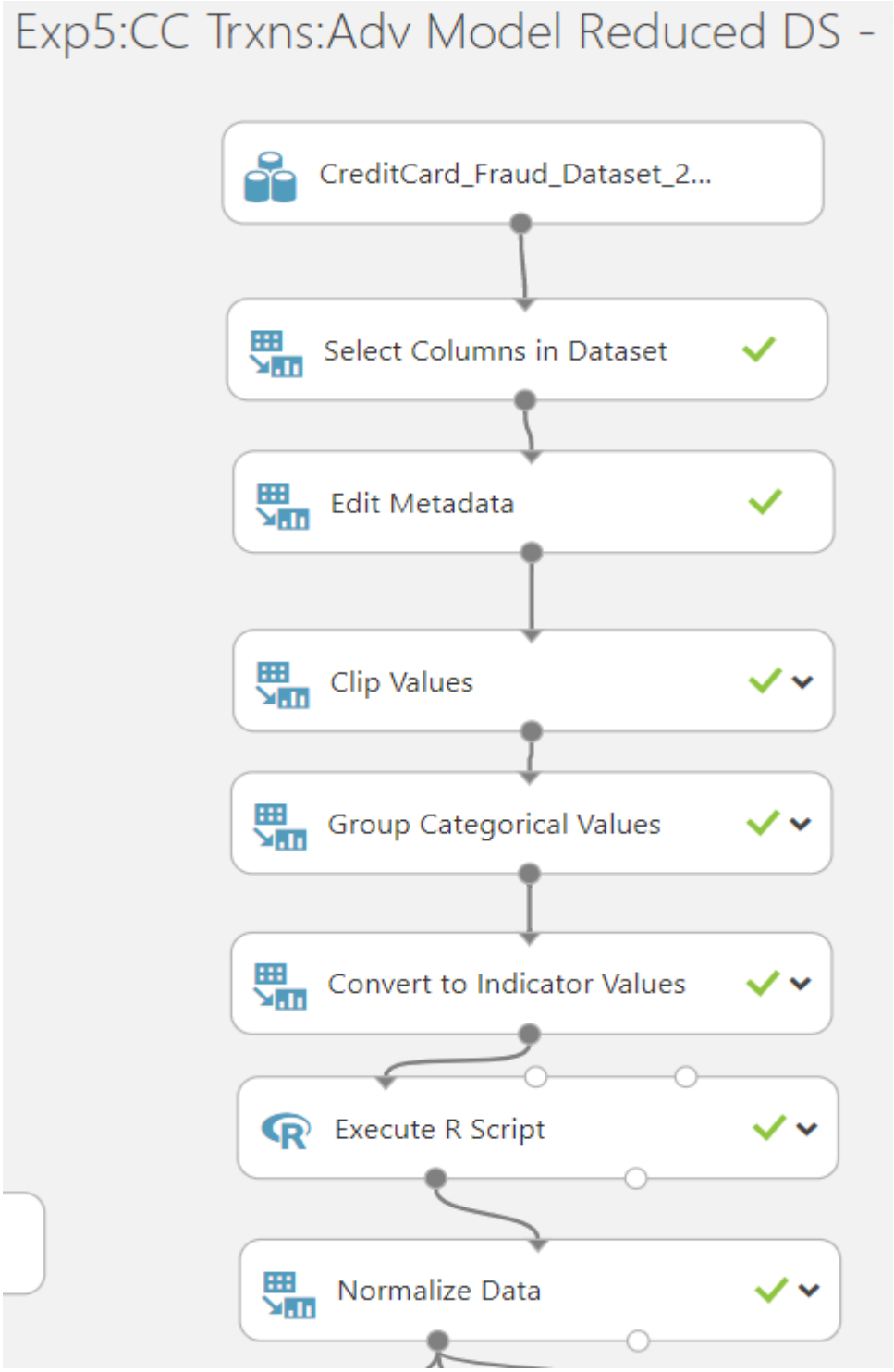


The separate training modules are set up so that we can see the difference in performance metrics when Feature Engineering is applied and when it is not.

9.2.5. Experiment 5: Breakdown

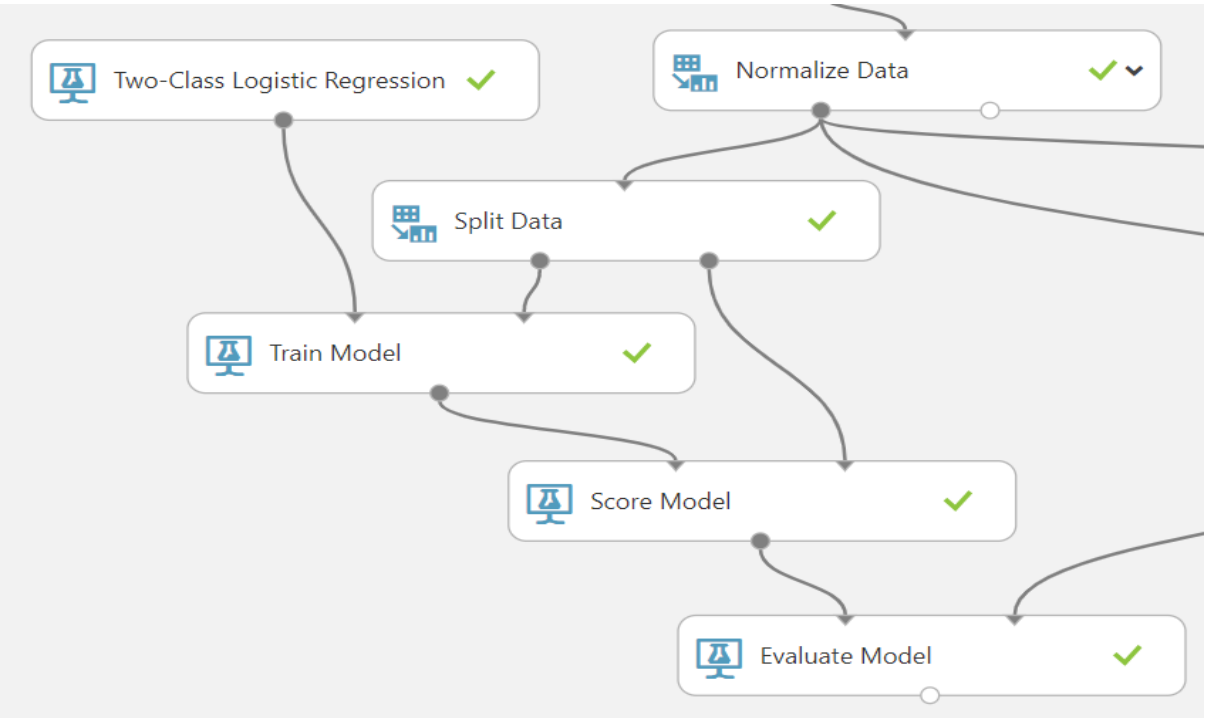
Breakdown of Experiment

Exp5: Step 1. Feature Engineering (Done before ALL Modelling)

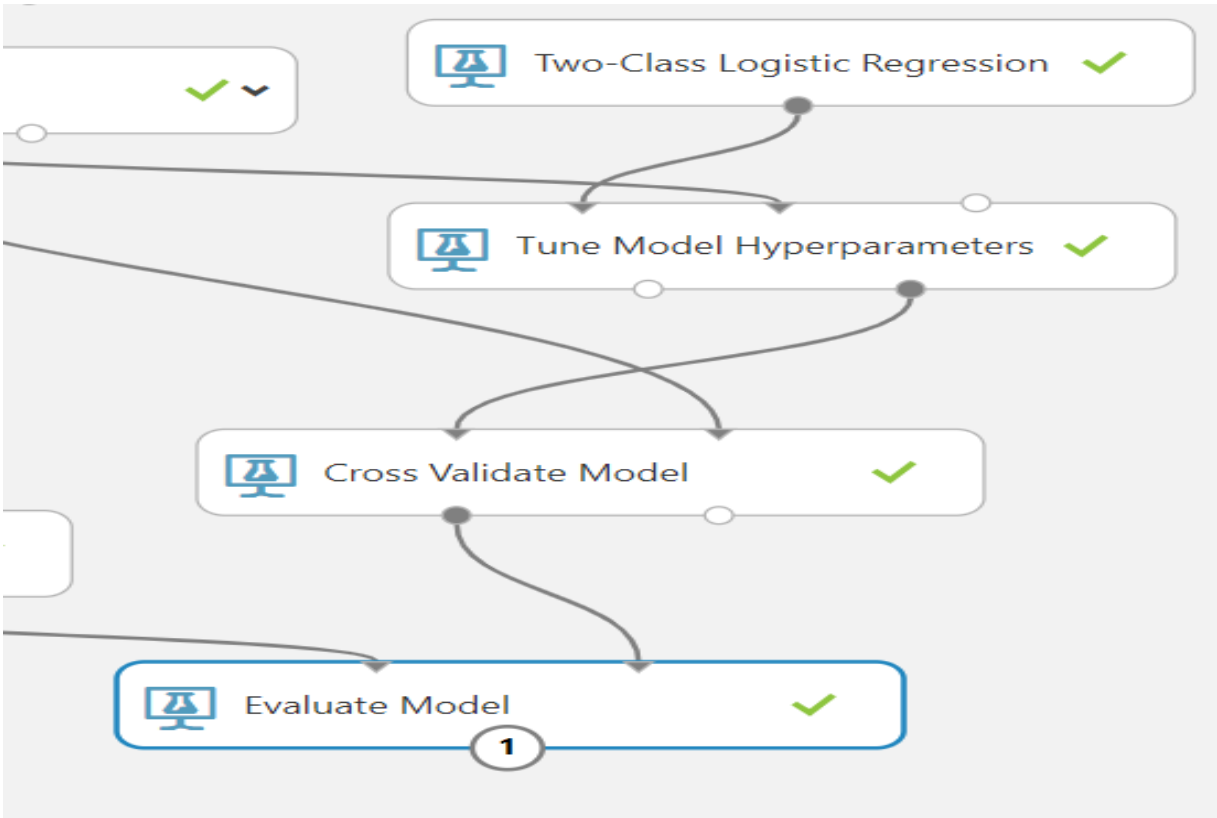


The previous experiment has shown the benefit of Feature Engineering, which will now be applied to all experiments going forward.

Exp5: Step 2. Train Model – Single Data Split / No Tuning



Exp5: Step 3. Train Model –Cross Validation / Tuning



The purpose of this experiment is to show the benefit in improved training of the credit card fraud model by applying Cross Validation and Hyperparameter routines to the process.

9.2.6. Experiment 6: Breakdown

Evaluation results for each classification algorithm

Exp 6 – LHS

Two-Class Averaged Perceptron - v – Two-Class Boosted Decision Tree

Two-Class Averaged Perceptron

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3617	264	0.926	0.920	0.5	0.970
False Positive	**True Negative**	**Recall**	**F1 Score**		
313	3568	0.932	0.926		
Positive Label	**Negative Label**				
1	0				

Two-Class Boosted Decision Tree

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3626	255	0.923	0.913	0.5	0.970
False Positive	**True Negative**	**Recall**	**F1 Score**		
344	3537	0.934	0.924		
Positive Label	**Negative Label**				
1	0				

Exp 6 – RHS

Two-Class Support Vector Machine - v – Two-Class Logistic Regression

Two-Class Support Vector Machine

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3476	405	0.876	0.862	0.5	0.945
False Positive	**True Negative**	**Recall**	**F1 Score**		
555	3326	0.896	0.879		
Positive Label	**Negative Label**				
1	0				

Two-Class Logistic Regression

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3639	242	0.926	0.916	0.5	0.973
False Positive	**True Negative**	**Recall**	**F1 Score**		
333	3548	0.938	0.927		
Positive Label	**Negative Label**				
1	0				

9.2.7. Experiment 7: Breakdown

Evaluation results for each classification algorithm

Exp 7 – LHS

Two-Class Decision Forest - v – Two-Class Decision Jungle

Two-Class Decision Forest

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3495	386	0.862	0.837	0.5	0.935

False Positive	True Negative	Recall	F1 Score
683	3198	0.901	0.867

Positive Label	Negative Label
1	0

Two-Class Boosted Decision jungle

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3453	428	0.812	0.770	0.5	0.899

False Positive	True Negative	Recall	F1 Score
1031	2850	0.890	0.826

Positive Label	Negative Label
1	0

Exp 7 – RHS

Two-Class Locally Deep Support Vector Machine - v – Two-Class Neural Network

Two-Class Locally Deep Support Vector Machine

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3600	281	0.920	0.914	0.5	0.956

False Positive	True Negative	Recall	F1 Score
339	3542	0.928	0.921

Positive Label	Negative Label
1	0

Two-Class Neural Network

True Positive	False Negative	Accuracy	Precision	Threshold	AUC
3657	224	0.925	0.911	0.5	0.971

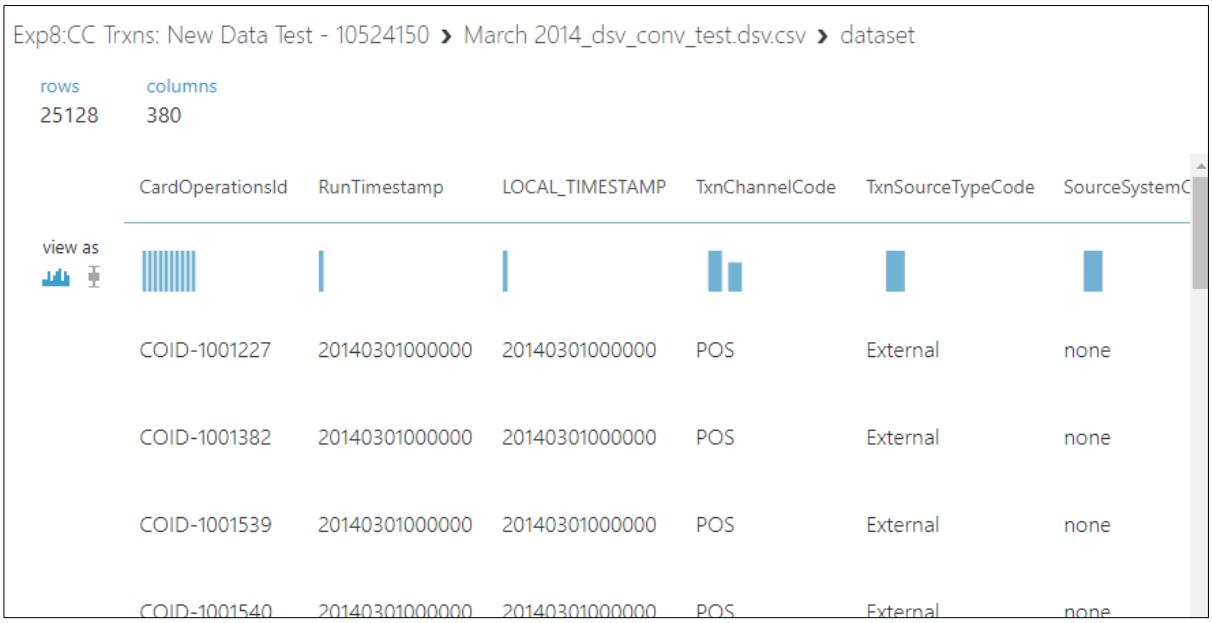
False Positive	True Negative	Recall	F1 Score
357	3524	0.942	0.926

Positive Label	Negative Label
1	0

9.2.8. Experiment 8: Breakdown

Breakdown of Experiment

Exp8: Step 2. Repeat Earlier Experiments with Larger Dataset

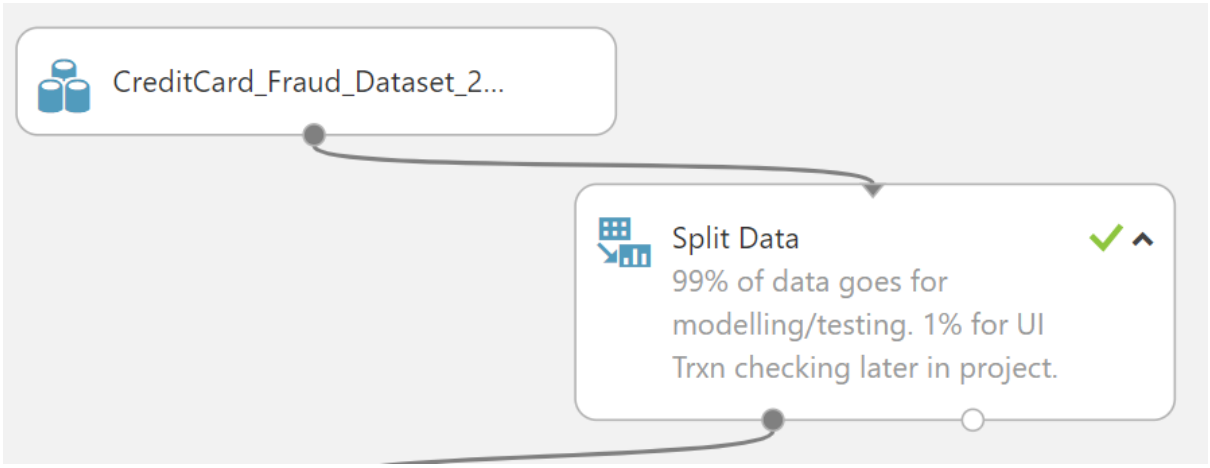


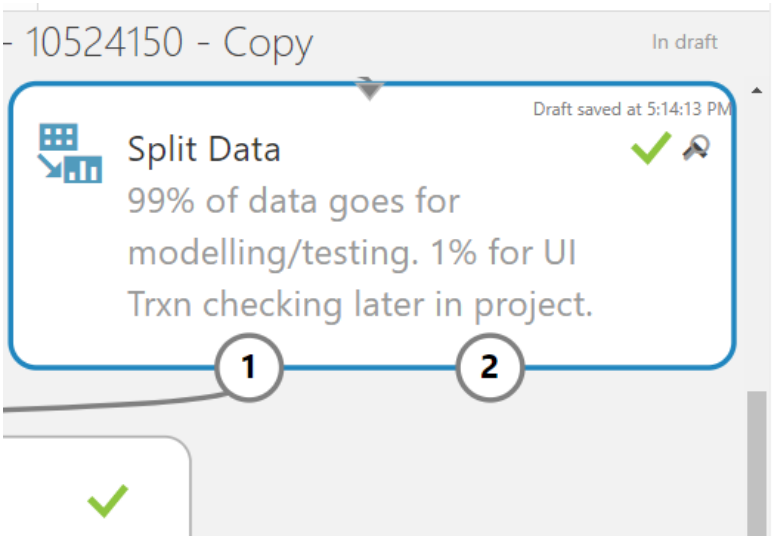
Earlier experiments are repeated for Feature Engineering on the credit card fraud dataset, but this time on the full dataset of 25K records.

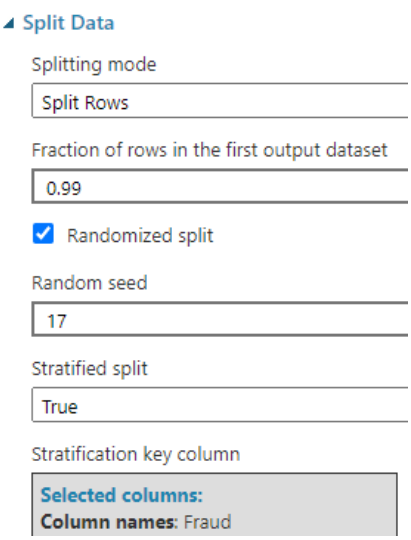
9.2.9. Experiment 9: Breakdown

Breakdown of Experiment

Exp9: Step 1. Extract 1% of Data – Train with 99%





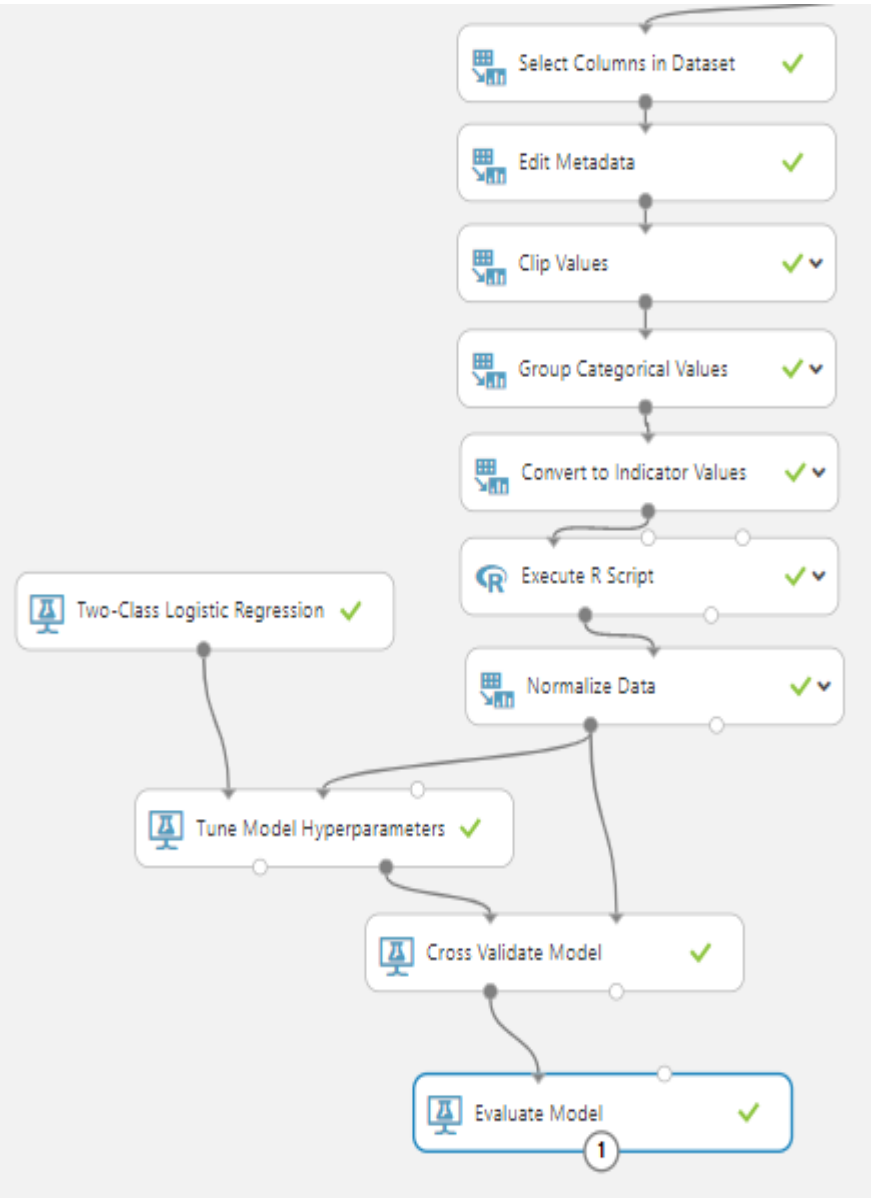


To train the model that will be deployed into production, 99% of the credit card fraud dataset is assigned to the training process.

The remaining 1% is taken to be used a 'new' data in the Shiny App UI. This data represents credit transactions that have not been used in the modelling process and can be considered 'unseen' data when the model is invoked through the project UI.

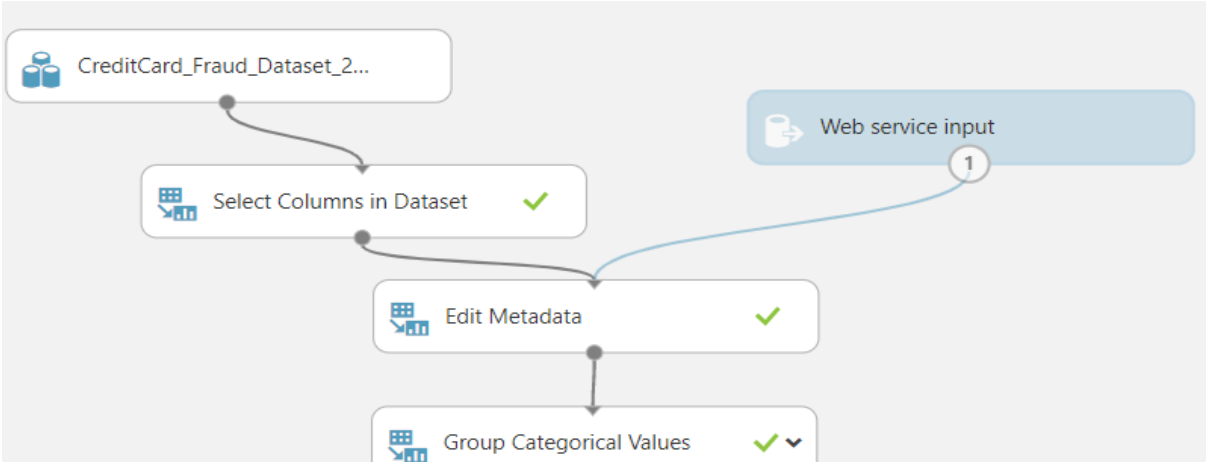
The 'Split Data' module is configured so that the proportion of Fraud/non-fraud records is maintained in both new datasets.

Exp9: Step 2: Train Model based on Previous Experiments



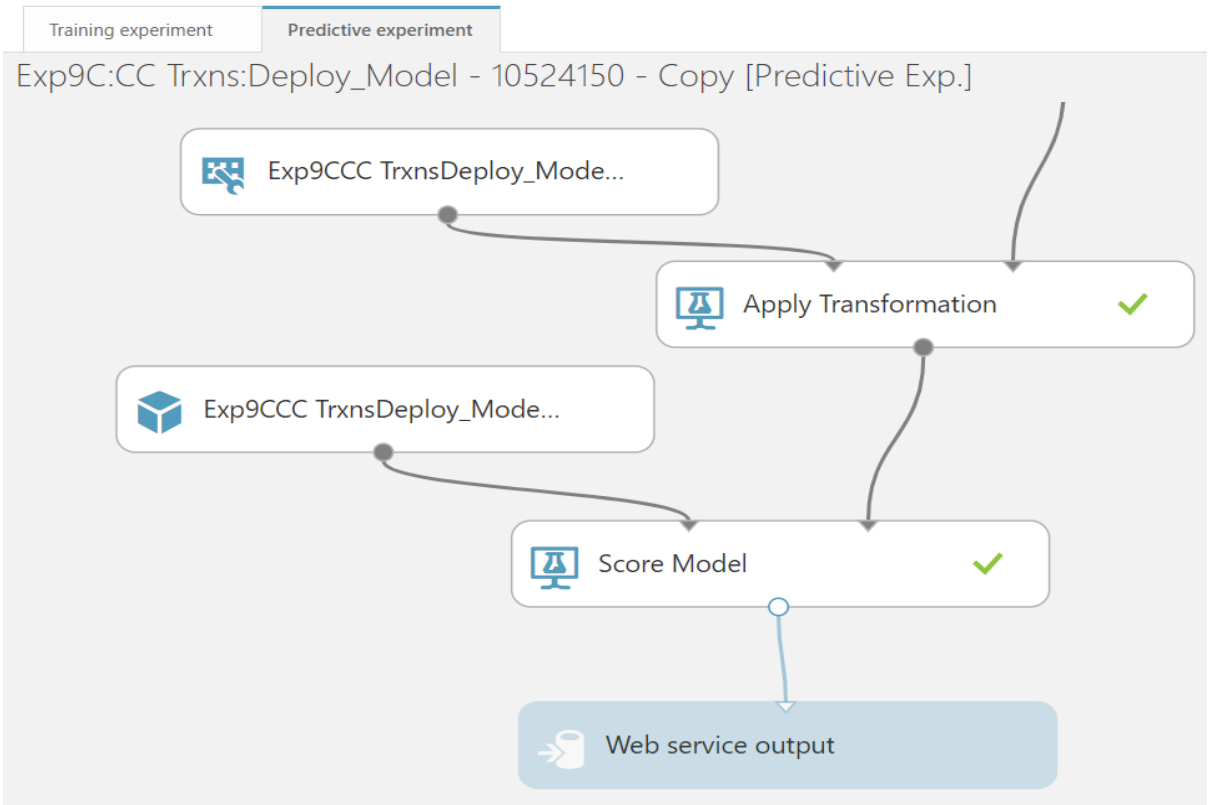
Applying all the learnings and outputs from previous experiments I have trained a credit card fraud predictive model ready for Production.

Exp9: Step 3: Configure Web Service Input



Once the Predictive experiment is created it is necessary to switch the Web service input so that it will expect the required 28 features, and not default to the original feature set of 380 from the starting dataset.

Exp9: Step 4: Configure Web Service output



The Predictive experiment condenses the Feature Engineering steps for numerical data into a single transformation and then applies the trained model.

The results of the model are then passed to the Web service output through which the data can be consumed by an external source.

9.3. Credit Card Fraud Datasets

The datasets are too large to package in this document, but this sample CSV file is included with the final project submission.

CreditCard_Fraud_
Dataset_NewRows_S

The Azure workspace, in which the model was trained and deployed, contains all the datasets used during the development of this project.

A zip file containing all 10 of the 'new' transaction data files is also included with this project submission. These are the only data files that need to be stored locally by the user.

NewDailyData.zip

10. References / Bibliography – Final Report

Bhattacharyya, S., Jha, S., Tharakunnel, K. and Westland, J., 2011. Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3), pp.602-613.

Ceronmani Sharmila, V., R., K., R., S., D., S. and R., H., 2019. Credit Card Fraud Detection Using Anomaly Techniques. *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, [online] 1(1), pp.1-4. Available at: <https://ieeexplore.ieee.org/document/8741421> [Accessed 11 September 2020].

Kurata, J., 2016. *Getting Started With Azure Machine Learning*. [online] App.pluralsight.com. Available at: <https://app.pluralsight.com/library/courses/azure-machine-learning-getting-started/table-of-contents> [Accessed 9 June 2020].

Lateef, Z., 2020. *Data Science Vs Machine Learning: What's The Difference? | Edureka*. [online] Edureka. Available at: <https://www.edureka.co/blog/data-science-vs-machine-learning/> [Accessed 9 September 2020].

Lima, R. and Pereira, A., 2017. Feature Selection Approaches to Fraud Detection in e-Payment Systems. *Lecture Notes in Business Information Processing*, [online] pp.111-126. Available at: <https://www.researchgate.net/publication/313731885_Feature_Selection_Approaches_to_Fraud_Detection_in_e-Payment_Systems> [Accessed 11 September 2020].

Lucas, Y., Portier, P., Laporte, L., He, L., Caelen, O., Granitzer, M. and Calabretto, S., 2019. *Towards Automated Feature Engineering For Credit Card Fraud Detection Using Multi-Perspective Hmms*. 1st ed. [ebook] Lyon: Research Gate, pp.4-6. Available at: <https://www.researchgate.net/publication/335600419_Towards_automated_feature_engineering_for_credit_card_fraud_detection_using_multi-perspective_HMMs> [Accessed 1 September 2020].

Mahmoudi, N. and Duman, E., 2015. Detecting credit card fraud by Modified Fisher Discriminant Analysis. *Expert Systems with Applications*, [online] 42(5), pp.2510-2516. Available at: <https://www.semanticscholar.org/paper/Detecting-credit-card-fraud-by-

Modified-Fisher-Mahmoudi-Duman/1cfb2a0a0f11dab8da5dc38d51a6e816f04ac8e3#paper-header> [Accessed 11 September 2020].

Microsoft, 2019. *ML Studio (Classic): Two-Class Averaged Perceptron - Azure*. [online] Docs.microsoft.com. Available at: <https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/two-class-averaged-perceptron?redirectedfrom=MSDN> [Accessed 10 September 2020].

Microsoft, 2019. *ML Studio (Classic): Two-Class Support Vector Machine - Azure*. [online] Docs.microsoft.com. Available at: <https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/two-class-support-vector-machine?redirectedfrom=MSDN> [Accessed 10 September 2020].

Narayanan, S., 2019. *Azure Machine Learning Studio Vs Azure Machine Learning Services | Codit*. [online] Codit. Available at: <https://www.codit.eu/blog/azure-machine-learning-studio-vs-services/?country_sel=uk> [Accessed 9 September 2020].

Rhodes, J., 2020. *Building Your First Data Science Project In Microsoft Azure*. [online] App.pluralsight.com. Available at: <https://app.pluralsight.com/library/courses/microsoft-azure-building-first-data-science-project/table-of-contents> [Accessed 3 July 2020].

Sonobe, S., 2017. *Microstrategy Community*. [online] Community.microstrategy.com. Available at: <https://community.microstrategy.com/s/article/How-to-Visualize-Azure-Machine-Learning-Models-with-MicroStrategy?language=en_US> [Accessed 10 September 2020].

Srinivasulu, R., 2019. *Preparing Data For Feature Engineering And Machine Learning In Microsoft Azure*. [online] App.pluralsight.com. Available at: <https://app.pluralsight.com/library/courses/microsoft-azure-preparing-data-feature-engineering-ml/table-of-contents> [Accessed 5 July 2020].

Whatman, P., 2019. *Credit Card Statistics 2020: 65+ Facts For Europe, UK, And US*. [online] Blog.spendesk.com. Available at: <https://blog.spendesk.com/en/credit-card-statistics-2020> [Accessed 19 September 2020].