GCE Computer Science

H446 - 03

Programming Project: Financial Fraud Detection Software

Name: Evan O'Leary

Candidate Number: 2152

Centre Number: 14613

Centre Name: Orleans Park School

Section 1. Analysis

Description of the problem

Credit card fraud is the general term for whenever a criminal makes a purchase using a payment card that is either counterfeit or obtained unlawfully.

A card holder could lose access to their physical card through either theft or unknowingly reveal their card information to a criminal or piece of malware by the means of phishing, hacking, or even social engineering. A customer's payment card information could also be obtained through a 'skimmer'. A small device usually hidden by criminals around ATM's which covertly scans the payment card's magnetic strip, which stores all of its information. The card information is usually sold online and used by multiple criminals from possibly anywhere in the world. This can be stopped by the card owner reporting any suspicious activity on the account to the bank or card company. Then the hijacked card will be deactivated while a new one is safely reissued.

However this won't happen if customers aren't regularly checking their statement online and if this is delivered through the post, it'll only happen on a monthly basis. This means that it is up to banks and card companies to analyse all the transactions that pass through their network so that they can detect fraudulent seeming behaviour as quickly as possible and alert at-risk customers. Solving cases of fraud is difficult and unlikely which is evident from the fact that less than 4% of the investigated fraud cases were actually solved in 2016/2017. Therefore the importance of reporting criminal activity to agencies like Action Fraud is only second to protecting potentially vulnerable customers. That is the bank's main responsibility and priority. Data about the process of reporting these crimes to both users and authorities are kept confidential by organisations and our solution will not encompass this part of the practice. Therefore, the sole aim of our project will be to detect fraudulent payments in an anonymised data set of transactions.

Stakeholders

A stakeholder is anyone with an interest in the actions of a business or organisation, and in the case of credit card fraud detection, there may be several potential stakeholders. Every year, banks, merchants, customers and financial services collectively lose billions to credit card fraud. In fact, UK banks and card companies suffered a total loss of £845 million during 2018 alone. This is why banks, merchants or payment card companies employ several fraud analysts to find instances of credit card fraud using software and statistical techniques and then set preventative measures to ensure that particular method of fraud doesn't reoccur.

Since that it is only fraud analysts that will be using and relying on the program I will be creating, they will be the sole stakeholders. Part of a fraud analyst's job description is to flag

the accounts where fraudulent activity is detected and then take action to keep it suspended until it can be checked and verified. This means they are accountable for the action of either falsely depriving a customer of the access of their account or failing to prevent money being stolen from the customers. This puts more importance on the final accuracy of the program for analysts. Although customers and even fraudsters are directly affected by the software's performance, they can't be considered stakeholders as they will not directly interact with the program.

An interview with the stakeholder.

The fraud analyst and main stakeholder in our example will be personified by Yorke O'Leary. He is currently employed at Citibank and is a member of my family. Since Yorke has worked at several other organisations like HSBC, he can identify the essential features consistent in all industry-standard pieces of software. As Yorke has worked for many banks and is still currently employed, he has up-to-date expertise in the requirements for a piece of software used in the finance industry.

The main aim of this interview is to determine the features and format of our program design. The purpose of this interview was not to change the design of the fraud detection process itself.

Transcript

Yorke was emailed the questions prior to the interview to give him time to properly think about his response. The interview itself was conducted in-person and the audio was recorded. I have transcribed this recording below.

Question 1

Interviewer: In your experience, is the graphical interface of most banking programs in the form of a web application or a piece of software installed directly onto their local computers?

Yorke: They are a mixture of web apps and software apps. The native software apps tend to be the older applications and the new web applications tend to be the new ones. There are now some apps built in a 'chromium' wrapper that are basically web apps that appear to be installed software as it takes the web app and makes it look a bit nicer with removal of the address bar etc. Much neater in a world of limited screen real-estate is vitally important. Also it's more difficult I think to get sign off from banks and their customers for new software or web apps so there are some web services that provide the authentication and other

requirements to comply with regulations. A good example of this is called OpenFin. If apps are built in that then many organisations have already signed off on open fin so it becomes much more straightforward. I would say now that it's 5050 web/software apps. But in the next 5 years will be more like 60:40 in favour of web apps. It's a slow process because the build costs to replace software are so high and they are very ingrained in the bank's operations / reporting etc.

Question 2

Interviewer: Do other programs that run live in accordance with the marketplace, tend to display their results in real-time or on a scheduled time interval?

Yorke: There are varying degrees of data updates ranging from microseconds to a week. Prices for financial instruments like forex prices are typically micro-seconds. We are streaming prices electronically to systematically dri trading algorithms so it needs to be as close to instantaneous as possible t protect the bank from losing money (and to be as competitive as possible). These micro-seconds don't only show the pricing but also detail the custom and any specific information about the relationship i.e do they have credit li in place and can we trade with them. Because as they trade, the banks willingness to trade with them more can change. Then there are less vital b of data such as sales coverage of particular accounts to reflect internally th revenue attributable to particular clients. This is less urgent so is done on a daily basis. Also profit and loss of a bank's positions can similarly range fro seconds for the people managing the risk to daily for the accounts checkin all at the end of the day.

Question 3

Interviewer: Would the number of transactions coming in across the day be viewable graphically by a time series and are analytics such as pie-charts and bar graphs useful visual graphics?

Yorke: Yes some have charts to reflect time series but also have the ability to download the data in a csv file too. I think flexibility is important for different people.

Interview Analysis

The first two questions concern the format of our program. In the web-development and software engineering industries, the part of a computer system or application with which the user interacts directly is defined as the front-end.

Question 1 decides the format of the front-end of our program. His answer clearly explained that the industry is slowly moving its software into web applications. He stated that the ratio of native software to web applications is currently “50 : 50” but in the next 5 years will move to become “60 : 40”. Therefore, to future-proof my program, I will be creating the front-end as a web application. Web apps can also be opened universally on any operating system granted the user has authentication.

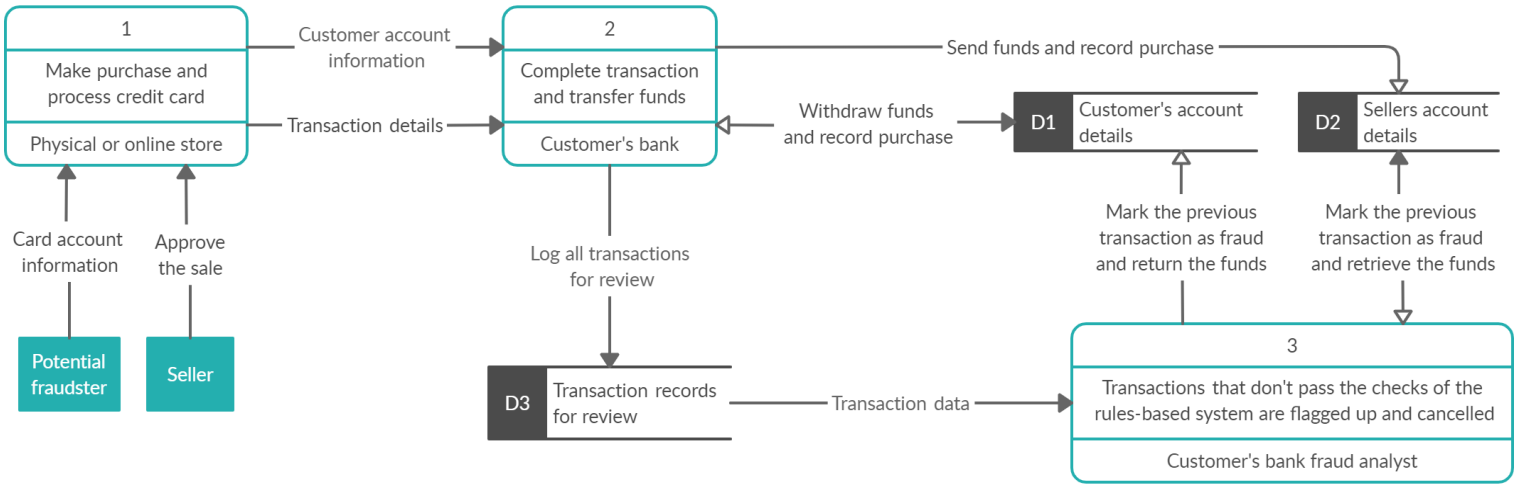
Question 2 decides the actual nature of how the application will run: whether that may be a live-simulation of detecting fraud or simply just a days report. Even though Yorke said that it can be displayed in a matter of “microseconds”, he said it can also be up to a couple of days

Research about an existing system

Let's see when this solution would be used in a real-world context. Firstly, after a payment is made using a payment card at an online or physical store (process 1), the bank digitally transfers the funds between the accounts of the buyer and the seller (process 2). The transaction’s details are recorded in a database which is labelled D3 in the diagram. This database is passed into process 3 to be checked for fraud. This is what we are interested in programming.

A static rules-based is the name of an industry standard method of detecting fraud. A set of rules used to score a transaction’s likelihood of being fraudulent on. These checks are decided upon by an analyst and the detection process itself is automated through a program. This is how the program would work. As you can see from the diagram, the database of transactions (D3) must be imported from the bank’s secure file server.

We’ll say that the file is stored in the Comma Separated Value (.csv) format. A function would turn this file into an array of dictionaries. Each dictionary represents a record of a transaction from the database and its keys map to the transaction’s characteristics. Then an empty array called ‘fraudDetected’ is created.

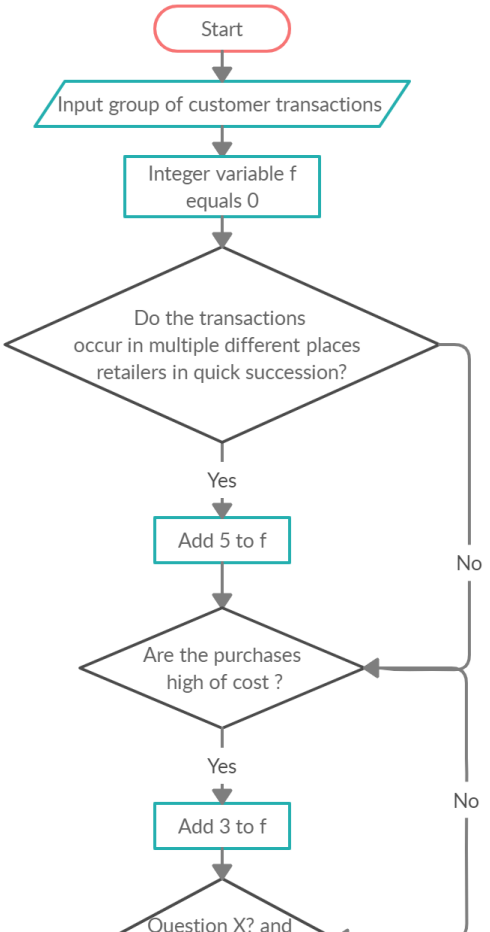


The program then loops over every transaction in the array. The process of each iteration is shown on the right and explained below.

At the start of each iteration, the integer variable ‘f’, which scores the potential fraudulency of the transaction, is initialized at 0. Then the transaction’s details are checked through a series of rules which in the program would be implemented as selection statements.

If the transaction meets a condition to be a fraudulent transaction, then ‘f’’s value is incremented by a certain value dependending on the weight of the question’s likeness to indicate fraudulency. After being checked by, if the score is higher than a threshold value then it is marked as fraud which is done through appending the transaction’s ID to the array ‘fraudDetected’.

At the end of the program's execution, the list of fraudulent transactions can be uploaded back to the file server and a notification be sent to the analysts dashboard. Alternatively this program could have a graphical user interface which displays a list of possible fraudulent transactions detected. It is up to the analyst to refund the transaction and lock the account.



Advantages of the current system

As it is an iterative solution which mainly consists of a series of selection statements. The program is simplistic in nature and would be, at first, easy to design. For example, once a new rule is conceived of, and if it doesn't conflict with any other existing rules, then it can be easily added at any place between the selection statements. Also, as the rules-based system is highly logical, practitioners in the industry are comfortable with this approach. These rules can be very effective in mitigating fraud risk.

The scale of the dataset is large and would not be a problem as the program could be designed to allow SIMD parallel processing so multiple transactions could be simultaneously checked against the same condition if the bank was to use many multicore computers.

Problems with the current system

There are three main reasons why the current system has become increasingly more inefficient and fragile.

Firstly, and most importantly, rules in their essence are limited. They are constrained to being a "yes" or "no" decision based on an absolute threshold value. Transactions have a lot of characteristics. These characteristics have hidden relationships which indicate fraudality. A simple example would be the average price of a series of payments linked to the time interval in which they occur. A manual probing of these hidden relationships between characteristics is difficult so consequently the rules created to capture them fail.

Secondly, due to rapid advances in technology, fraud attacks have become much more elaborate, thus requiring rules to become more complex and much more error-prone. Each new fraud scheme detected turns into a new ad-hoc rule making rules systems grow to uncontrollable sizes. Not only does this increase the execution time of the program but it also leads to a high amount of false positives being reported and a vast amount of false negatives being accepted and transacted. This causes a poor customer experience and an increase in operational costs and a significant reduction in revenue for the bank.

Thirdly, in the modern world, commerce and e-commerce are moving in large volumes across borders at a rapid velocity and customer bases around the world are getting bigger thanks to the standardisation of the internet. So, the amount of data in need of being regularly checked is growing larger and larger which makes the analysis process much more time-consuming. This gives potential for new fraud schemes to blend in with the huge volumes of data letting suspicious payments fall through the cracks. This longer average period of time until a fraud

attack may be detected risks more money being stolen from the account. This wastes the banks time and resources and also damages their reputation.

The combination of these three dilemmas results in analysts being torn on deciding if the checks aren't too under or over-sensitive, regardless of the expertise of the analyst. This creates a lag between identifying the need for a new rule and implementing it. Which only adds to the inefficiency of this system and again hints at the system's overarching problem: it's reliance on a team of analysts to manually update a model attempting to prevent thousands of criminals and enterprises seeking financial gain around the world.

What I will adapt

The final program must not require frequent alterations of the code to stay on top of new shifts in fraud attacks. Using an adaptation of the existing solution would be an unattainable goal as I do not have access to the same resources and expertise as a bank. Moreover, constant human intervention was the problem most persistently highlighted in the drawbacks of the previous approach. It was also an essential feature of the last approach. Therefore our proposed solution will be a complete overhaul of the existing system while still completing the same task.

Features of the Proposed Solution

The most important feature of the proposed solution is that it must be able to recognize subtle patterns that indicate fraudulency in the dataset by examining thousands of payment characteristics that may seem completely unrelated to a human being. Another criteria that is crucial for our program's success, is that it must have the ability to continuously examine and be informed by large quantities of new data. The software should be able to autonomously update itself to reflect the latest trends in fraud attacks and stay on top of new shifts in fraudulent activity. This means the program should be able to learn and become more effective through analysing incoming sets of data, all without being explicitly to do so. This is the definition of Machine Learning.

Using a ML model would address nearly all the prior issues of the rules-based system. For example, the earlier requirement of expensive human labour to enumerate all the possible detection rules over a long time period will be eliminated and replaced by autonomous detection. This lets the program handle new transactions in real-time allowing them to be blocked before they're finalised. This aspect of the proposed solution is by far the most attractive to banks as they can finally discontinue the preceding multiple validation steps which once harmed user-experience.

Why the problem is suitable for a computational solution

Firstly, a manual handwritten approach would not suffice for solving a problem which involves analysing a huge volume of data. In comparison, computers are far more superior at processing large datasets. In the modern world, thanks to the digital revolution and the rise of the internet, online payments and e-commerce, nearly all transactions are recorded digitally and stored in databases. This makes it easy to import into a program where algorithms can be repeatedly tested and adjusted until effective.

Contrary to humans, machines can perform repetitive tasks at an extremely fast rate and are far more superior at processing large datasets. This is a desirable attribute for a solution where speed is an important factor. This is because the longer fraudulent transactions go unseen, the more time fraudsters have to distribute stolen money and making it intangible to trace and return.

Moreover, as the previous approach was heavily dependent on human labour, computers can do the job around the clock without rest. Computers work around the clock. This well suits a finance rooted problem where transactions occur 24 hours a day. If payments were to be analysed in the hour they were first carried out, applying a manual approach would not be feasible without great expense.

Limitations of the Proposed Solution

One defect of using machine learning models is that if there is an undetected fraud in the training data, the model will train the system to ignore that type of fraud in the future. Also the dataset was recorded in 2013 and fraud methods may have new different fraud determinants so may not give a realistic demonstration if used today.

The dataset from Kaggle (a data set publisher) we're going to use has also undergone a process called Principal Component Analysis (PCA), a method of dimensionality reduction. The original data must have had originally hundreds of fields for every transaction. PCA reduced this by standardising the data (scaling the values to between 0 and 1) and then projecting a large set of data fields to a lower dimensional subspace while still keeping the useful data.

This as a result lets the learning algorithm train faster, reduces the complexity of a model and improves the accuracy of a model if the right features are chosen. However as most of the fields data have been combined and rescaled, it may appear meaningless to a fraud analyst without adequate context. This highlights a wider problem with using a non-rules based solution such

as our own. The algorithm cannot justify explicitly what specific variable caused it to predict a case as fraud except other known cases were similar.

Nevertheless, the bank, or Kaggle, would have had to carry out PCA in order to anonymise the dataset in order to maximise the security of their clients. Also, if we were to reapply the same techniques on the non-anonymised original dataset to detect fraud, our fraud detection methodology would work exactly the same. Therefore, from now on, we will assume that the fraud analyst is familiar with what each field represents when they are interacting with the final solution.

Hardware and Software Requirements

Hardware	Justification
A monitor	To display the program's statistical findings and to allow the analyst to see the program and interact with it.
A mouse	So the analyst can open the program.
A powerful machine with parallel processing capabilities (effectively having a GPU and multicore CPU installed).	Machine learning is parallel processing dependent and the program development will run much faster if so.
A plotting library for Python	So we can visualize the program's performance and analyse performance.
A large dataset of transactions containing fraudulent ones which are labeled as such (Kaggle)	So we can have sample data to train our method on and find fraud in.

Project Objectives

Setting objectives is important because they address what will be achieved as a result of the work, so that the stakeholders can see what you are working towards, and so that you can demonstrate achievement, by measuring and reporting on progress. They aid better communication and provide focus for effort over time.

Below is a table of all the objectives I'm going to set for each component of our project. It is a brief outline of what we are aiming to achieve and why. It should guide the premise of how the project is designed. It also needs to be completed in a particular order since the project's functionality is heavily interdependent. This is included as part of our analysis section as this is where we identify the goals which define our project's success.

No . 1*	**Order of completion:**	**1**
Data Preparation and Correlation Analysis. (✷)		
Subtasks		
1. Do a missing values check: ● If a record has a missing value for an attribute, remove record. 2. Do a duplicate records check: ● If two records are exactly the same, remove one. 3. To enable a meaningful correlation analysis: ● Standardise the amount and time columns. 4. Before correlation analysis and future testing. ● Balance the dataset. 5. In order to create a base set of features before selection: ● Use a correlation matrix to identify which other features are correlated to the fraud class field - these will be used in the prototype model		
Success Criteria		
I. Raw data must be free of missing values, outliers, inaccuracies and duplicate records. II. Features for the prototype model are chosen after correlation analysis. III. Dataset is balanced for future use.		
Justification		
Removing records with missing values prevents inconsistency errors during training the predictive model. If duplicates are in the dataset, redundant data may appear both in the training and testing sets and cause inaccurate learning on our model.		

Column standardization is required for a meaningful correlation comparison since correlation is scale sensitive and data columns with very similar trends are also likely to carry very similar information. Most learning algorithms fail to cope with imbalanced training datasets as they are sensitive to the proportions of the different classes.

No . 2*	**Order of completion:**	**2**
Tune parameters to maximise model accuracy for a set of given features. (✷)		
Subtasks		
1. Create the Sigmoid function: ● This generates a fraudulency probability given a set of parameterized features. 2. Create the Cost function: ● Given a dataset which has been tested (each record was predicted class) - find the sum of average errors between predicted probability and actual class. 3. Create the Partial Derivative Function: ● For a small increase in the value of one feature from a list of given features, find the rate of change of cost value (measures if accuracy improved) 4. Create the Gradient Descent Algorithm: ● Iterative algorithm which finds the minima of the Cost Function for a given each optimized feature value.		
Success Criteria		
A significant decrease should be seen after calculating the cost values for before and after tuning the parameters for a set of correlated features to the fraudulency class.		
Justification		
This indicates that the performance of the predictive model has significantly improved and is ready to be used in the testing and UI creation phase - model can be deployed.		

No. 3*	**Order of completion:**	4
Feature selection by testing and evaluation. (✷)		
Subtasks		
1. Create a Testing function. ● An inputted dataset is tested by the model (set of parameterized features) passed into the function. 2. Create an Evaluative function. ● The number of true/false positives/negatives in a tested dataset are returned. 3. Create multiple evaluation metrics. ● Test results passed in and numerical value indicating performance is returned. 4. Create the K-fold Cross Validation method. ● This method reduces bias by ensuring that every record from the original dataset will appear in the training and test sets. 5. Use an iterative algorithm to select features. ● A final feature set will be selected by picking features which when added to a model, improves the performance metrics.		
Success Criteria		
Metric evaluative functions to help evaluate performance further into the project. The best performing set of features are chosen for the model used in production.		
Justification		
The analyst will work more efficiently alongside a more skilled algorithm so more fraud will be prevented.		

No. 4*	**Order of completion:**	**3**
User interface for the application's functionality. (✷)		
Subtasks		
1. Create the structure for the web page using HTML and style it using CSS.		

2. Import the transactions into the page from the database using flask.
3. Add buttons to let the user classify transactions and make changes to the database using POST requests.
4. Add useful graphs and display performance metrics.

Success Criteria

I. The user should be able to view all transactions which have been tested by the deployed model.

II. The user should be able to reclassify any transaction in the table using buttons.

III. Changes to classifications by the user must be saved in the database.

IV. The user can use the graphics to inform their decisions in classifying a transactions fraudulency / authenticity.

V. The design of the web-app should be minimal and easily readable.

VI. The user should have a help and documentation page to guide them using the program.

Justification

Graphical user interface will allow the user to interact with the programs and make changes to records classifications using a familiar readable website instead of a complicated command line or complex database management software which would require analysts to have additional training.

Graphs are an example of visualization which is a useful computational method to help a user interpret trends in data.

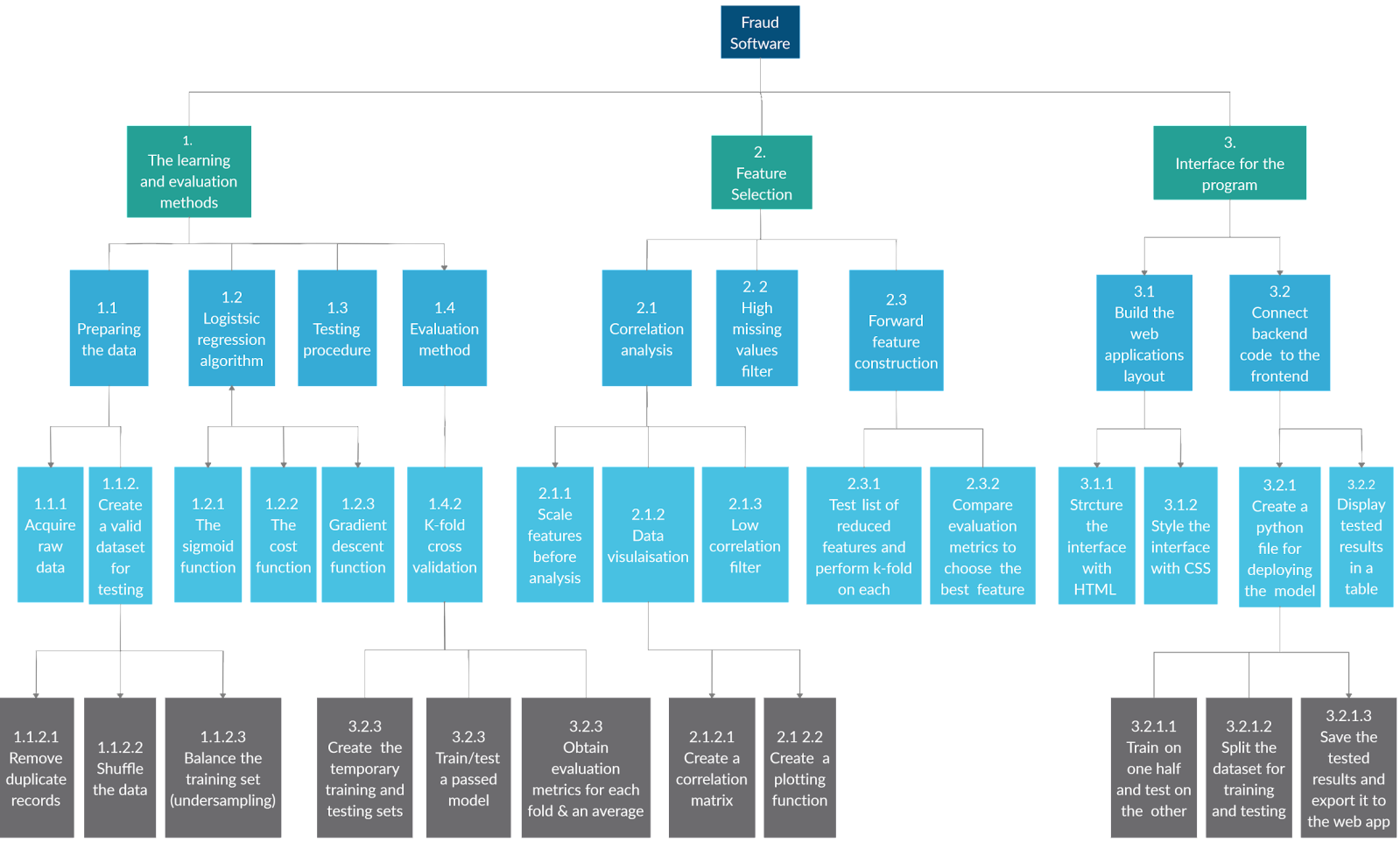
Section 2. Design

Justification for problem decomposition

The top-down strategy uses the modular approach to develop the design of a system. It is called that because it systematically breaks the highest-level problems and moves towards the lowest level modules. I will be using a top-down strategy to simplify the process of designing my project solution since that sub-problems are much easier to investigate the components of their required solution.

Systems diagram

The diagram below shows a structure chart for the breakdown of our problem. You might want to zoom in to see the text as clearly as possible.



Explanation of Modules and Algorithms

The development of our program can be done in two parts. The first part is creating an effective machine learning model for predicting fraudulent transactions (modules 1 and 2). The second part is deploying the finalised model (3) in such a way so its predictions on the testing set can be displayed through the browser.

Importance of calculating time-complexity in our project

Since we have such a large dataset 280,000 records. When we go through feature selection, we have to test every single combination, removing or adding one at a time which could lead to a factorial case of time-complexity. This could take the computer potentially days to complete the algorithm. Therefore the time-complexity subroutine that is called in the feature selection process should be known so we can preemptively plan our algorithm around its efficiency during development.

Preparing the data (Objective 1*)

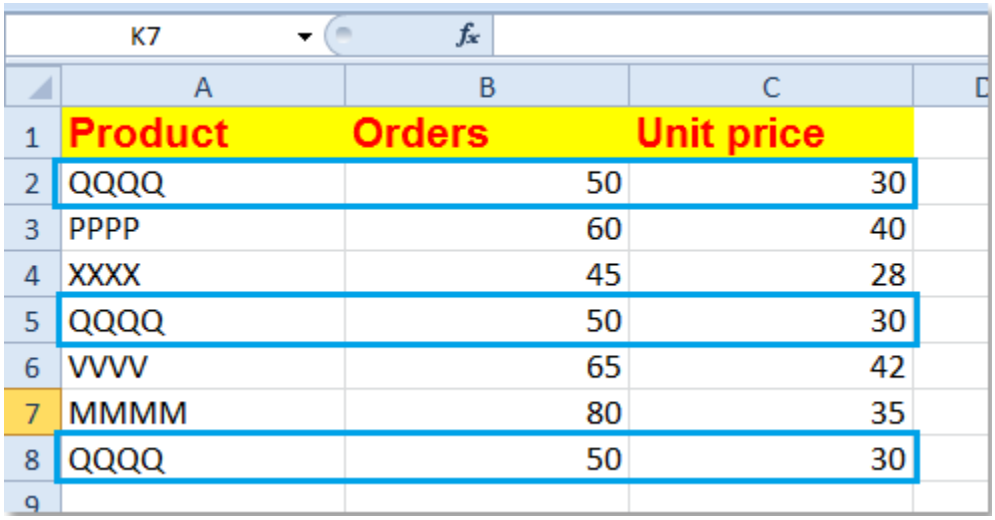
First we must obtain the raw data by importing it into our workspace. Then we must clean the data. Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset.

1.1. Remove records from the dataset with missing values

Add the downloaded database file of financial transactions into our project directory. Use a CSV reading function to input the data into our program. Scan and find records in our dataset with missing records and remove them. We can continue if there are no such records.

1.2 Remove duplicate records

Check if there are duplicates in our data in order to remove them. Initially we might not choose to delete duplicates if there are not a significant amount of them and we can't confirm that they are identical due to similar transactions being carried out twice in turn.



	A	B	C
1	Product	Orders	Unit price
2	QQQQ	50	30
3	PPPP	60	40
4	XXXX	45	28
5	QQQQ	50	30
6	VVVV	65	42
7	MMMM	80	35
8	QQQQ	50	30

1.3 Column Standardisation

Time and amount should be scaled as the other columns to help our algorithms better understand patterns that determine whether a transaction is a fraud or not. Since correlation is scale sensitive and data columns with very similar trends are also likely to carry duplicate data. Here are two common methods of standardizing:

1. $$x_{scaled} = \frac{x - mean}{sd}$$
2. $$X_{new} = \frac{X - X_{median}}{IQR}$$

I'm going to use my own implementation of the robust scaler (image 1) from pandas instead of the standard scaler (image 2) since it is affected by non-normally distributed variables which have extreme outliers like our time and amount columns.

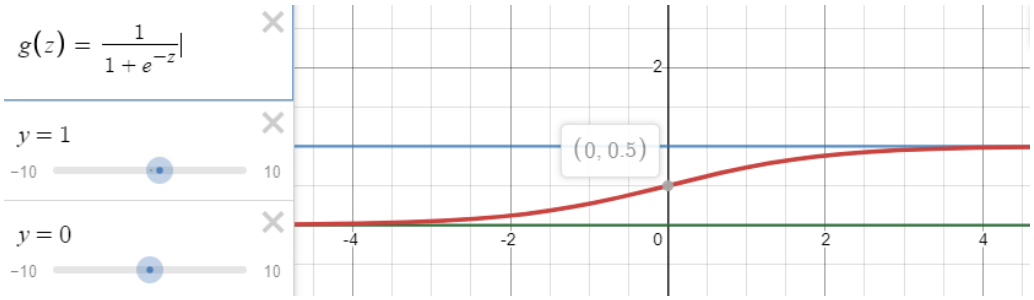
Creation of the Training Algorithm (Objective 2*)

Logistic regression is the standard method for solving binary classification problems such as mine - as fraudulency can only be measured by a true or false value, let y be the fraudulency prediction: $y \in \{0, 1\}$. Even though we first calculate the probability of a transaction being

fraudulent. The actual fraudulency is predicted as 1 (true) if the probability is equal to or more than 0.5.

2. The Sigmoid Function

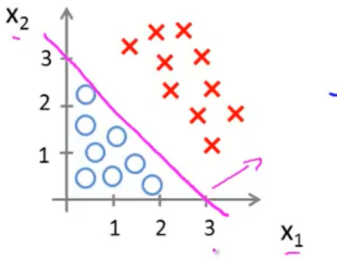
The sigmoid function $g(z) = \frac{1}{1+e^{-z}}$ calculates the probability of a transaction being predicted as fraudulent. I created the graph on Desmos as shown below.



As you can see, $g(z)$ captures the intuition of a good probability function as it has a range between 0 and 1 and it's domain, z is any real number, $z \in \mathbb{R}$ where z represents an undefined function whose input variables are to be decided on during the process of feature selection (3.2) and whose parameters are going to be iteratively optimized during gradient descent(3.3) .

However we'll demonstrate how it works using an example with two variables x_1 and x_2, in our example these could be any variables of a transaction like 'amount' and 'time'.

A scatter graph for a list of 17 already classified transactions shown as coordinates may look like this:



- Note that the fraudulent coordinates (training examples) are marked as red crosses.
- When input $z \geq 0$, $g(z) \geq 0.5$ (as $e^0 = 1\ so\ \frac{1}{1+1} = 0.5$). So when $z \geq 0$, we can predict the case as fraudulent ("y=1"), since . As discussed z is a function defined in terms of the variables x_1 and x_2 and parameterized by θ.
- Substituting z gives: $\theta_0 + \theta_1(x_1) + \theta_2(x_2) \geq 0$which can be rearranged for the equation of the pink line: $x_2 \geq \frac{-\theta_0 - \theta_1(x_1)}{\theta_2}$. The pink line is the decision boundary. If a training

example is above the straight line above , it will be predicted as fraud. Since “y=1” if $x_2 \geq -\frac{\theta_1}{\theta_2}(x_1) - \frac{\theta_0}{\theta_2}$

- For our example $z = 3 + 1(x_1) + 1(x_2)$, but the decision boundary doesn’t have to be linear. It can be any nonlinear function of the input variables.

The sigmoid function finds z by multiplying the parameters by the corresponding input features. The pseudocode for the sigmoid function is:

```
Function sigmoid(x, parameters)
    Z = 0
    For i = 0 to x.length:
        Z += x[i] * parameters[i]
    Next i
    Return Z
Endfunction
```

Sigmoid Time Complexity

```
Let n = number of parameters / features
Linear time - one loop: O(n)
```

3. The Cost Function

The cost function returns the average error between predicted outcomes compared with the actual outcomes. This is a measure of how wrong the model is in terms of its ability to make predictions. In a binary classification problem, for a given training example, there are only two cases for how we determine the performance of our model.

The cost function takes in the sigmoid function’s outputted predicted probability as an input and about the training example with

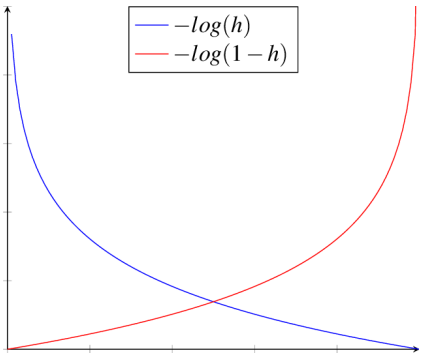
For the first training example, if it’s target class were 1 (Fraud), and our model predicted

It is mathematically defined as:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} - y^{(i)} log(h(x^{(i)}) - (1 - y^{(i)})(log(1 - h(x^{(i)})))$$

Which is more simply defined as:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} - y^{(i)} log(\hat{y}^{(i)} - (1 - y^{(i)})(log(1 - \hat{y}^{(i)}))$$



$\hat{y}^{(i)}$ is the predicted probability (a decimal in the range 0-1) whereas $y^{(i)}$ is an integer 0 or 1 which makes one of the two terms cancel based on its value.

where:

- m is the number of training example
- i is the index of the training example in the dataset
- $\hat{y}^{(i)}$ is the predicted value for the target value.

4 Gradient Descent - the parameter tuning algorithm

This is our learning algorithm. Gradient descent relies on multivariate differentiation in order to tune a model's parameters. It calculates the partial derivative of the cost function with respect to the parameters in inputs in order to find the minimum cost output - the lowest error when making predictions.

It is mathematically defined as:

Function gradientDescent:

Repeat until convergence {

$$\theta_j^{new} := \theta_j^{old} - \frac{\partial}{\partial \theta_j} J(\theta)$$

} (for j = 0, ... , n)

Return [$\theta_j \; for \; j \; = \; 0 \; to \; n$]

Endfunction

The partial derivative of $J(x^{(i)}, \theta)$ with respect to θ_j is : $\frac{1}{m}\sum_{i=1}^{m}(G(x^{(i)}, \theta) \; - y^{(i)})x_j$

PD is calculated separately and the value is assigned to a local variable and used on each iteration of the inner loop of our algorithm. The inner count-controlled loop iterates from j = 0 to j=n, where the partial derivatives are calculated to find a new value for the parameter. The outer loop is condition-controlled and only terminates when the average absolute mean value of all the partial derivatives converges to 0, which means the optimum parameter values causing the minimum cost have been found for the given features chosen.

It should be noted that even when the cost is at a minimum, this does not mean we have a good model since that is defined by the features we have selected, our main factor for performance.

Time Complexity - Partial Derivative of Cost

- Let m = number of transactions
- Let n = number of parameters / features

Iterations = m * (sigmoid(n)) = O(mn)

Time Complexity - Gradient Descent

Assume the number of iterations to train a parameter reach convergence (a) is independent of the number of the number of parameters (n) -> a is constant.

- Let a = time to train a parameter (coefficient cancels)
- Let m = number of transactions
- Let n = number of parameters / features

Number of iterations =
$a \; * \; [n \; * \; \; patialDerviative(m, \; n)] \; = \; a \; * \; [n \; * \; mn] \; = \; O(mn^2)$

Feature selection, testing and evaluation. (Objective 3*)

5.1 The Testing Routine

This method is required for the evaluation model. Both might initially be written as one function. The purpose of this function is to take in a subset of the dataset for testing. Testing refers to applying the fraud prediction model to the testing set.

This is how the model will be tested:

1. The function will take in a passed model (the names of the features and parameter values) and will proceed to start iterating through each record in the testing set.
2. On each iteration it will calculate the predicted probability of each record.
3. This will be saved into the dataframe.
4. It will then reiterate over each record in the dataframe and access the predicted probabilities and feed it into a function named decision boundary which decides if a transaction is fraudulent based of it's predicted probability.
5. The outputted binary decision will be saved to the record.

It is important to note that during the loop we assume the testingSet's records will be passed by reference so any changes made to it inside the code will modify the original copy outside of the loop. The pseudocode for the function will be as follows:

```
Function test(testingSet, featureNames, parameterValues):

    For record in testingSet:  # 1

        predictedProbability = sigmoid(record, featureNames, parameterValues)  # 2

        record.PredictedProbality = predictedProbability # 3
```

```
            predictedClass = decisionBoundary(predictedProbability, featureNames, parameterValues) #4
            record.PredictedClass = predictedClass  # 5
        Next record
Endfunction
```

Time Complexity

- Let m = number of transactions
- Let n = number of parameters / features

Iterations = $m * sigmoid(n) = O(mn)$

5.2 Evaluation Method

A false positive is when a predictive model determines something is true when it is actually false. A false negative is saying something is false when it is actually true. Positive in our case is if a transaction is fraudulent while negative is authentic.

All the evaluative method must do is compare the predicted and actual class of each record to enumerate the total number of true positives, true negatives, false positives and false negatives then return these back to the main program.

```
function evaluate(testedSet):

    # Count and class every correct and instance of a prediction
    tp = tn = fp = fn = 0 # where True Positive = tp

    for i = 0 testedSet.length:

        predictedClass = testedSet[i].predictedClass
        actualClass = testingSet[i].Class

        # Got the prediction correction
        if actualClass == predictedClass:

            if actualClass == 1: # and it was a positive correct... etc.
            tp += 1
```

```
                else:
                    tn += 1
                Endif
            else:
                if predictedClass == 1:
                    fp += 1

                else:
                    fn += 1
                Endif
            Endif

    Next i
    return [tp, tn, fp, fn]

Endfunction
```

Time Complexity

- Let m = number of transactions
- Let n = number of parameters / features

Time Complexity = O(m)

6. The choice of evaluation metrics

Before we can actually create a learning model and train it, we have to first decide how we will evaluate its success. The choice of evaluation metrics can affect how we properly judge the performance of our learning model especially for highly unbalanced datasets such as my own. For instance if we chose accuracy $\left(\frac{true\ negatives + true\ positives}{total\ predictions}\right)$ as our primary performance measure, arbitrarily predicting every case as non-fraudulent would mean the accuracy would be $\left(\frac{0 + 284{,}807}{285{,}299}=\right)$ 99.83%, as there are 492 frauds out of 284,807 transactions and 285,299 in total.

Looking back at the success criteria of a good evaluation metric: “a defined way of determining which algorithm performed the best at detecting fraud in the testing dataset”. Therefore as accuracy gave a high result without correctly predicting any cases of fraud, it is not suitable for our task.

There are two other evaluation metrics that we should consider:

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives} \qquad precision = \frac{true\ positives}{true\ positives + false\ positives}$$

Recall: ability of a classification model to identify all relevant instances.

Precision: ability of a classification model to return only relevant instances.

How do we evaluate the dataset fairly?

In machine learning, it is commonplace to divide our dataset into the data we train our learning model on and the data we must test this model on. The model should ideally have good results when evaluating both the training and testing sets. The model will mostly always perform better on the data it is trained on. However, an important requirement of my fraud detection software is that it is ready to stay on top of new trends in fraud. This means that the learning model should perform as best as possible on the unseen data in the testing set when compared to its performance on the training set.

Variance is the error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise present in the training data, rather than the intended outputs . This is an example of overfitting: “making an overly complex model to explain idiosyncrasies in the data under study”. A good model should have low variance. Before a dataset is tran/test splitted, we should shuffle the data to ensure that each data point creates an "independent" change on the model, without being biased by the same points before them.

7. K-fold cross validation

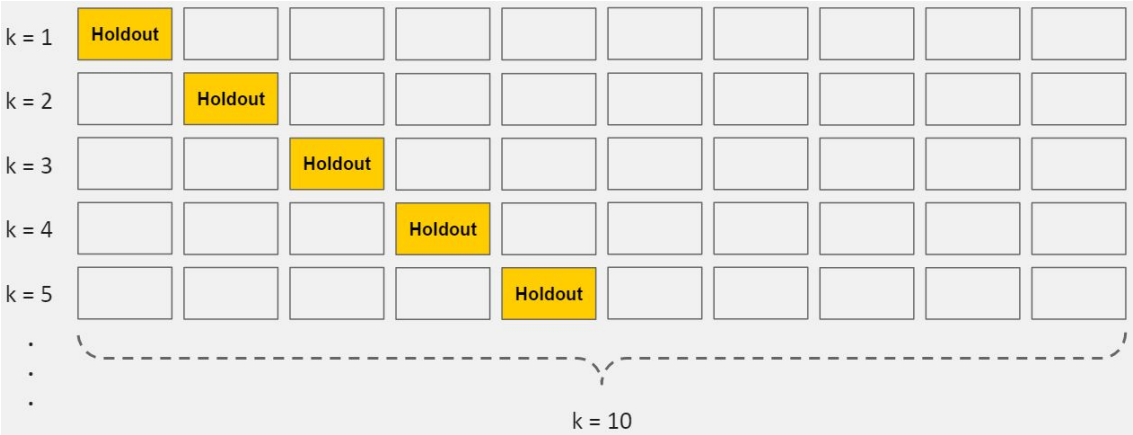
The way in which a sampling procedure splits a dataset into testing and training sets affects the performance of a model. This means that different train/splits will produce different models which vary in performance. Therefore we require a resampling procedure which creates different train/test splits and evaluates the model trained/tested produced from these different subsets.

A method of doing this is called k-fold cross validation in which every record from the original dataset will appear in the training and test set over the course of k iterations. This is a standard technique to detect overfitting as it determines the variance by comparing an average of the evaluation scores for how the model performed on the training sets and on the testing sets.

It works by splitting the dataset into k groups and for each unique group:

1. Take the group as a hold out or test data set
2. Take the remaining groups as a training data set
3. Fit a model on the training set and evaluate it on the test set
4. Retain the evaluation score and discard the model

Summarize the skill of the model using the sample of model evaluation scores




```
Function cross_validation(k,  dataset, features):

        Split dataset into k subsets of data and store in array

        Folds = divide(dataset, k)

        FoldSize = dataset.length / k

        recallVals = precisionVals = []

        For i = 0 to k - 1:

                # Take the ith subset of data to hold for testing.
                testingSet = Folds.pop(i)

                # Train the model on the other k-1 folds
                trainingSet = merge(folds)
                Parameters = gradientDescent(dataset, trainingSet, features)

                # Test and evaluate the results on the ith fold
                 testResults = evaluate( test(testingSet, features, parameter) )
```

```
            precisionVals.append(precision(testResults))
            recallVals.append(recall(testResults))

    Next i
    Return mean(precisionVals), mean(recallVals)

End Function
```

Time complexity

- Let m = number of transactions
- Let n = number of parameters / features
- Let k = number of folds

Iterations = k * (train(n, [m - (m / k)]) + test([m/k], n) + evaluate(m/k)

$= \ k * ([m \ - \ (m \ / \ k)]* \ n^2 + (m/k) \ * \ n \ + (m/k))$

$= \ k * ([m \ - \ (m \ / \ k)]* \ n^2 + (mn/k) + \ (m/k))$

$= \ k * ([mn^2 \ - \ (mn^2 \ / \ k)] + \ \ (mn/k) + \ (m/k)))$

$= \ kmn^2 \ - \ mn^2 \ + \ mn \ + \ m \ = O(\ kmn^2)$

8. Feature selection

Feature selection is the process of selecting a subset of relevant features for use in model construction to maximise the performance measures of our model.

Methods of Feature Reduction

Filter based methods such as the high correlation filter reduction are quick heuristic methods similar to that we used in module 1.5, are good for building prototype models but are best suited to when there are less fields. In our case we need a high level of evidence to show our model will be evidence. We will have to use an exhaustive search of the possibilities with some cutbacks otherwise the problem will become computationally intractable for all but the smallest of feature sets.

We will use the **Forward Feature Construction** method to achieve this which works as follows:

1. A base model (for us V0) is established and evaluated to obtain a benchmark

performance.

2. One by one features V1 - V28 were temporarily added to the model to gather whether the performance increased on their arrival.
3. After iterating over all features, the feature which caused the greatest performance gain was permanently added to the model.
4. This is repeated until no performance gain was observed.

Time complexity - unknown until implemented,

Creating the Interface (Objective 4*)

Once the model is created, we need an interface for the user to work alongside it. The user will not input training cases into the model like other machine learning projects. Instead, all transactions are stored internally and the model will output predictions on cases to the user with predicted probabilities. Then the user will have the option to overrule the model's decision.

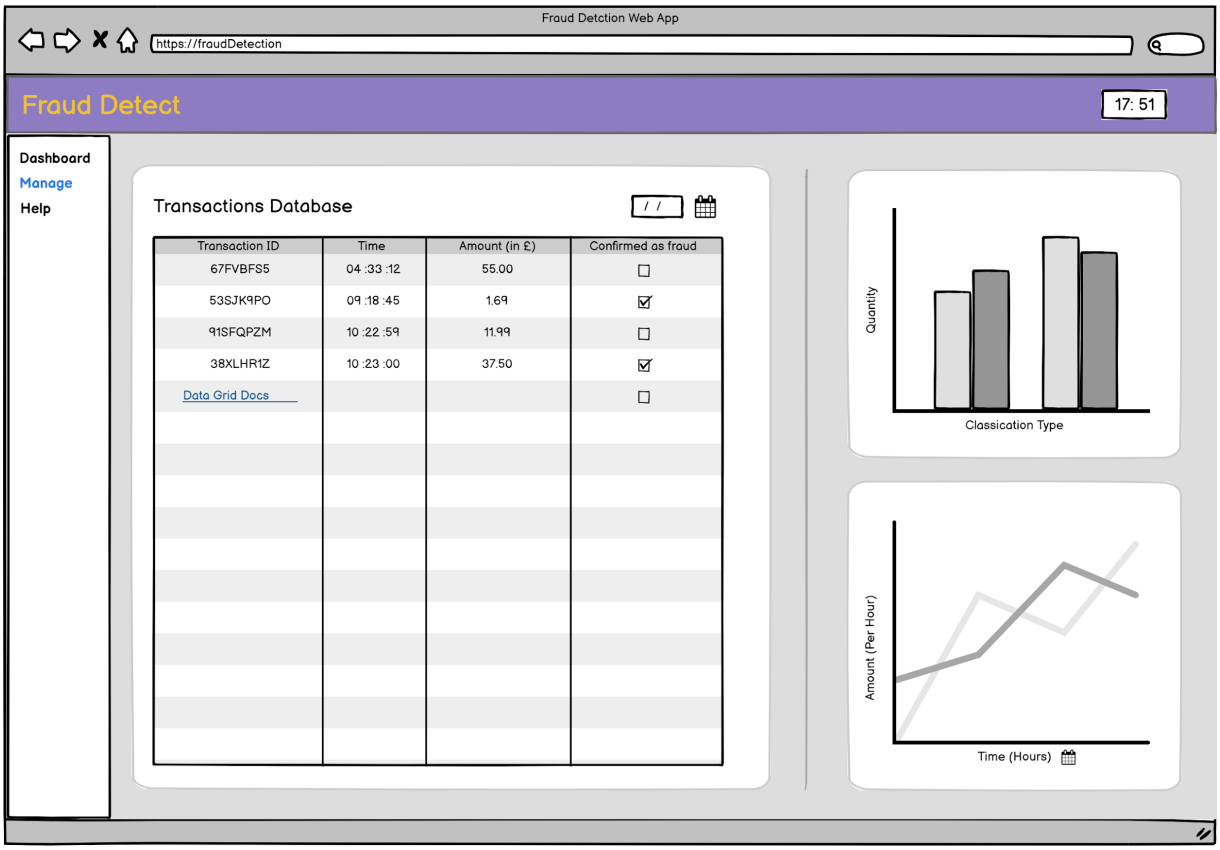
First we'll have to make the HTML templates on which the model will display its results, at first it can be rudimentary and simply lay the foundation for how the program will function. Cases close to probability 0.5 should be evaluated since it was on the decision boundary. One previously discussed problem is that we have no information that could be interpreted by a human to help them decide if they should classify the case as fraud/non-faud.

Screen design (Objective 4*)

This mockup made on Balsamiq shows our main dashboard.

Dashboard page

The purpose of the dashboard page is to display all the records of fraudulent or unclassified records which need to be dealt with by the analyst. To assist the analyst with making those decisions the dashboard page makes use of graphs so that the data can be visualised and easily interpreted. The records will be displayed in a table and each row will have a checkbox or button to classify the data as fraud or genuine.



The graph on the top right is a bar graph which compares the amount of false positives, false negatives, true positives and true negatives. This graph is due to change because the actual ratio of transactions with a negative class to a negative one will be very high. For this reason we will ask the stakeholder during the iterative development when we have launched the web app. This could have also been represented as a pie chart.

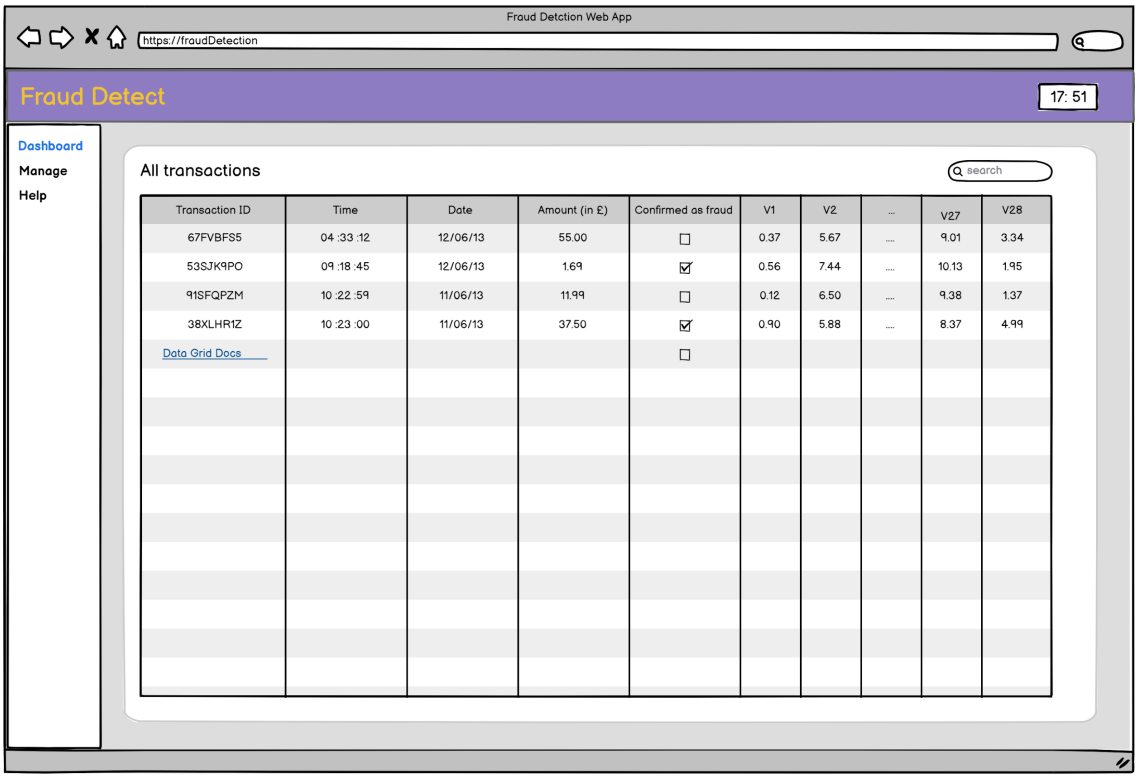
The graph on the bottom right is a time series/line graph which shows the amount of transactions happening of each class per hour. Since (fraudulency) risk due to time of day is not a physical field/attribute of the data, this graph will be a powerful data visualization technique because the analyst could utilise the graph to identify a region of time with a high amount of frauds and review records which have other attributes similar to fraudulent transactions and add them to a list to be investigated.

Since hours are in discrete intervals I could have opted to use a bar graph to represent the amount for every hour but I felt the line graph more clearly shows trends and exact turning points for whenever fraudulency peaks or troughs. We could have also chosen to create two separate graphs for both the amount of fraud and genuine transactions with respect to time but I believe this would fail to show how they compare quantitatively, this is why I superposed the lines onto one single graph.

The left side panel will act as a navigation bar for our website. The blue text colour indicates that this is the active page. This last feature is common to all pages.

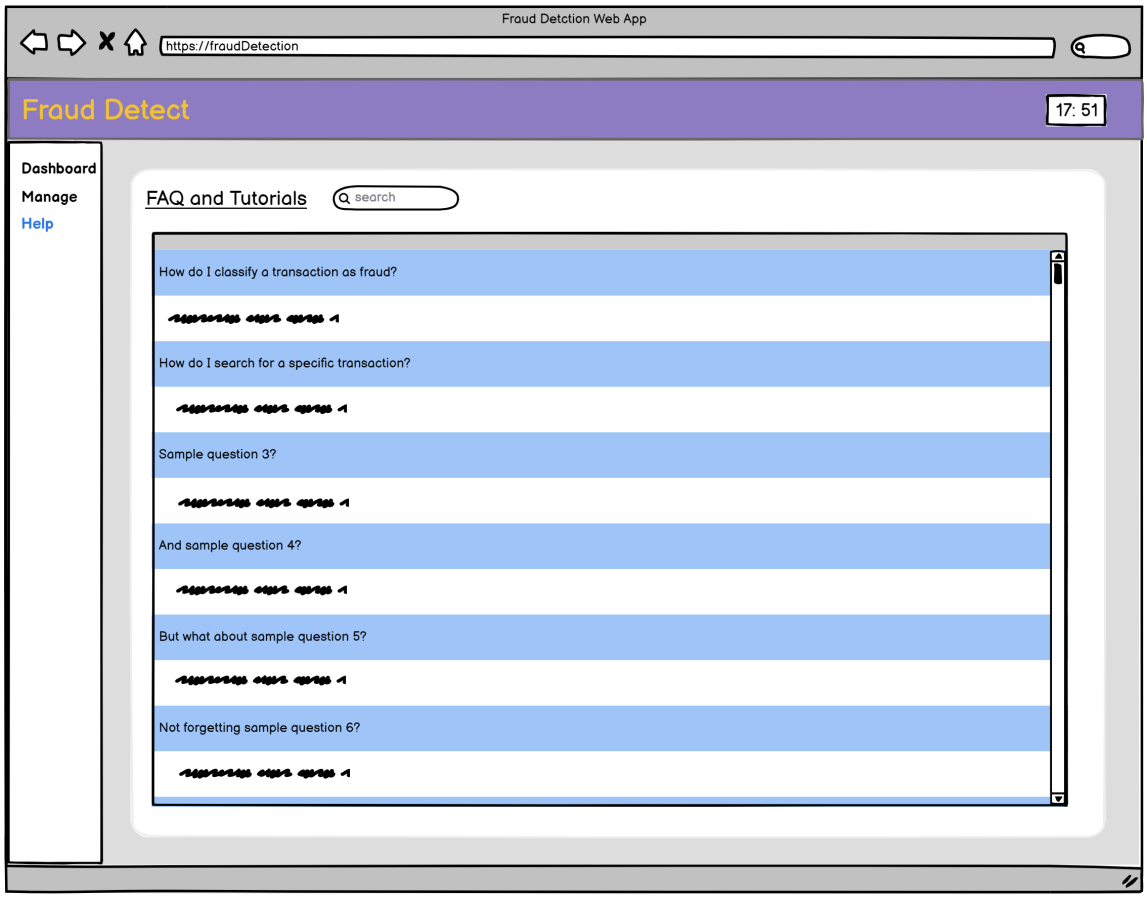
Manage page

The purpose of the manage page is to allow the analyst to browse and search through the entire database of transactions and review their classification by the prediction algorithm. It will have a column for every feature so that this data can be analysed. It will also have some sort of filtering option or search bar so that transactions that meet a certain criteria can be displayed while the other irrelevant ones are hidden.



Help page

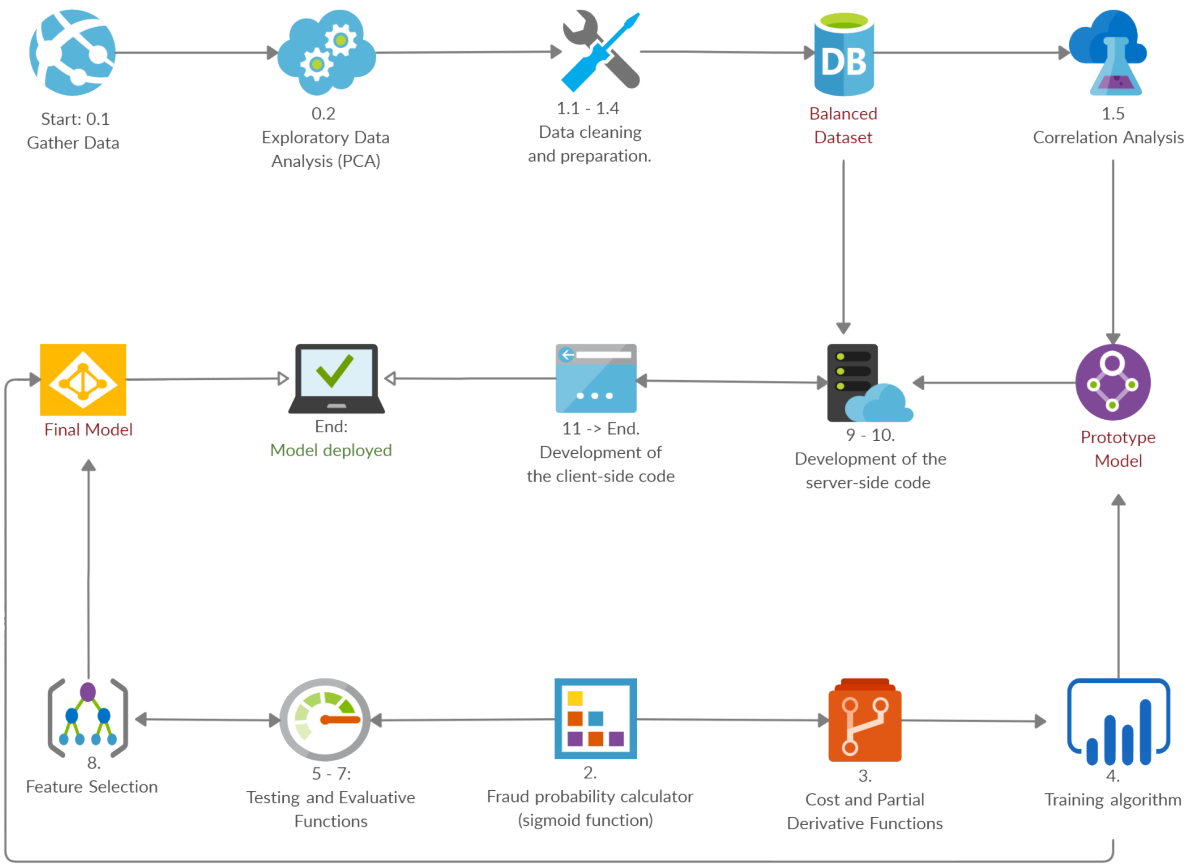
The purpose of the Help and FAQ page is to assist the user in understanding how to use the program and address their frequently asked questions. It will consist of a searchable list of common questions with their respective answers below.



I used the color light blue to highlight questions alternating with white for answers. This contrast should improve readability for users.

Forming the complete solution (subroutines diagram)

The diagram I made on Creately below outlines how the modules work together and make use of the required data. The number on each white box symbol is the module number and are later referred to in the post development test table.



The need for a prototype model

During the creation of the testing and evaluative functions we need a valid model to test and debug them. In addition, we will most likely be developing the feature selection after web-app development since it is of less importance and has huge time taking potential due to its brute force approach.

Test Strategy

Testing is a fundamental part of developing an application. Since I am going to be using an iterative development method of producing this program, I am going to continuously test the functionality of each subroutine after it is created or significant changes are made to it. I will mostly be performing these 4 standard types of tests:

1. Invalid or erroneous data test - testing data that the program cannot process and should not accept.

2. Absent data test - no data is passed into the subroutine.
3. Boundary data test - extreme or strange values which should still be accepted are passed into the subroutine.
4. Valid data test - sensible, possible data that the program should accept and be capable of processing

Tests for post development

Our program doesn't naturally lend itself to invalid testing since user input by the means of the keyboard will be rarely taken in.

No	Type	What is being tested?	Input / Action Taken	Expected Outcome
Preparing the data				
1.1	Valid	Any duplicate records present in the dataset are removed.	Dataset of 280,000 transactions searched.	No remaining duplicate records left in the dataset.
1.2	Valid	Any records with missing fields are removed.	Dataset of 280,000 transactions searched.	No records with missing values left in the dataset.
1.3	Valid	Amount and time columns are standardised in preparation for correlation analysis.	Amount and time elapsed columns of the 280,000 transactions.	Histogram after standardisation fattens.
1.4	Valid	Dataset is balanced for testing and iterative development.	Taken from the total 280,000 transactions	Equal number of each of the classifications.
1.5	Valid	Correlation matrix is produced and the feature most correlated to the fraud class becomes the prototype model.	Reduced portion of the transactions dataset analysed.	A color mapped matrix to visualise highest levels of correlation.
Sigmoid Function				

2.1	Valid	Calculating a probability for typical z value entered.	Z = Two lists whose element wise product is between -1000 and 1000.	A decimal between 0 and 1 is returned.
2.2	Boundary	Calculating a probability for a large positive z value entered	Two lists whose element wise product is more than 10^3	1.0 returned
2.3	Boundary	Calculating a probability for a large negative z value entered	Two lists whose element wise product is less than -10^3	0.0 returned.
Cost Function & Partial Derivative Functions				
3.1	Valid	Cost before parameters are tuned on a preselected set of features.	Dataset of over 500 records, small set of features and random / null value parameters.	High positive cost value.
3.2	Valid	The partial derivative cost function is the same value as one calculated using the first principles formula.	Comparing two values. The two cost values we are finding the difference between should have a very small change in one of the parameter values,	The two numbers are approximately the same with little to no error.
Gradient Descent Procedure				
4	Valid	Cost value significantly decreases after tuning the parameters.	The dataset to train the model on and list of features to train parameters for are passed into the subroutine.	A list tuned parameters is returned. Difference between two cost values is large.
Testing & Evaluative Functions				
5.1	Valid	Predict a class for every record in a given dataset.	The dataset to be tested is passed into the subroutine.	Returns the tested dataset with no errors..
5.2	Valid	Get a count of the	The tested dataset is	Returns a tuple

		number of instances of false, true positives and negatives in the tested records.	passed into the subroutine.	containing four values: number of true, false positive negatives.
Recall / Precision				
6.1	Valid	The recall metric is successfully produced.	Valid tuple of 4 positive integers entered as results.	Return the metric as a decimal between 0-1. (inclusive).
6.2	Valid	The precision metric is successfully produced.	Valid tuple of 4 positive integers.	Return the metric as a decimal between 0-1. (inclusive).
6.3	Boundary	Exception is made when a valid tuple causes a division by zero error.	Valid tuple of four integers but true positives and false negatives are equal to zero.	Print a message that no true positives and false negatives present so cannot calculate.
6.4	Boundary	Exception is made when a valid tuple causes a division by zero error.	Valid tuple of four integers but true positives and false positives are equal to zero.	Print a message that no true positives and false positives present so cannot calculate.
K-Fold Cross-Validation				
7	Valid	A model is tested on three folds of a dataset and the evaluation metrics are reported back.	Correct parameters passed. K is set to three during cross-validation.	Two valid average precision and recall values returned.
Forward Feature Selection				
8	Valid	Features are selected based on their performance when added to a predictive model.	The dataset to select features for is passed into the subroutine.	A list of the names of the selected features is returned.
Launching and Loading the Database				
9.1	Valid	Opening the page for the first time.	N/A	The page should show all transactions in a

				table.
9.2	Valid	Page is refreshed.	Mouse click on browser's refresh button.	The web page is refreshed with all the stored data loaded back.
9.3	Valid	Page is closed and reopened.	Mouse clicks on the close tab button and then re-enters the URL of the web-app.	The web page is reloaded with all the stored data loaded back.
Classifying Records				
10.1	Valid	Cursor clicks on classify as non-fraud (0).	Mouse click on button.	Transaction should be permanently classified as non-fraud.
10.2	Valid	Cursor clicks on classify as fraud (1)	Mouse click on the classify button.	Transaction should be permanently classified as fraud.
10.3	Valid	Cursor clicks on classify as undecided (NaN)	Mouse click on button.	Transaction should be permanently classified as undecided.

Data Requirement

The dataset we will be making predictions on and training our learning algorithm on is called creditcard.csv. It consists of 284,807 transactions stored as records in a comma separated value file. This database is from a European bank in September 2013 over two days which we will assume were the 1st and 2nd of September. Each record has 31 fields, 28 of which are anonymous characteristics and the other 3 is the time it took place, the amount being transacted and the class (whether they are fraudulent or genuine). The time is the time elapsed in seconds since the beginning of the dataset. The currency of the amount of each record is not disclosed but since the bank is located in Europe we will assume that it is euros.

A data frame is an abstract data type which represents a database and is provided by a library called Pandas. The library includes hundreds of methods which help us to operate on the data. This will simplify the development phase of our program since we can focus on the machine learning aspect and web app interface without wasting time creating functions to manipulate data. We will be using data frames throughout the duration of our program to store subsets of the large dataset for training and testing purposes. For example, the training subset will be used every time we choose new features and adjust the weights to maximise the performance

and the testing set will be used to evaluate the efficiency of our prediction model throughout its development and evaluation.

We will also use the dataframe during correlation analysis to create the correlation matrix, a color coded matrix where each cell value represents the values of the features in the column and row of the features. In addition, a list will also be used to store a target variable's correlation to all the other variables. A list is an appropriate data structure as it can be sorted to have the highest correlated variables values at the front so it can be traversed linearly to select the best correlated values in the least amount of time. Lists will also be used in our cross-validation function to store the different folds of data to be tested.

Key Variables

Some of these variables will not be explicitly declared but instead they represent values which will be returned from a function.

Name	Data Type	Purpose
Project Folder Directory		
creditcard	.csv	Store all the transaction records in a file. Will be loaded at the start of each session to create the
decisionDB	.csv	
Global Scope		
transactions	dataframe	To store all transaction records. The data read from the creditcard CSV file is assigned to this.
dataset	dataframe	To store all the records in the balanced dataset we will be developing our model on.
Sigmoid		
features	list	List of strings. Each of which is the name of the column of the feature we are indexing to obtain their value for the use in their probability calculation.
parameters	list	List of floating point numbers. Each of which is the value for the weight which parametrizes the feature whose index is held at the same position in the feature list.
z	float	A local variable which represents the sum of the feature values parameterised by the value of the weights. This will be the input to

		the mathematical definition of the sigmoid probability function.
Cost Function		
costSum	decimal	Used to sum the cost added due to the average error of each mispredicted transaction.
Cost Partial Derivative		
jthFeature	string	Name of the feature for which the partial derivative must be calculated and is present in the list of features.
pdVal	decimal	Final value of calculated partial derivative for a given feature
Evaluative Function		
Tp, tp, fp, fn	integer	The four counter variables are packed into a tuple to be returned as the results from the function.
Evaluation Metrics (Precision/Recall)		
testResults	tuple	Passed as parameter to function. Immutable data structure to store 4 integer counter variables
K-fold cross validation		
k	integer	Store the number of segments the data needs to be split up into

Usability features

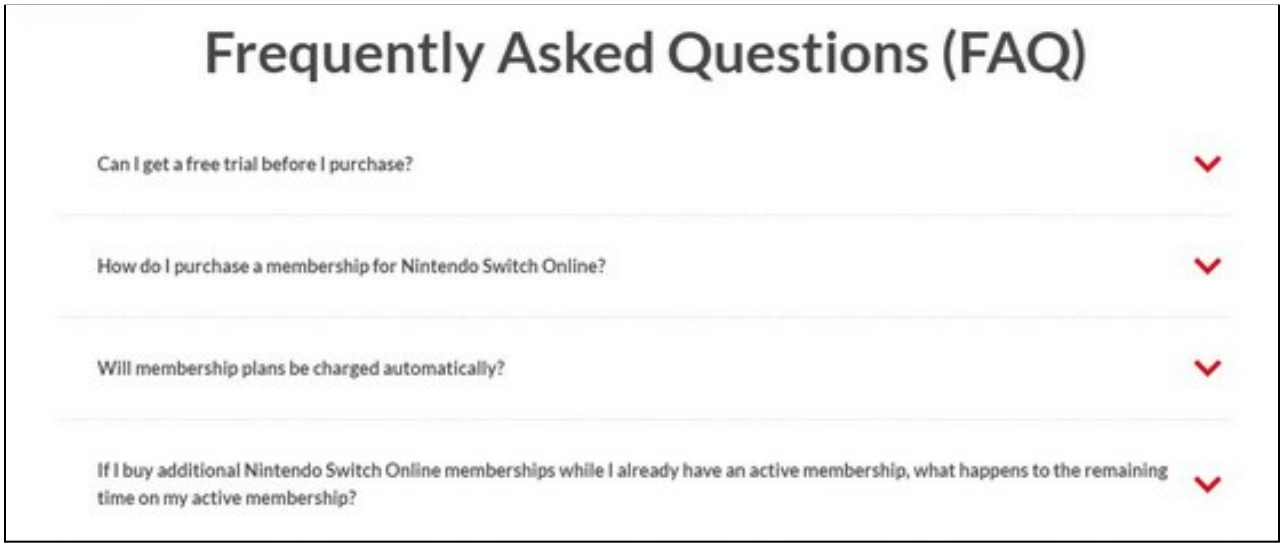
Usability features are a measure of how well a user can use a product or service to achieve a defined goal efficiently and with satisfaction. There are certain features needed to ensure that the product is usable. I've divided the usability features into two categories: standard features that are an unspoken expectation of a stakeholder and special features that we're adding to improve usability for specifically our program catered to our stakeholder's needs.

Error handling and validation

The program must have some sort of method to prevent errors breaking the program. We address this through error handling in our code. Another way of preventing user errors is through warning messages. I will also make use of form validation when we filter to ensure that incorrect queries don't cause errors or invalid responses from the server. This will also reduce the load on the server improving the user experience.

Error handling and validation

Help and documentation is the second most important usability feature. This will be provided on our web app on a separate page. It will run through how the system works and explain how to complete each task. For example, how to report a case of fraud can be shown through images on the screen, alongside a description of how to complete the task. The image below shows an example FAQ page similar to the one we desire.

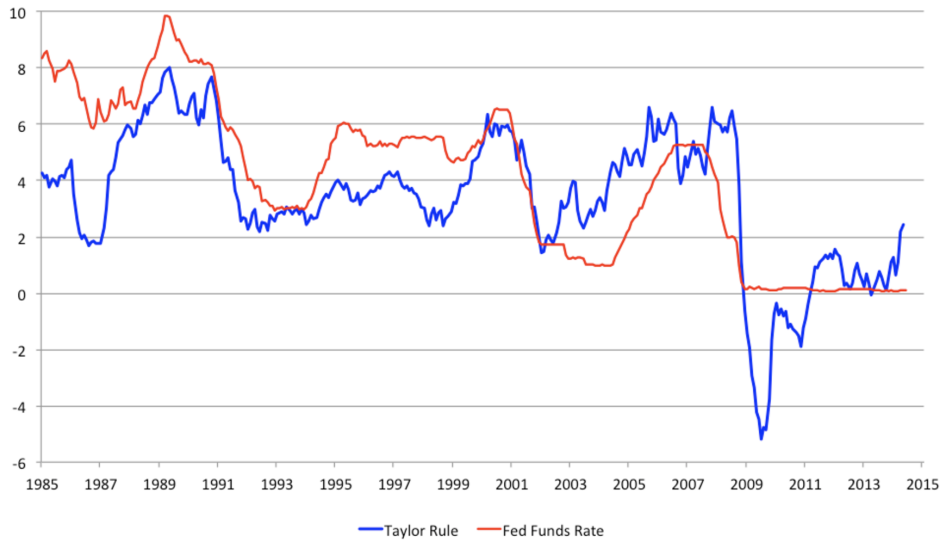


Data Visualization

As shown in the screen design, multiple graphs will be used to assist the user in identifying trends in fraudulency and spotting outliers. I will decide with the stakeholder later on into the project which graphs will be most necessary.

Colouring consistency and connotations

The example image below shows how a nameless quantity fluctuates with respect to time.



In my graphs, I will use orange or red for the line which shows fraudulency since these colours connote danger. I will use blue for genuine transactions to indicate . As you can see the deep contrast between red and blue improves its readability. This is because they are opposites on the colour wheel, and also in the emotions and feelings that they represent.

Another important feature is having consistency and standards in the design of the program. For example, the colour coding for all graphs must be the same in order to prevent mistakes.

Buttons and interactive elements

I will make use of buttons to categorize transactions as fraudulent, genuine or unclassified. Buttons are large and are a familiar element of websites to users. They will have an outline to stand out to users and indicate they are meant to be interacted with.

Section 3. Developing the Coded Solution

Chapter 0: Planning the iterative development

I chose Jupyter notebook as the Python IDE I wanted to use. Jupyter allows you to run code in cells. This is an example of stepping and is a useful feature of an IDE. Debugging is simple as code can be independently tested. Before I created a function I would use a markdown cell, essentially a comment, to plan out the function I was about to write.

In the first iteration of our development, all code will be written into one file and later destructured later into separate modules.

Chapter 1: Preparing the data - (Objective 1*)

The first step was to import all the required modules for the project. Pyplot will be used to graph to graph relationships between variables of the fraud dataset

Next I had to import the data and detect if there were any missing values in the data. I decided to use Pandas' own method to check for missing values as this is a unimportant component in the logic of my overall project.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import math, statistics, random
```

Preparing the data ¶

- First lets import it.
- Then lets clean the data by removing rows which contain missing values

In [2]:
```
# Read the CSV
transactions = pd.read_csv("creditcard.csv")

# the isnull method detects missing values for an array-like object.
print("Any missing values?", transactions.isnull().values.any())
```

```
Any missing values? False
```

It concluded that there were no missing values meaning we could continue on. So the next step was to check if there were a significant amount of duplicate rows, for which I used another Pandas inbuilt method.

Out of the 500 or so fraudulent transactions, it detected 19 duplicate rows which is not a large amount and seemed to look as if they occurred sequentially in time inferring the same client did the two transactions on purpose. So as they are legitimate we can move on. We may come back in the second iteration of development and test if removing these rows positively affects the performance of our model.

- As there was no missing values, we can go on to remove any duplicate records:
- duplicated() is an inbuilt method that finds duplicate rows based on all columns or some specific columns andreturns a Boolean Series with True value for each duplicated row.

In [3]:
```
duplicates = transactions[transactions.duplicated()]
duplicates[duplicates.Class == 1]
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
102442	68207.0	-13.192671	12.785971	-9.906650	3.320337	-4.801176	5.760059	-18.750889	-37.353443	-0.391540	...	2
102443	68207.0	-13.192671	12.785971	-9.906650	3.320337	-4.801176	5.760059	-18.750889	-37.353443	-0.391540	...	2
102444	68207.0	-13.192671	12.785971	-9.906650	3.320337	-4.801176	5.760059	-18.750889	-37.353443	-0.391540	...	2
102445	68207.0	-13.192671	12.785971	-9.906650	3.320337	-4.801176	5.760059	-18.750889	-37.353443	-0.391540	...	2
102446	68207.0	-13.192671	12.785971	-9.906650	3.320337	-4.801176	5.760059	-18.750889	-37.353443	-0.391540	...	2
141258	84204.0	-0.937843	3.462889	-6.445104	4.932199	-2.233983	-2.291561	-5.695594	1.338825	-4.322377	...	
141260	84204.0	-1.927453	1.827621	-7.019495	5.348303	-2.739188	-2.107219	-5.015848	1.205868	-4.382713	...	
143334	85285.0	-7.030308	3.421991	-9.525072	5.270891	-4.024630	-2.865682	-6.989195	3.791551	-4.622730	...	
143336	85285.0	-6.713407	3.921104	-9.746678	5.148263	-5.151563	-2.099389	-5.937767	3.578780	-4.684952	...	
150661	93853.0	-6.185857	7.102985	-13.030455	8.010823	-7.885237	-3.974550	-12.229608	4.971232	-4.248307	...	
150663	93853.0	-5.839192	7.151532	-12.816760	7.031115	-9.651272	-2.938427	-11.543207	4.843627	-3.494276	...	
150667	93860.0	-10.850282	6.727466	-16.760583	8.425832	-10.252697	-4.192171	-14.077086	7.168288	-3.683242	...	
150669	93860.0	-10.632375	7.251936	-17.681072	8.204144	-10.166591	-4.510344	-12.981606	6.783589	-4.659330	...	
150678	93879.0	-13.086519	7.352148	-18.256576	10.648505	-11.731476	-3.659167	-14.873658	8.810473	-5.418204	...	
150680	93879.0	-12.833631	7.508790	-20.491952	7.465780	-11.575304	-5.140999	-14.020564	8.332120	-4.337713	...	
151007	94362.0	-26.457745	16.497472	-30.177317	8.904157	-17.892600	-1.227904	-31.197329	-11.438920	-9.462573	...	-8
151008	94362.0	-26.457745	16.497472	-30.177317	8.904157	-17.892600	-1.227904	-31.197329	-11.438920	-9.462573	...	-8
151009	94362.0	-26.457745	16.497472	-30.177317	8.904157	-17.892600	-1.227904	-31.197329	-11.438920	-9.462573	...	-8
234633	148053.0	1.261324	2.726800	-5.435019	5.342759	1.447043	-1.442584	-0.898702	0.123062	-2.748496	...	

19 rows × 31 columns

Chapter 2 : Balancing the dataset - (Objective 1*)

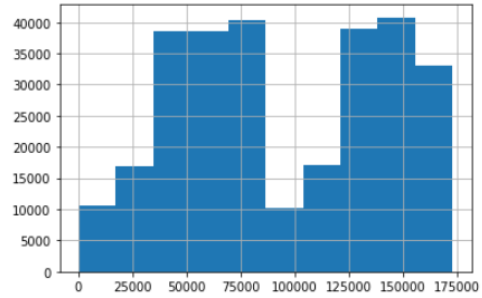
We need to perform correlation analysis in order to select any features. However we can't analyse the correlation unless it's a smaller balanced dataset. Therefore we need to create roughly the same amount of fraud cases as non-fraud cases.

```
# Separating the portion of records that were fraulent
fraud = transactions[transactions.Class == 1]

# Get the amount of fraudulent records
fraudCount = len(fraud)

# Take the random sample of non-fraudulent records
non_fraud = transactions[transactions.Class == 0].sample(fraudCount)

frames = [fraud, non_fraud]
dataset = pd.concat(frames).sample(frac=1).reset_index(drop=True) # shuffle again

# drop unscaled Amount and Time for correlation analysis
dataset = dataset.drop(['Time', 'Amount'], axis = 1)
```

Chapter 3: Column Standardisation (Objective 1*)

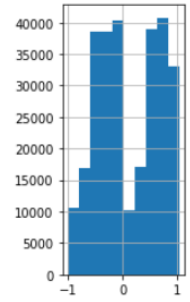
Data columns with very similar trends are also likely to carry very similar information. Correlation is scale sensitive; therefore column standardization is required for a meaningful correlation comparison.

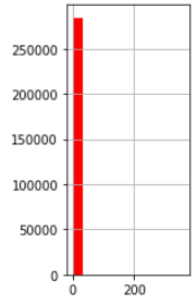
```
# Lets see the distribution for time and amount before
transactions['Time'].hist()

plt.show()
```




```
def standardiseColumn(columnName: str, trainingSet: pd.DataFrame):

    # get the lower and upper quartiles and calculate the interquartile range
    LQ, UQ = trainingSet[columnName].quantile([0.25, 0.75])
    IQR = UQ - LQ

    # and then get median of the column
    median = trainingSet[columnName].median()

    # get all the column values - which are stored in the float64 type
    vals = trainingSet[columnName].values

    # standardise every value in the column:
    scaledVals = list(map(lambda vals: (float(vals) - median) / IQR, vals))

    # convert the scaled array into the dataFrame
    trainingSet[columnName + '_Scaled'] = pd.DataFrame(scaledVals)
```

```
standardiseColumn('Amount', transactions)
standardiseColumn('Time', transactions)

# Now lets see how the distribution for time has changed

plt.subplot(1, 3, 1)
transactions['Time_Scaled'].hist()

plt.subplot(1, 3, 3)
transactions['Amount_Scaled'].hist(color='r')

plt.show()
```





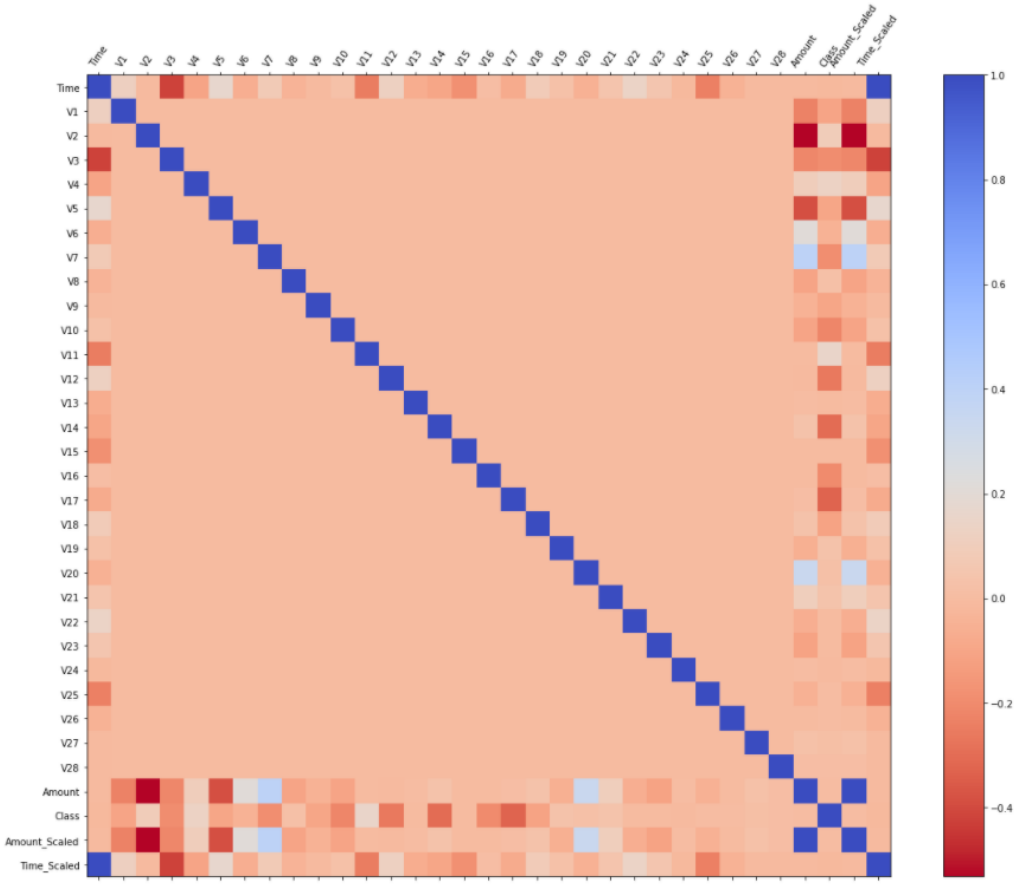

```
# standardise every value for correlation analysis

standardiseColumn('Amount', transactions)
standardiseColumn('Time', transactions)
```

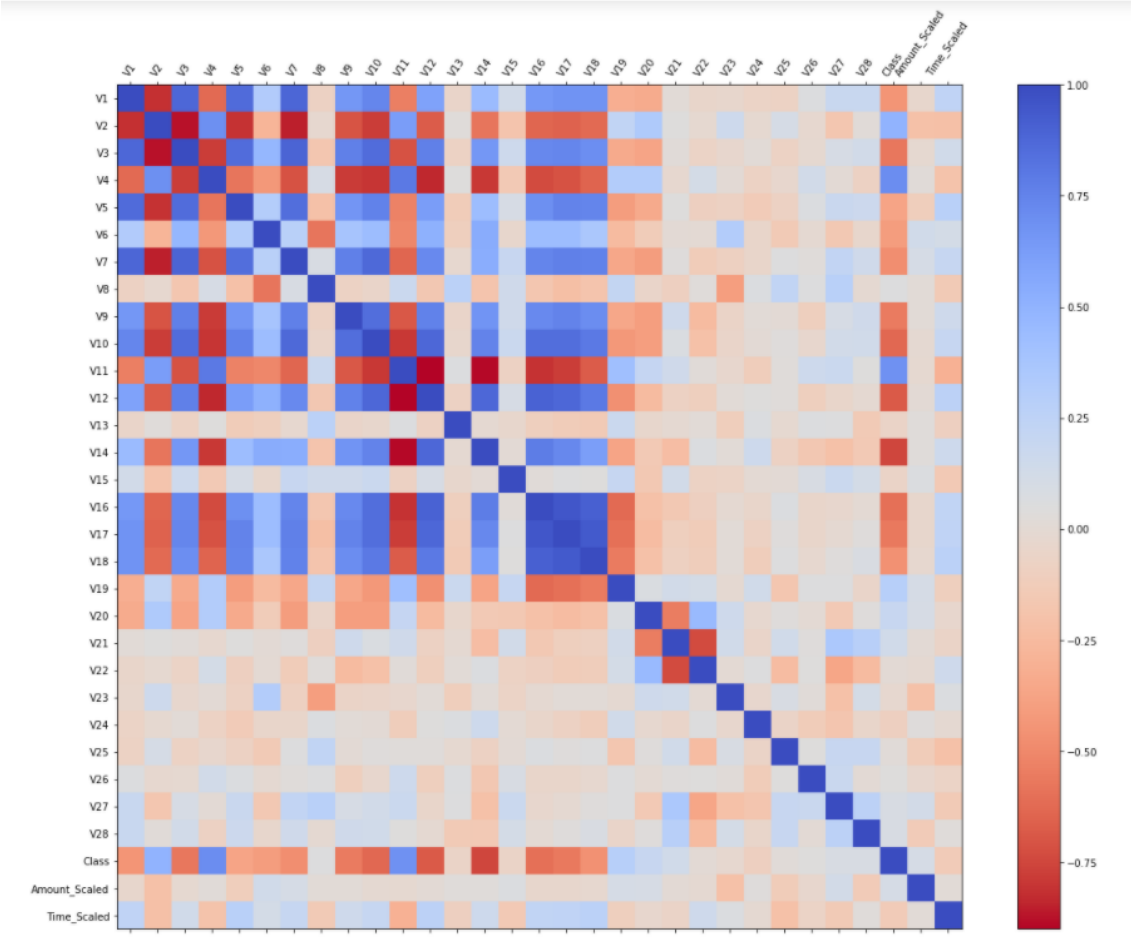
Chapter 4: Correlation analysis and data visualization (Objective 1*)

Below is the code which creates this coloured matrix.

```
def correlationMatrix(dataset, columnsToDrop):

    # create a copy of dataset for analysis
    analysisDataset = dataset.copy().drop(columnsToDrop, axis=1)

    # create a large display
    fig = plt.figure(figsize=(20, 15))
    plt.matshow(analysisDataset.corr(), cmap='coolwarm_r', fignum=fig.number)

    # number of features
    featuresCount = analysisDataset.shape[1]

    plt.xticks(range(featuresCount), analysisDataset.columns, rotation="55")
    plt.yticks(range(featuresCount), analysisDataset.columns)

    plt.colorbar()
    plt.show()
```

Here is the matrix for the entire large unbalanced dataset:

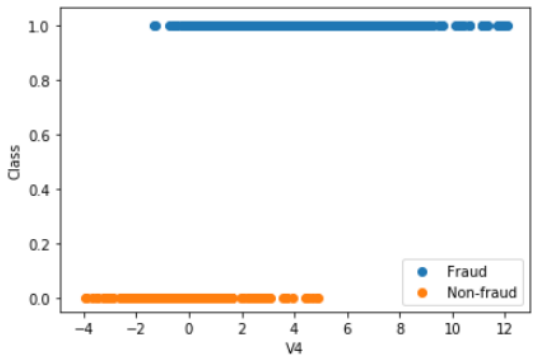




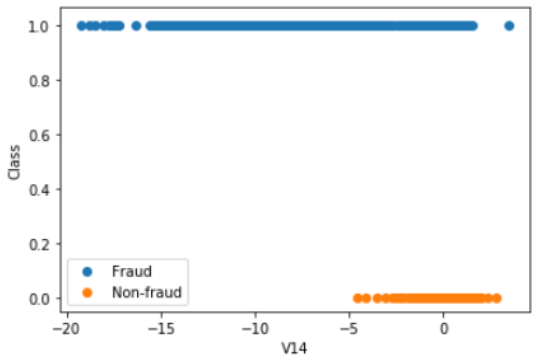
The correlation was weak before reducing and balancing the dataset

```
# Plotting the data - .plot(...) for line & .scatter(...) for scatter

def plotFeatures(trainingSet, feature1, feature2):

    fraud = trainingSet[trainingSet.Class == 1]
    non_fraud = trainingSet[trainingSet.Class == 0]

    # Input features to plot
    plt.scatter(fraud[feature1], fraud[feature2])
    plt.scatter(non_fraud[feature1], non_fraud[feature2])

    # Label the axis
    plt.xlabel(feature1)
    plt.ylabel(feature2)

    # Create legend
    plt.legend(["Fraud", "Non-fraud"])

    # Display
    plt.show()
```

```
# Seems to be strong negative correlation with V14 and strong positive with V4 so lets plot these
plotFeatures(dataset, 'V4', 'Class')
```



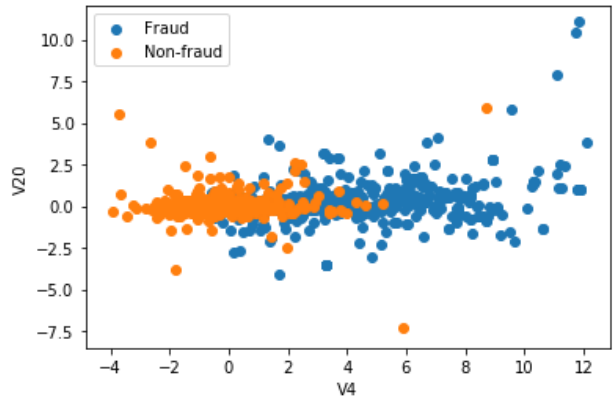

```
plotFeatures(dataset, 'V14', 'Class')
```



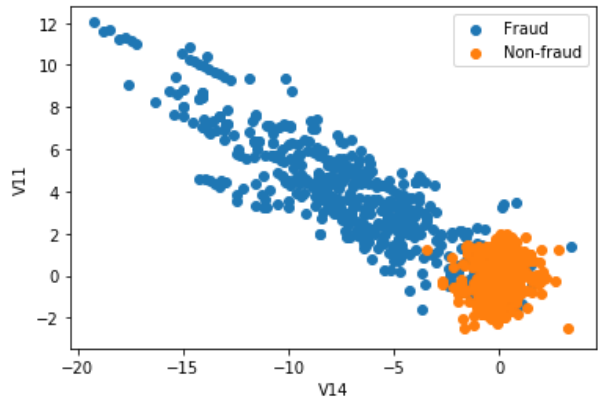
Lets see if fraud and non-fraud cluster by other two variables

- Lets start randomly and then find a clustering algorithm: K-nearest-neighbours is probably suitable.

```
plotFeatures(dataset, 'V4', 'V20')
```




```
plotFeatures(dataset, 'V14', 'V11')
```



Chapter 5: The fraud decision routine - (Objective 2*)

We decide if a transaction is fraudulent if the probability is more than 50%. This can be achieved through a simple selection statement.

```
def decision_boundary(probability):
    if probability > 0.5:
        return 1
    else:
        return 0
```

This probability is calculated through the sigmoid function, whose formula has been previously stated in design (module 3.1).

```
def sigmoid(z):
    return 1 / (1 + exp(-z))
```

Our input z will be formulated later when the blueprint of our model is defined.

Chapter 6: Defining the model

We are going to define a model as a list of features. Each feature will be a string (i.e. 'V2') which is used as a key to access the value of that feature in that row. In the final trained model, each feature will be accompanied by a corresponding parameter value. Parameter values are denoted by theta. These values are tuned by a training algorithm. For now we will use a random number generator function to create testing values

Temporarily we will create a new feature for our dataset called 'V0' which we're declaring as our 0th feature. It will universally have a value of 1 and will act as a constant term for our parameters. This feature is the fundamental model we will be adding features to. This is because it is required for all models yet it's value is indifferent between models.

```
# Read the CSV
transactions = pd.read_csv("creditcard.csv")
transactions["V0"] = 1
```

I've added an assignment statement which creates a new column for this 0th and sets the value of this field as 1 for all records in the transactions dataframe. This is the second line of code in the above diagram, a block of code created in the first chapter. This means we can define a valid model and generate some random parameter values since now 'V0' exists.

```
# Create a training set one fifth the size of our dataset
trainingSet = dataset.sample(len(dataset) // 5)

#  Lets arbitrarily choose some features
testFeatures = ['V0' ,'V16', 'V17'] # - v0 is always mandatory

# Generate some random parameter values for testing
testParameters = [random.randint(-2,2) for x in testFeatures]
print(testParameters)
```

```
[2, -1, 1]
```

Chapter 7: The fraud probability (sigmoid) function (Objective 2*)

Let's proceed to create the code which multiplies all feature values by their parameter values and add each to z. We are going to use a for loop to achieve this. It will access the parameter and feature value by indexing their lists with an index j. This will repeat over n iterations, the number of features in the list.

Finally this j input value will be substituted into the sigmoid function and the output value will be returned.

```
def sigmoid(trainingExample, features: list, parameters: list) -> float:
    """
    j is the index of the feature out of our selected feature set.
    n is the number of selected features
    """
    z = 0 # our input to the sigmoid (probability) function
    n = len(features)
    for j in range(n):
        featureName = features[j]
        jthFeatureVal, jthParameterVal = trainingExample[featureName], parameters[j]
        z += jthFeatureVal * jthParameterVal
    return 1 / (1 + math.exp(-z))
```

Chapter 8: Cost Algorithm 03/11/20 (Objective 2*)

This function should follow the mathematical definition of our cost function in the design section.

```
def cost(trainingSet, features: list, parameters: list) -> float:
    """
    m is the total number of training examples
    """
    costSum = 0
    # iterate over dataframe by row
    for index, trainingExample in trainingSet.iterrows():
        probability = sigmoid(trainingExample, features, parameters) - 0.000001
        # the piecewise implemation
        if trainingExample.Class == 1:
            costSum += -math.log(probability)
        else:
            costSum += -math.log(1 - probability)
    # Makes an average
    m = trainingSet.shape[0]
    return costSum / m
```

Chapter 9: Cost Partial Derivative 03/11/20 (Objective 2*)

```
def costPartialDerivative(trainingSet, jthFeature: str, features: list, parameters: list):
    """
    'jthFeature' is the name of the feature we're partially differentiating the cost function with resp
    'm' is the total number of training examples
    """
    m = trainingSet.shape[0]
    pdSum = 0
    for index, trainingExample in trainingSet.iterrows():
        probability = sigmoid(trainingExample, features, parameters)
        jthFeatureVal = trainingExample[jthFeature]
        pdSum += (probability - trainingExample["Class"]) * jthFeatureVal
    return pdSum / m
```

Testing to inform evaluation

Our first test will be to ensure that the costPartialDerivative outputs a sensible float value for all 3 of our features.

```
print(
    costPartialDerivative(trainingSet, 'V0', testFeatures, testParameters),
    costPartialDerivative(trainingSet, 'V16', testFeatures, testParameters),
    costPartialDerivative(trainingSet, 'V17', testFeatures, testParameters)
)
```

```
0.15408147037315081 1.5902151289700273 2.913571320292546
```

The test produced the expected result.

Now we are going to identify if the function correctly captures the definition of a derivative of the cost function with respect to a particular feature. V17 is the test feature from the list of test features we are going to be testing. It's respective parameter value was randomly generated a value of 1.

A derivative is the rate of change of a function with respect to a variable.

$$\frac{dy}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Where:

- f is the cost function
- it's input x represents the parameter value for the function V17
- h is the infinitesimally small change in the variable

Below we are going to manually calculate this value using the formula and compare it to the value produced by the actual cost partial derivative function.

```
# Test the cost partial deriavtive is following it's definition for the feature 'V17'
actualCostPd = costPartialDerivative(trainingSet, 'V17', testFeatures, testParameters)
print(actualCostPd)
```

```
2.913571320292546
```

```
# Create a functional equivalence of this to test acuraccy
numerator = cost(trainingSet, testFeatures, [2, -1, 1]) - cost(trainingSet, testFeatures, [2, -1, .9999999999])

functionalCostPD = numerator / (1 - 0.9999999999)
print(functionalCostPD)
```

```
2.917250643929301
```

```
error = 100 * (functionalCostPD - actualCostPd) / actualCostPd
print(error, "% error")
```

```
0.12628225748686073 %
```

The function produced approximately the same value as the formula. It produced a percentage error of 0.126% which is very accurate. This is the desired result.

Chapter 10: Development of the gradient descent algorithm (Objective 2*)

```python
def train(trainingSet, features: list, trackProgress = False):

    learningRate = 1

    # initialise all parameters at 0 as it's as good as being a random guess
    parameters = [0 for x in features]

    # boolean flag indicating if the cost function has reached a minimum
    hasConverged = False

    # infinite loop but we break out when we reach convergence - this is subject to change
    while not hasConverged:

        # temp holders for calculated parameters
        newParameters = parameters

        # initialise values of the partial derviatives of the cost function (with the respect to the current parameter)
        # to null as they cannot be calculated yet
        pdValHistory = [None for x in features]

        # iteratively work out each parameter of index idx
        for j, jthParameter in enumerate(parameters):

            # Calculate the value of the partial derviatives of the cost function (with the respect to the current parameter)
            pdVal = costPartialDerivative(trainingSet, features[j], features, parameters)

            # Calculate the new value of the current parameter
            newParameter = jthParameter - (learningRate * pdVal)

            # update our new parameter and pd value
            newParameters[j] = newParameter
            pdValHistory[j] = pdVal

        # simultaneously update all our variables // we don't have to change pds as they will be recalulcated
        prevParameter = newParameters

        # checks for convergence
        if statistics.mean(pdValHistory) < 0.003:
            # breaks and exits program
            hasConverged = True

    return newParameters
```

```
def train(trainingSet, features: list, learningRate = 1):

    # initialise all parameters at 0 as it's as good as being a random guess
    parameters = [0 for x in features]

    # boolean flag indicating if the cost function has reached a minimum
    hasConverged = False

    # infinite loop but we break out when we reach convergence - this is subject to change
    while not hasConverged:

        # temp holders for calculated parameters
        newParameters = parameters

        # initialise values of the partial derviatives of the cost function (with the respect to the cu
        # to null as they cannot be calculated yet
        pdValHistory = [None for x in features]

        # iteratively work out each parameter of index idx
        for j, jthParameter in enumerate(parameters):

            # Calculate the value of the partial derviatives of the cost function (with the respect to
            pdVal = costPartialDerivative(trainingSet, features[j], features, parameters)

            # Calculate the new value of the current parameter
            newParameter = jthParameter - (learningRate * pdVal)

            # update our new parameter and pd value
            newParameters[j] = newParameter
            pdValHistory[j] = abs(pdVal)

        # simultaneously update all our variables // we don't have to change pds as they will be recalu
        prevParameter = newParameters

        # checks for convergence
        if statistics.median(pdValHistory) < 0.003:
            # breaks and exits program
            hasConverged = True

    return newParameters
```

Originally the training algorithm failed. Since cost partial derivative values can be positive or negative, their magnitudes could not be sorted. So, I fixed this issue by only taking an absolute value with the inbuilt abs() function. Absolute means making the number positive. This fixed the issue and the average partial derivative value started to converge. However it converged quickly since the average I took was a mean. This meant high extreme values could seem small when accompanied by small values. That's why I replaced this with a median. This fixed our problem as the "middle" value was not affected by extreme values.

Chapter 11: The testing/evaluative function (Objective 3*)

The aim of this function is to complete two goals:

1. Take in a trained model (a list of features and their tuned parameters) and a set of data to test it on. Apply the model to predict the probability and then fraud classification of each transaction in a given testing set.

2. After each prediction, determine by comparing this to the model's predicted classification, whether the model got the classification correct and then whether this correct prediction was made for a fraudulent positive/negative transaction. ecord this classification by incrementing a counter variable.

These four classifications are true positive, true negative, false positive and false negative. This data can be represented by a confusion matrix which is shown by the diagram on the right. In our subroutine, they will be stored as variables whose names will be abbreviated to save space.

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Lastly, the four variables will be packed into a tuple and returned by the subroutine.

```
def test(testingSet, featureNames, tunedParameters):
    """
    Evaluates every case where the model correctly/incorrectly predicted positive or negative
    """
    # where True Positive = tp, False Negative = fn, etc
    tp = tn = fp = fn = 0

    for index, testingExample in testingSet.iterrows():

        # Use our tuned parameters and selected features to calculate a probability of a test example being fraudulent
        probability = sigmoid(testingExample, featureNames, tunedParameters)

        # Decision bounday
        if probability > 0.5:
            predictedClass= 1
        else:
            predictedClass= 0

        # Assign these predicted values to the respective set
        testingSet.at[index, "Predicted_Prob"] = probability
        testingSet.at[index, "Predicted_Class"] = predictedClass

        # Get the actual target class
        actualClass = testingSet.loc[index]["Class"]

        if actualClass == predictedClass:
            if actualClass == 1:
                tp += 1
            else:
                tn += 1
        else:
            if predictedClass == 1:
                fp += 1
            else:
                fn += 1
    return (tp, tn, fp, fn)
```

Chapter 12: Evaluation Metrics (Objective 3*)

The evaluation metrics will be calculated by a simple function which inputs the test/evaluative results in the form of a tuple. On the first line this tuple is unpacked onto 4 variables which store a count of each classification. We then use the mathematical definition of our evaluation metrics to calculate a value:

```
# % of all fraudulent transactions, predicted as actually fraudulent
def recall(testResults):
    tp, tn, fp, fn = testResults
    # true positives / true positives + false negatives
    return tp / (tp + fn)

# % of all transactions, which were actually fraudulent
def precision(testResults):
    tp, tn, fp, fn = testResults
    # true positives / true positives + false positive
    return tp / (tp + fp)
```

Although it may seem quite unlikely for something to go wrong with such a simple function, it is possible to have no (zero) true positives and also no false or true positives. In these cases, the denominator would be 0 causing an error, since anything divided by 0 equals undefined. I handled this error by printing a message to the screen informing the user of this instead of raising an exception. Lastly, I returned 0 to show the inaccuracy of the program.

```
# % of all fraudulent transactions, predicted as actually fraudulent
def recall(testResults):
    tp, tn, fp, fn = testResults
    try:
        # true positives / true positives + false negatives
        return tp / (tp + fn)
    except:
        print("Can't calculate recall as there were no true positives or false negatives.")
        return 0

# % of all transactions, which were actually fraudulent
def precision(testResults):
    tp, tn, fp, fn = testResults
    try:
        # true positives / true positives + false positive
        return tp / (tp + fp)
    except:
        print("Can't calculate precision as there were no true positives or false positives.")
        return 0
```

Chapter 13: Cross_validation (Objective 3*)

The aim of this function is to return an average for the evaluation metrics calculated over k folds during cross validation (explained in design module 1.2)

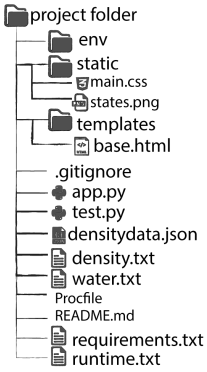
```python
def cross_validation(features, k, dataset, trackProgress):
    """
    Returns the average precision and recall values when training and testing a dataset k times.
    """
    sampleSize = len(dataset) // k
    folds = []

    # Split dataset into k folds
    for i in range(k):

        start = i * sampleSize
        end = (i + 1) * sampleSize - 1 if i != k - 1 else len(dataset) - 1 # final index if last group
        folds.append(dataset.iloc[start: end]) # the- kth fold: testing)

    # Now our folds are created we can iterate through it:

    precisionVals = []
    recallVals = []

    for i in range(k):

        folds_copy = folds.copy() # create a copy of the folds

        # Create the training/test sets
        testingSet = folds_copy.pop(i) # kth fold
        trainingSet =  pd.concat(folds_copy) # remaining k-1 folds

        # Train our model
        tunedParameters = train(trainingSet, features, trackProgress)

        # Test our model
        testResults = test(testingSet, features, tunedParameters)
        print(testResults)

        # Add results for future calculations
        precisionVals.append(precision(testResults))
        recallVals.append(recall(testResults))
        print(precisionVals, recallVals)

    # Final evaluation
    plt.boxplot((precisionVals, recallVals))
    plt.show()

    return statistics.mean(precisionVals), statistics.mean(recallVals)
```

Chapter 14: Setting up the web application - (Objective 4*)

We are going to work on the actual fraud detection program at the same time as building the web application that will host it. We are switching environments to the Visual Studio Code IDE for developing the web app.

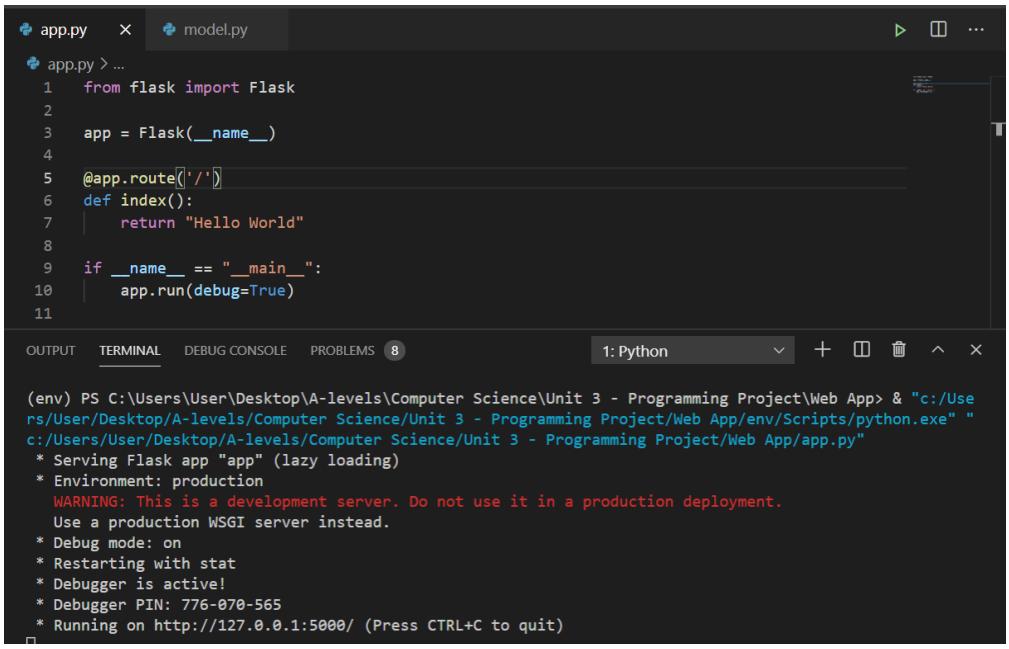
Once the machine learning model is finalised we are going to bring the code into the web app and have it process fraud cases on the backend and display it on the front end. Visual Studio Code has an inbuilt terminal which will allow us to launch the web app and alter it through commands and also install external libraries into the project using PIP Install and keep them contained in a virtual environment.

A framework is a platform for developing software applications. By using a framework, we can save lots of time and effort allowing us to focus on building an unique feature for their web based projects rather than re-inventing by coding. Flask is a framework for building simple web apps and it is written in Python which is also the language of choice for machine learning. This compatibility is why I opted to use flask.



The diagram on the right shows a standard project folder. I will use this to aid my development of the web application. App.py is where we are going to start the development of our web app since this is the location of where we create an app object, an instance of the Flask object, and the central configuration object for the entire application.

On the first line we import flask and we create an instance of the application called app. A decorator (denoted by the @ sign) allows a user to add new functionality to an existing object without modifying its structure. Flask uses this to configure routes for the app object. Routing is the process of binding a URL to a function which is invoked when an URL address is entered into the search box of the browser.



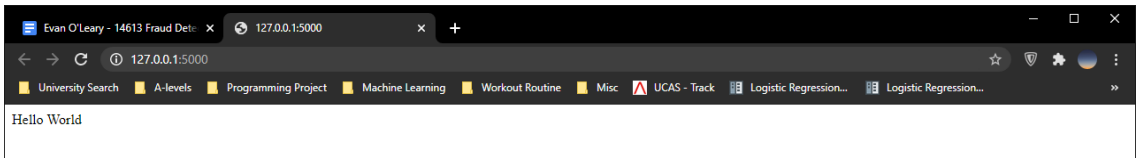
The above diagram shows the code for our first iteration of developing the app.py file.

In the third line of code we configure a route for when the program is launched. This is enabled with an input of '/' meaning the relative path is empty and the uniform resource locator should direct the user to the home or index page. Below the decorator is a linked function which returns the string "hello world". This body of the function is where we will later render templates instead and add functionality for dynamically loading the page with content.

Whenever the Python interpreter reads a script, it declares a special variable called __name__ and then executes the code found in the script. During the script's execution, modules/libraries are imported so the high-level code in the file is rand and translated for it to be linked to the program. '__main__' is the name of the scope in which top-level code executes. Therefore a module's __name__ is set equal to '__main__' when read directly from a script but not when it is imported.

This is what we check in the selection statement in the second last line of code. This condition will only be true when we run it directly from the IDE using the little green play button in the right top corner. Hence when it is true, we are running the server in a production mode. For development purposes, we use debug mode (debug = True) which makes the server restart when we add new code to our Flask Application.

In the bottom of the screenshot above shows the terminal's output when the web app is launched by running the app.py file. The text shows the active settings and status of the app. It confirms debugger mode is active. If we paste the local host link into our browser, we can view our rendered web app. The diagram below shows the first successful launch!



Chapter 15: Creating Routes - (Objective 4*)

In this chapter, I'm going to demonstrate how we use routing to bind URLs to their respective HTML templates in the app.py file.

When the index function is called, it renders the HTML file index.html. We will need these functions for each new view we create. A view is simply the web page we see. A page that displays the text, images of a website. If you build websites with HTML, then every page you create is the view like the homepage, about page, and the contact page. We create a template for the view so it can be filled with content from the database.

Chapter 16: Structuring the content of the master HTML file - (Objective 4*)

An example of how flask speeds up development of our web application is by letting us import and reuse HTML code blocks. This is shown by the '{% %}' "Ginger 2" syntax. These are the sections where we choose for code to be inserted into our base template. We use this to create a skeleton or base layout for all pages. This should be parts of the webpage common to all pages like a header or navigation bar. This means we only have to write this code once and not duplicate code wasting space. We are going to use the balsamiq mockup screenshot shown in the screen design section to base our project on.

templates > base.html > html > body > div.container > main

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
    {% block head %}{% endblock %}
</head>
<body>
    <div class="container">
        <header class="header-container">
            <div class="logo-container">
                <h1>Fraud Detect</h1>
            </div>
            <div>
            <!-- Time -->
            </div>
        </header>
        <div class="sidebar-container">
            <ul class="sidebar">
                <li><a href="{{ url_for('index') }}">Summary</a></li>
                <li><a href="">Manage</a></li>
                <li><a href="">Settings</a></li>
                <li><a href="{{ url_for('help') }}">Help</a></li>
            </ul>
        </div>
        <main>
            {% block body %}{% endblock %}
        </main>
    </div>
</body>
</html>
```

Between the head tags, the stylesheet linked uses the url_for method to retrieve the file main.css file from the css folder. All static content like css and script files is stored in a main folder of the project called ‘static’. Main.css will contain the styling rules for the base HTML skeleton. Stylesheets independent to specific pages should be kept separate and can be imported in the head block in the separate page HTML. This way classes won’t be accidentally edited and errors will be isolated.

In the next HTMl page shown below we’ll see an example of code being used as a template using the ‘extends’ command. The page is incomplete but demonstrates how we can make independent changes to pages. For instance, not all pages should have the same title, therefore we can put this in the page’s independent head block, and it’ll be inserted into the base template’s head tags. Similarly I demonstrate this can be done for code in the body as well. Our sample text shows this with the h2 tag which makes the ‘fraud detected’ heading.

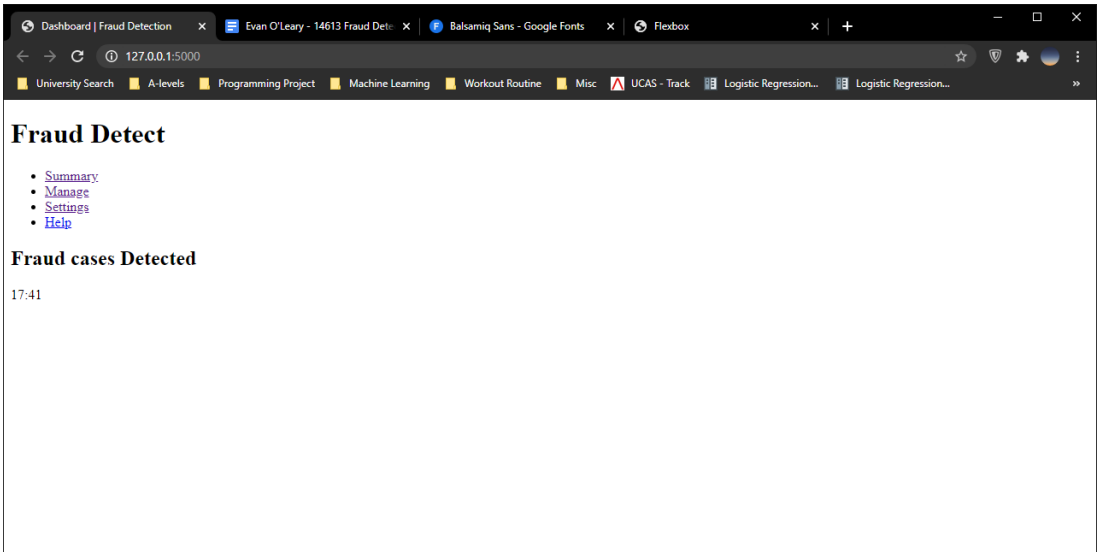
templates > index.html > ...

```
<!-- HTML comment syntax btw: this brings in our base structure -->
{% extends 'base.html' %}

{% block head %}
    <title>Dashboard | Fraud Detection </title>
{% endblock %}

{% block body %}
    <div class="main-cloumn">
        <div class="fraud-detected">
            <header>
                <h2>Fraud cases Detected</h2>
                <span>17: 41</span>
            </header>
            <table>

            </table>
        </div>
    </div>
    <div class="main-column">
        <div class="bar-graph">

        </div>
        <div class="line-graph">

        </div>
    </div>
{% endblock %}
```

Chapter 17: Styling the content of the master HTML file - (Objective 4*)

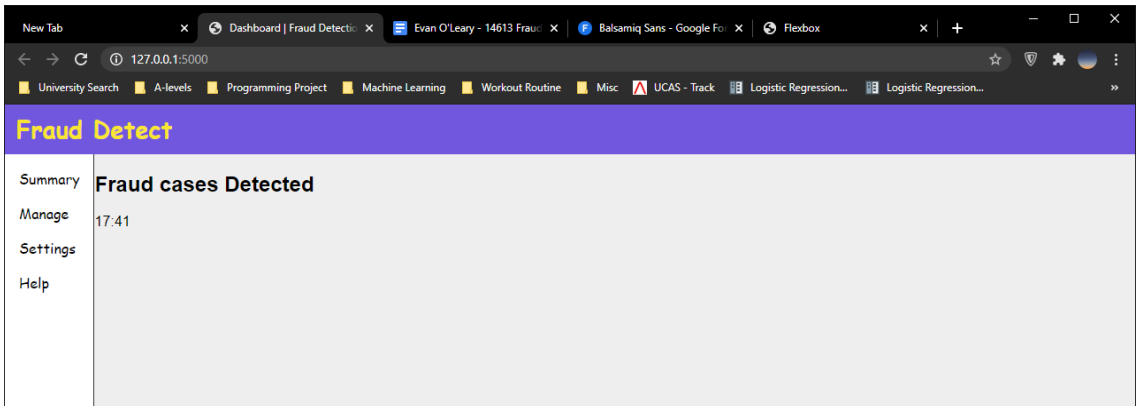
This might be the only time I talk about changes to the styling in detail since it's non-essential to the success of our program. However this stylesheet will be imported into all of our pages and sets a precedent for the kind of design language which will persist consistently throughout our project. Below shows a snippet of the HTML code from base.html created in the previous chapter.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
    {% block head %}{% endblock %}
</head>
```



static > css > # main.css > .logo-container

```css
/* Main content */
body{
    margin: 0;
    font-family: sans-serif;
}

.container > div, main {
    height: 94vh;
}

main{
    background-color: #eee;
    display: flex;
}

/* Header section */

.header-container {
    height: 6vh;
    background: #7057dd;
    font-family: cursive;
}

.logo-container {
    color: #fae630;
    padding: 0.5rem;
    font-weight: 0;
}

h1 {
    font-size: 1.7em;
    margin: 0 5px;
}
```



Chapter 18: Deploying the model - (Objective 4*)

Now is the first test deployment of our model. This doesn't mean we are going to stop developing the feature selection model or even the learning algorithm - more functions and development to come! However this is the first test run of giving a primitive graphical user interface for our model. Although this solution will be temporary, I will end up reusing a version of this code for our final product.

The aim of this chapter is to:

1. Successfully train a model from within the flask program's runtime.
2. Test the model on a test dataset
3. Output the features selected
4. Display the tested dataset of transactions into

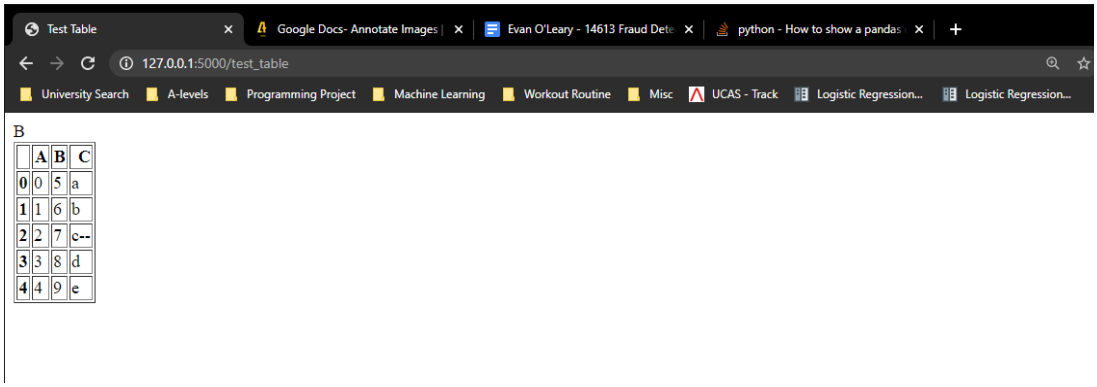
Our first priority is to test with a small sample random data frame that we can print to the screen. In the screenshot below we configure a route for this test. We then proceed to pass the dataframe as HTML as an argument for the 'tables' parameter. This allows the data to be accessed in the routed HTML file.

```python
@app.route('/test_table', methods=("POST", "GET"))
def html_table():
    # Create a small sample data frame
    df = pd.DataFrame({
        'A': [0, 1, 2, 3, 4],
        'B': [5, 6, 7, 8, 9],
        'C': ['a', 'b', 'c--', 'd', 'e']
    })
    # Pass this dataframe as HTML as an argument for our table
    return render_template('test_table.html', tables=[df.to_html(classes='data')], titles=df.columns.values)
```

In the routed HTML file we have a loop in the ginger 2 syntax. This automatically formats the table and fills the cells with the field values of our HTML converted dataframe.

```
templates > test_table.html > ...
1   <!DOCTYPE html>
2   <html lang="en">
3       <head>
4           <meta charset="UTF-8">
5           <title>Test Table</title>
6       </head>
7       <body>
8
9       {% for table in tables %}
10          {{ titles[loop.index] }}
11          {{ table|safe }}
12      {% endfor %}
13
14      </body>
15  </html>
16
```

Below is a screenshot of the web page rendered by this above code after the test_table URL is entered.

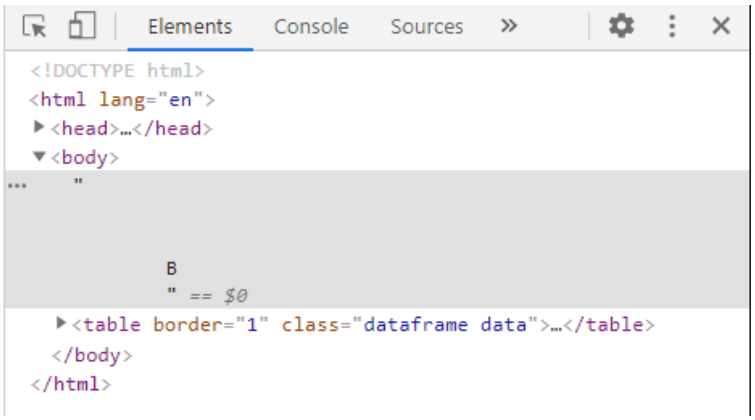


	A	B	C
0	0	5	a
1	1	6	b
2	2	7	c--
3	3	8	d
4	4	9	e

We got the desired result. However there was an unexpected additional resulting markup. It showed the name of the second column title. I used the inspect element tool provided by Chrome's developer tools to see if I could identify why it appeared.

B

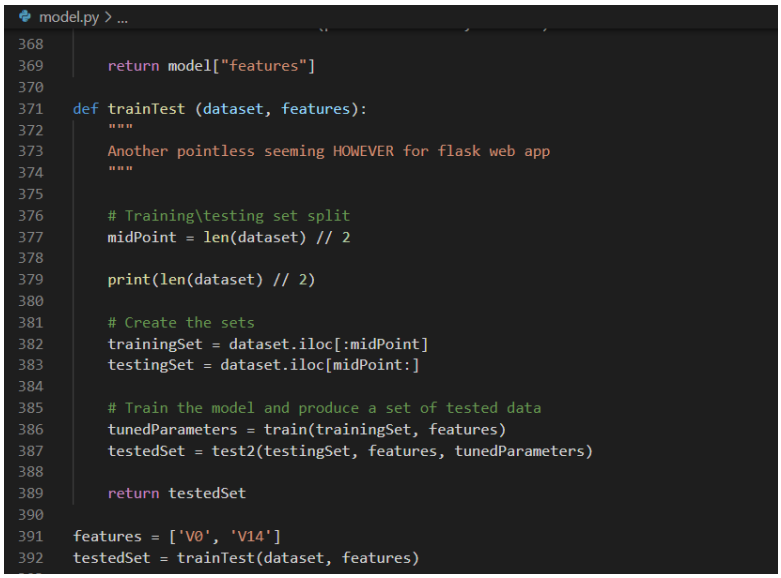
	A	B	C
0	0	5	a
1	1	6	b
2	2	7	c--
3	3	8	d
4	4	9	e



As you can see it showed us no useful information to help us determine what is triggering it. But as it is not disrupting our progress I am going to move on and continue with our plan. We need to train and test a model on a dataset. Instead of using the extensive feature selection method which will take a long time to run, I am going to define a simple model of just two features: [‘V0’, ‘V14’].

We need to copy the essential code from our jupyter notebook and paste it all into a python file. For now, I am going to call this file model.py. However this name may conflict with a component of Flask which concerns databases in the future which we will stay aware of and see to later on.

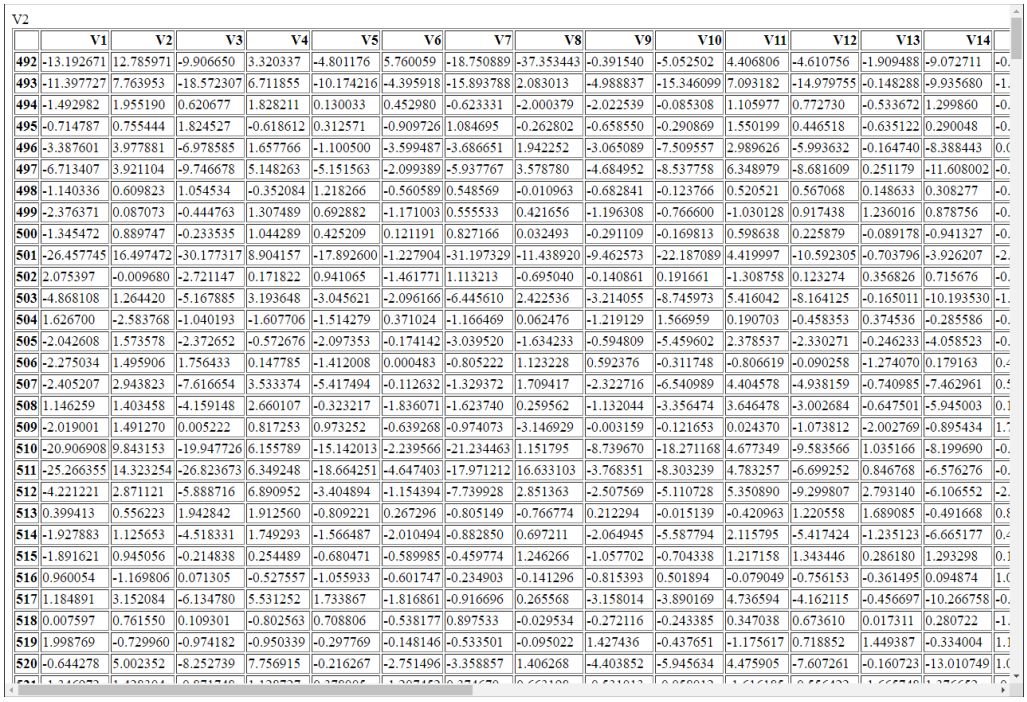
I use these features to produce a tested dataset in model.py. Here is the code below:



We then replace the sample data frame with the new tested data frame in the HTML table function we recently created in app.py. The model.py file is imported at the top of app.py so the tested dataframe is referenced as ‘model.testedSet’. This is shown below.

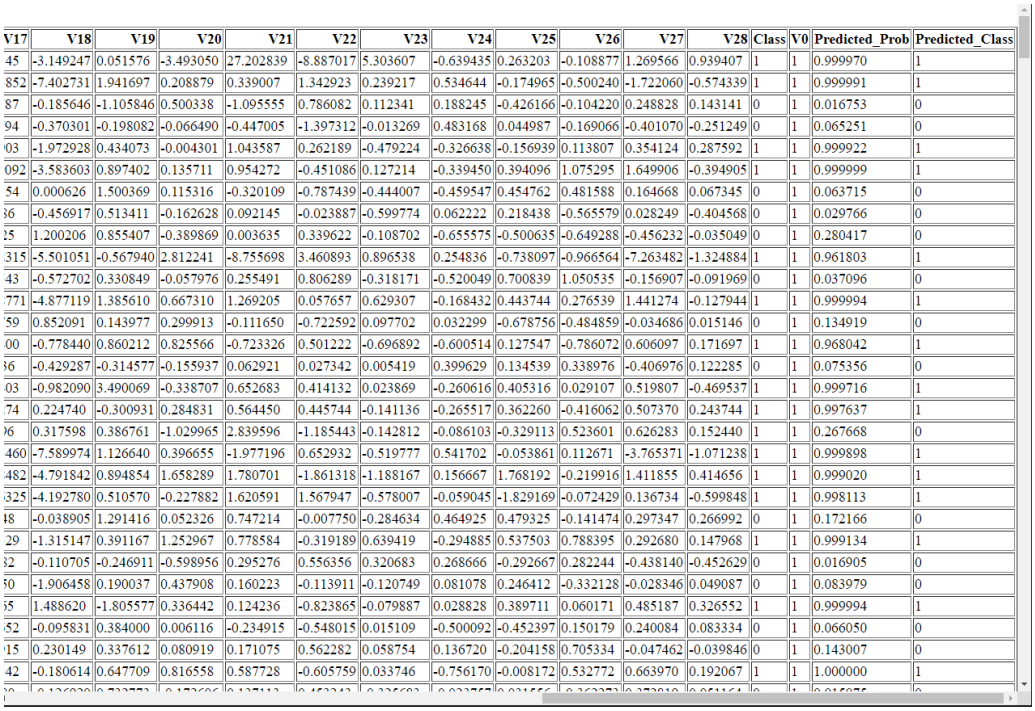
```
@app.route('/test_table', methods=("POST", "GET"))
def html_table():
    return render_template('test_table.html',  tables=[testedSet.to_html(classes='data')], titles=testedSet.columns.values)
```

Running app.py with the ‘test_table’ path in the search box renders:



V2

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	
492	-13.192671	12.785971	-9.906650	3.320337	-4.801176	5.760059	-18.750889	-37.353443	-0.391540	-5.052502	4.406806	-4.610756	-1.909488	-9.072711	-0.
493	-11.397727	7.763953	-18.572307	6.711855	-10.174216	-4.395918	-15.893788	2.083013	-4.988837	-15.346099	7.093182	-14.979755	-0.148288	-9.935680	-1.
494	-1.492982	1.955190	0.620677	1.828211	0.130033	0.452980	-0.623331	-2.000379	-2.022539	-0.085308	1.105977	0.772730	-0.533672	1.299860	-0.
495	-0.714787	0.755444	1.824527	-0.618612	0.312571	-0.909726	1.084695	-0.262802	-0.658550	-0.290869	1.550199	0.446518	-0.635122	0.290048	-0.
496	-3.387601	3.977881	-6.978585	1.657766	-1.100500	-3.599487	-3.686651	1.942252	-3.065089	-7.509557	2.989626	-5.993632	-0.164740	-8.388443	0.
497	-6.713407	3.921104	-9.746678	5.148263	-5.151563	-2.099389	-5.937767	3.578780	-4.684952	-8.537758	6.348979	-8.681609	0.251179	-11.608002	-0.
498	-1.140336	0.609823	1.054534	-0.352084	1.218266	-0.560589	0.548569	-0.010963	-0.682841	-0.123766	0.520521	0.567068	0.148633	0.308277	-0.
499	-2.376371	0.087073	-0.444763	1.307489	0.692882	-1.171003	0.555533	0.421656	-1.196308	-0.766600	-1.030128	0.917438	1.236016	0.878756	-0.
500	-1.345472	0.889747	-0.233535	1.044289	0.425209	0.121191	0.827166	0.032493	-0.291109	-0.169813	0.598638	0.225879	-0.089178	-0.941327	-0.
501	-26.457745	16.497472	-30.177317	8.904157	-17.892600	-1.227904	-31.197329	-11.438920	-9.462573	-22.187089	4.419997	-10.592305	-0.703796	-3.926207	-2.
502	2.075397	-0.009680	-2.721147	0.171822	0.941065	-1.461771	1.113213	-0.695040	-0.140861	0.191661	-1.308758	0.123274	0.356826	0.715676	-0.
503	-4.868108	1.264420	-5.167885	3.193648	-3.045621	-2.096166	-6.445610	2.422536	-3.214055	-8.745973	5.416042	-8.164125	-0.165011	-10.193530	-1.
504	1.626700	-2.583768	-1.040193	-1.607706	-1.514279	0.371024	-1.166469	0.062476	-1.219129	1.566959	0.190703	-0.458353	0.374536	-0.285586	-0.
505	-2.042608	1.573578	-2.372652	-0.572676	-2.097353	-0.174142	-3.039520	-1.634233	-0.594809	-5.459602	2.378537	-2.330271	-0.246233	-4.058523	-0.
506	-2.275034	1.495906	1.756433	0.147785	-1.412008	0.000483	-0.805222	1.123228	0.592376	-0.311748	-0.806619	-0.090258	-1.274070	0.179163	0.
507	-2.405207	2.943823	-7.616654	3.533374	-5.417494	-0.112632	-1.329372	1.709417	-2.322716	-6.540989	4.404578	-4.938159	-0.740985	-7.462961	0.
508	1.146259	1.403458	-4.159148	2.660107	-0.323217	-1.836071	-1.623740	0.259562	-1.132044	-3.356474	3.646478	-3.002684	-0.647501	-5.945003	0.
509	-2.019001	1.491270	0.005222	0.817253	0.973252	-0.639268	-0.974073	-3.146929	-0.003159	-0.121653	0.024370	-1.073812	-2.002769	-0.895434	1.
510	-20.906908	9.843153	-19.947726	6.155789	-15.142013	-2.239566	-21.234463	1.151795	-8.739670	-18.271168	4.677349	-9.583566	1.035166	-8.199690	-0.
511	-25.266355	14.323254	-26.823673	6.349248	-18.664251	-4.647403	-17.971212	16.633103	-3.768351	-8.303239	4.783257	-6.699252	0.846768	-6.576276	-0.
512	-4.221221	2.871121	-5.888716	6.890952	-3.404894	-1.154394	-7.739928	2.851363	-2.507569	-5.110728	5.350890	-9.299807	2.793140	-6.106552	-2.
513	0.399413	0.556223	1.942842	1.912560	-0.809221	0.267296	-0.805149	-0.766774	0.212294	-0.015139	-0.420963	1.220558	1.689085	-0.491668	0.
514	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	-5.587794	2.115795	-5.417424	-1.235123	-6.665177	0.
515	-1.891621	0.945056	-0.214838	0.254489	-0.680471	-0.589985	-0.459774	1.246266	-1.057702	-0.704338	1.217158	1.343446	0.286180	1.293298	0.
516	0.960054	-1.169806	0.071305	-0.527557	-1.055933	-0.601747	-0.234903	-0.141296	-0.815393	0.501894	-0.079049	-0.756153	-0.361495	0.094874	1.
517	1.184891	3.152084	-6.134780	5.531252	1.733867	-1.816861	-0.916696	0.265568	-3.158014	-3.890169	4.736594	-4.162115	-0.456697	-10.266758	-0.
518	0.007597	0.761550	0.109301	-0.802563	0.708806	-0.538177	0.897533	-0.029534	-0.272116	-0.243385	0.347038	0.673610	0.017311	0.280722	-1.
519	1.998769	-0.729960	-0.974182	-0.950339	-0.297769	-0.148146	-0.533501	-0.095022	1.427436	-0.437651	-1.175617	0.718852	1.449387	-0.334004	1.
520	-0.644278	5.002352	-8.252739	7.756915	-0.216267	-2.751496	-3.358857	1.406268	-4.403852	-5.945634	4.475905	-7.607261	-0.160723	-13.010749	1.



V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Class	V0	Predicted_Prob	Predicted_Class
45	-3.149247	0.051576	-3.493050	27.202839	-8.887017	5.303607	-0.639435	0.263203	-0.108877	1.269566	0.939407	1	1	0.999970	1
852	-7.402731	1.941697	0.208879	0.339007	1.342923	0.239217	0.534644	-0.174965	-0.500240	-1.722060	-0.574339	1	1	0.999991	1
87	-0.185646	-1.105846	0.500338	-1.095555	0.786082	0.112341	0.188245	-0.426166	-0.104220	0.248828	0.143141	0	1	0.016753	0
94	-0.370301	-0.198082	-0.066490	-0.447005	-1.397312	-0.013269	0.483168	0.044987	-0.169066	-0.401070	-0.251249	0	1	0.065251	0
03	-1.972928	0.434073	-0.004301	1.043587	0.262189	-0.479224	-0.326638	-0.156939	0.113807	0.354124	0.287592	1	1	0.999922	1
092	-3.583603	0.897402	0.135711	0.954272	-0.451086	0.127214	-0.339450	0.394096	1.075295	1.649906	-0.394905	1	1	0.999999	1
54	0.000626	1.500369	0.115316	-0.320109	-0.787439	-0.444007	-0.459547	0.454762	0.481588	0.164668	0.067345	0	1	0.063715	0
36	-0.456917	0.513411	-0.162628	0.092145	-0.023887	-0.599774	0.062222	0.218438	-0.565579	0.028249	-0.404568	0	1	0.029766	0
25	1.200206	0.855407	-0.389869	0.003635	0.339622	-0.108702	-0.655575	-0.500635	-0.649288	-0.456232	-0.035049	0	1	0.280417	0
315	-5.501051	-0.567940	2.812241	-8.755698	3.460893	0.896538	0.254836	-0.738097	-0.966564	-7.263482	-1.324884	1	1	0.961803	1
43	-0.572702	0.330849	-0.057976	0.255491	0.806289	-0.318171	-0.520049	0.700839	1.050535	-0.156907	-0.091969	0	1	0.037096	0
771	-4.877119	1.385610	0.667310	1.269205	0.057657	0.629307	-0.168432	0.443744	0.276539	1.441274	-0.127944	1	1	0.999994	1
59	0.852091	0.143977	0.299913	-0.111650	-0.722592	0.097702	0.032299	-0.678756	-0.484859	-0.034686	0.015146	0	1	0.134919	0
00	-0.778440	0.860212	0.825566	-0.723326	0.501222	-0.696892	-0.600514	0.127547	-0.786072	0.606097	0.171697	1	1	0.968042	1
36	-0.429287	-0.314577	-0.155937	0.062921	0.027342	0.005419	0.399629	0.134539	0.338976	-0.406976	0.122285	0	1	0.075356	0
03	-0.982090	3.490069	-0.338707	0.652683	0.414132	0.023869	-0.260616	0.405316	0.029107	0.519807	-0.469537	1	1	0.999716	1
74	0.224740	-0.300931	0.284831	0.564450	0.445744	-0.141136	-0.265517	0.362260	-0.416062	0.507370	0.243744	1	1	0.997637	1
06	0.317598	0.386761	-1.029965	2.839596	-1.185443	-0.142812	-0.086103	-0.329113	0.523601	0.626283	0.152440	1	1	0.267668	0
460	-7.589974	1.126640	0.396655	-1.977196	0.652932	-0.519777	0.541702	-0.053861	0.112671	-3.765371	-1.071238	1	1	0.999898	1
482	-4.791842	0.894854	1.658289	1.780701	-1.861318	-1.188167	0.156667	1.768192	-0.219916	1.411855	0.414656	1	1	0.999020	1
325	-4.192780	0.510570	-0.227882	1.620591	1.567947	-0.578007	-0.059045	-1.829169	-0.072429	0.136734	-0.599848	1	1	0.998113	1
48	-0.038905	1.291416	0.052326	0.747214	-0.007750	-0.284634	0.464925	0.479325	-0.141474	0.297347	0.266992	0	1	0.172166	0
29	-1.315147	0.391167	1.252967	0.778584	-0.319189	0.639419	-0.294885	0.537503	0.788395	0.292680	0.147968	1	1	0.999134	1
32	-0.110705	-0.246911	-0.598956	0.295276	0.556356	0.320683	0.268666	-0.292667	0.282244	-0.438140	-0.452629	0	1	0.016905	0
50	-1.906458	0.190037	0.437908	0.160223	-0.113911	-0.120749	0.081078	0.246412	-0.332128	-0.028346	0.049087	0	1	0.083979	0
55	1.488620	-1.805577	0.336442	0.124236	-0.823865	-0.079887	0.028828	0.389711	0.060171	0.485187	0.326552	1	1	0.999994	1
52	-0.095831	0.384000	0.006116	-0.234915	-0.548015	0.015109	-0.500092	-0.452397	0.150179	0.240084	0.083334	0	1	0.066050	0
15	0.230149	0.337612	0.080919	0.171075	0.562282	0.058754	0.136720	-0.204158	0.705334	-0.047462	-0.039846	0	1	0.143007	0
42	-0.180614	0.647709	0.816558	0.587728	-0.605759	0.033746	-0.756170	-0.008172	0.532772	0.663970	0.192067	1	1	1.000000	1

This is the result we desire. I had to use two screenshots to capture the full horizontal width. You can see the rightmost column shows that the model predicted and saved it to the column. Success! However when scrolling to the bottom, I notice that the number of rows printed is 984. This is the total number of records in our balanced dataset, however the number of rows in the tested dataset should be half this since it is split.

To test the function in jupyter to diagnose whether this was an error with the function itself or due to the new flask environment altering its behaviour:

I added a print statement which outputs the number of rows in the testing set so that I can compare the value being processed in the body of the function and the outputted value. I gave the function some arbitrary features and run the the code:

```
def trainTest (dataset, features):

    # Training\testing set split
    midPoint = len(dataset) // 2

    # Create the sets
    trainingSet = dataset.iloc[:midPoint]
    testingSet = dataset.iloc[midPoint:]

    print("Number of rows in testing set:", len(testingSet))

    # Train the model and produce a set of tested data
    tunedParameters = train(trainingSet, features)
    testedSet = test2(testingSet, features, tunedParameters)

    return testedSet
```

```
testResults = trainTest(dataset, ['V0', 'V14', 'V3'])
print("Outputted number of records in the tested set:", len(testResults))
```

```
Number of rows in testing set: 492
Outputted number of records in the tested set: 492
C:\Users\User\anaconda3\lib\site-packages\ipykernel_launcher.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
C:\Users\User\anaconda3\lib\site-packages\ipykernel_launcher.py:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

It seems to produce a warning we ignored before and also the correct expected value. I am going to stop the warning by changing the function so we don't have to worry about it causing any errors.

'SettingWithCopyWarning' informs you that your operation might not have worked as expected and that you should check the result to make sure you haven't made a mistake since some actions in pandas can return a view of your data, and others will return a copy. We fix this by removing ambiguity and confirm to flask that we are taking a copy when slicing the data we want for our training/testing sets.

```
# Create the sets
trainingSet = dataset.loc[:midPoint].copy()
testingSet = dataset.loc[midPoint:].copy()
```

We run the same test but print the testset instead:

```
testResults = trainTest(dataset, ['V0', 'V14', 'V3'])
print(testResults)
```

```
           V1         V2         V3         V4         V5        V6  \
492   1.909423  -0.368523  -3.150923  -0.603011   2.517304  3.209951
493  -0.870462  -0.352276   1.562985  -0.405168   0.063030 -0.697392
494  -0.937843   3.462889  -6.445104   4.932199  -2.233983 -2.291561
495  -1.437443   1.421577   1.382786   0.864135  -0.171392  0.639759
496  -2.986466  -0.000891   0.605887   0.338338   0.685448 -1.581954
..         ...        ...        ...        ...        ...       ...
979  -0.440095   1.137239  -3.227080   3.242293  -2.033998 -1.618415
980  -1.105524   1.077549   1.636962   1.159112  -0.332010  0.007124
981  -2.423535   1.659093  -3.071421   2.588033   1.135791 -1.892388
982  -4.641893   2.902086  -1.572939   2.507299  -0.871783 -1.040903
983 -25.825982  19.167239 -25.390229  11.125435 -16.682644  3.933699

           V7         V8         V9        V10  ...       V25       V26  \
492  -0.457625   0.818622   0.869339  -0.899841  ... -0.047635 -0.072892
493   0.144253   0.049521   0.562188  -0.963091  ... -0.168419  0.162588
494  -5.695594   1.338825  -4.322377  -8.099119  ...  0.906802  1.165784
495   0.172902   0.741150  -0.054219  -0.134464  ...  0.456407 -0.778942
496   0.504206  -0.233403   0.636768   1.010291  ...  0.448552  0.193490
..         ...        ...        ...        ...  ...       ...       ...
979  -3.028013   0.764555  -1.801937  -4.711769  ...  0.606434 -0.315433
980   0.057065   0.711323   0.168748  -0.660582  ...  0.333478 -0.772406
981  -2.588418  -2.226592  -1.670173  -3.508925  ...  0.299285 -0.263801
982  -1.593901  -3.254905   1.908963   1.077418  ... -0.354558 -0.611764
983 -37.060311 -28.759799 -11.126624 -23.228255  ... -0.671983 -0.209431

          V27       V28  Class  V0  Amount_Scaled  Time_Scaled  \
492  0.004411 -0.016007      0   1       0.530986     0.907071
493  0.108782  0.172982      0   1       0.936352     0.557196
494  1.374495  0.729889      1   1      -0.307413    -0.005733
495  0.216097  0.121508      0   1       0.016768     0.680647
496  1.214588 -0.013923      1   1      -0.282401    -0.350639
..        ...       ...    ...  ..            ...          ...
979  0.768291  0.459623      1   1       2.868721    -0.171771
980  0.226486  0.056245      0   1      -0.246349     0.679402
981  0.150156  0.292112      1   1      -0.177601     0.778404
982 -3.908080 -0.671248      1   1      -0.148257    -0.809161
983 -4.950022 -0.448413      1   1      -0.275554     0.195268

     Predicted_Prob  Predicted_Class
492        0.477787                0
493        0.081248                0
494        0.999999                1
495        0.057560                0
496        0.195594                0
..              ...              ...
979        0.997496                1
980        0.041571                0
981        0.981747                1
982        0.847329                1
983        0.999481                1

[492 rows x 34 columns]
```

The warning message was successfully removed. Also spotted that there was no reason for concern about the entire dataset being printed. It was simply that the index integers ranged from 492 - 983 not 1 - 983 since the index IDs were also sliced.

Chapter 19: Saving the model and tested dataset. - (Objective 4*)

Currently, every time we launch the web app, the model retrains and tests a dataset. This is inefficient and will worsen the development process further on into the production of our program. The aim of this chapter is to save our tested dataframe and model into a file so we can bring in this information when we reload the application without spending time training the model.

For now, we are going to create a separate tested dataset comma separated value file for when we train/test a model. We'll call the file 'testedSet.csv'. If it exists then no need to train/test a model, read the csv contents to a pandas dataframe and export it to HTML ready to output. And if not, train/test the model and create this CSV file.

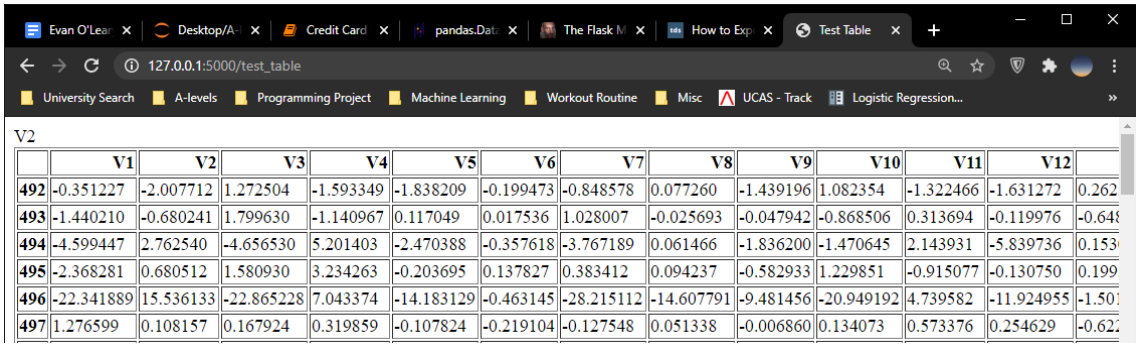
If the existence of the train/tested dataset depends on the condition that the web app has run before, the scope which controls whether this dataset shall exist, must also create the dataset. I am going to achieve this by reading the large credit card dataset from the app.py file. Additionally I am going to make the previous code in the model.py's global scope for balancing the dataset into a function we can call from app.py.

```
# importing required libraries
import pandas as pd
import matplotlib.pyplot as plt
import math, statistics, random, os

def balanceDataset(largeDataset):

    # Separating the portion of records that were fraulent
    fraud = largeDataset.loc[largeDataset.Class == 1]

    # Get the amount of fraudulent records
    fraudCount = len(fraud)

    # Take the random sample of non-fraudulent records
    non_fraud = largeDataset.loc[largeDataset.Class == 0].sample(fraudCount)

    frames = [fraud, non_fraud]
    dataset = pd.concat(frames).sample(frac=1).reset_index(drop=True) # shuffle again

    # drop unscaled Amount and Time for correlation analysis
    dataset = dataset.drop(['Time', 'Amount'], axis = 1)

    return dataset
```

Below shows this decision to reload when file exists or train/test the model and create the file for it:

```python
@app.route('/test_table', methods=("POST", "GET"))
def html_table():
    # Check if a model has already been trained and a set has already been tested
    if os.path.exists("dataset/tested_set.csv"):
        # bring in pre-tested dataset
        testedSet = pd.read_csv("dataset/tested_set.csv")
    else:
        # doesnt exist so train/test the model
        transactions = pd.read_csv("dataset/creditcard.csv")  # Import the large datset locally
        transactions["V0"] = 1 # assign the 0th feature
        dataset = model.balanceDataset(transactions) # balance/reduce dataset
        features = ['V0', 'V14'] # define some test model
        testedSet = model.trainTest(dataset, features)
        # Save to csv to prevent future wasted time
        testedSet.to_csv("dataset/tested_set.csv")
    # render table
    return render_template('test_table.html', tables=[testedSet.to_html(classes='data')], titles=testedSet.columns.values)
```

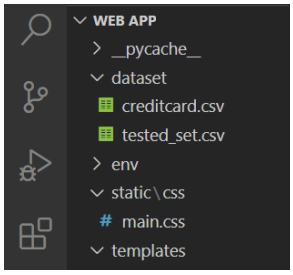
Running the web app produces the desired result for when there is no tested csv file present:



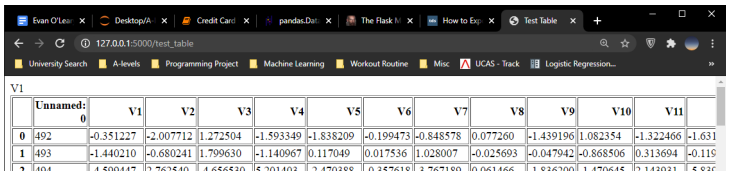
V2

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
492	-0.351227	-2.007712	1.272504	-1.593349	-1.838209	-0.199473	-0.848578	0.077260	-1.439196	1.082354	-1.322466	-1.631272	0.262
493	-1.440210	-0.680241	1.799630	-1.140967	0.117049	0.017536	1.028007	-0.025693	-0.047942	-0.868506	0.313694	-0.119976	-0.648
494	-4.599447	2.762540	-4.656530	5.201403	-2.470388	-0.357618	-3.767189	0.061466	-1.836200	-1.470645	2.143931	-5.839736	0.153
495	-2.368281	0.680512	1.580930	3.234263	-0.203695	0.137827	0.383412	0.094237	-0.582933	1.229851	-0.915077	-0.130750	0.199
496	-22.341889	15.536133	-22.865228	7.043374	-14.183129	-0.463145	-28.215112	-14.607791	-9.481456	-20.949192	4.739582	-11.924955	-1.501
497	1.276599	0.108157	0.167924	0.319859	-0.107824	-0.219104	-0.127548	0.051338	-0.006860	0.134073	0.573376	0.254629	-0.622

We can see it saved the csv file to the by checking the project directory:



But to confirm it actually loads up let's close our browser tab with the web app loaded in it and then launch the app again:



V1

	Unnamed: 0	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
0	492	-0.351227	-2.007712	1.272504	-1.593349	-1.838209	-0.199473	-0.848578	0.077260	-1.439196	1.082354	-1.322466	-1.631
1	493	-1.440210	-0.680241	1.799630	-1.140967	0.117049	0.017536	1.028007	-0.025693	-0.047942	-0.868506	0.313694	-0.119

Success!

Chapter 20: Managing the tested dataset. - (Objective 4*)

The aim of this chapter is to create functions to let the interface user permanently change their decision of the fraud classification predicted by the model.

I am planning to have a button containing actions running alongside the table printed in our dataset. I mention in our design section that we plan to only show transactions with a probability close in value to the decision boundary (0.5). This is because we will be printing too many rows and this does not abstract the records that have importance to the fraud analyst. In our tests, the fact that we had to use two large screenshots to capture the entire width of the screen shows that we simply have too much data. I therefore think we should cut this down by removing columns V0-V28 from our initial display and give the fraud analyst a window to view these attributes in better detail.

A pop up window might be suitable. The remaining columns to be shown will be the transaction ID, the predicted fraud class and predicted fraud probability. The actual fraud classification should not be shown since the fraud analyst nor the program itself would actually have access to this information.

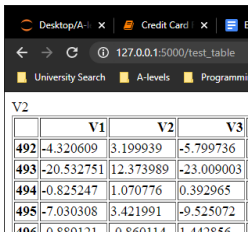
The development process will be simplified if the backend code is ready to be connected to the frontend. Therefore we are going to create this backend code first.

Chapter 21: Managing user decisions - (Objective 4*)

I need all records to have their original unique id/index assigned to them on the transactions dataframe so we can access their original values.

Remove the previous indexing system

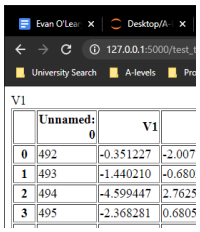
The first order of business is to remove the previous index. After looking at the screenshot of the tested set being displayed through the web app. I notice that an index has been added on the leftmost column. It doesn't show a column name which confused me and additionally I didn't recall ever having set an index and was wondering why it added this. See below.



	V1	V2	V3
492	-4.320609	3.199939	-5.799736
493	-20.532751	12.373989	-23.009003
494	-0.825247	1.070776	0.392965
495	-7.030308	3.421991	-9.525072

Furthermore, this indexing was the source of my original confusion when inferred that the test set wasn't half of the dataset after seeing the last row index was 983 when this was a result of taking a slice from 492 (the midpoint) and 893 (the last row).

After comparing the two previous screenshots from chapter 21 I notice that when we reloaded the web app it added another index column.



Therefore it must be the code which reloads it and creates/handles the csv conversion. After looking at the pandas documentation at the to_csv method:

pandas.DataFrame.to_csv

DataFrame.to_csv(*path_or_buf=None, sep=',', na_rep='', float_format=None, columns=None, header=True, index=True, index_label=None, mode='w', encoding=None, compression='infer', quoting=None, quotechar='"', line_terminator=None, chunksize=None, date_format=None, doublequote=True, escapechar=None, decimal='.', errors='strict'*) ¶ [source]

Write object to a comma-separated values (csv) file.

I see an argument called index which has true as it's default value. This is why we are getting new duplicate index columns every time our web app is reloaded. So we'll set this argument value to false.

```
# Save to csv to prevent future wasted time
testedSet.to_csv("dataset/tested_set.csv", index=False)
```

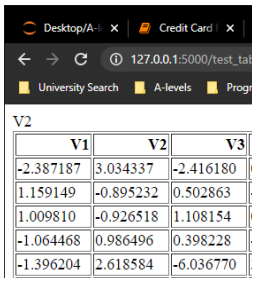
This stopped this continuous column duplication error. However it still showed an index from 0. And as I checked the CSV file and the index was not there anymore. So I wondered whether this was a result from the to_HTML pandas method or an automatic flask setting.

pandas.DataFrame.to_html

DataFrame.to_html(*buf=None, columns=None, col_space=None, header=True, index=True, na_rep='NaN', formatters=None, float_format=None, sparsify=None, index_names=True, justify=None, max_rows=None, max_cols=None, show_dimensions=False, decimal='.', bold_rows=True, classes=None, escape=True, notebook=False, border=None, table_id=None, render_links=False, encoding=None*) [source]

Render a DataFrame as an HTML table.

Again the same index argument is by default set to true. This might become useful later but for now I just want a universal index on the transactions database so I'm going to set this to false in the to_html method.



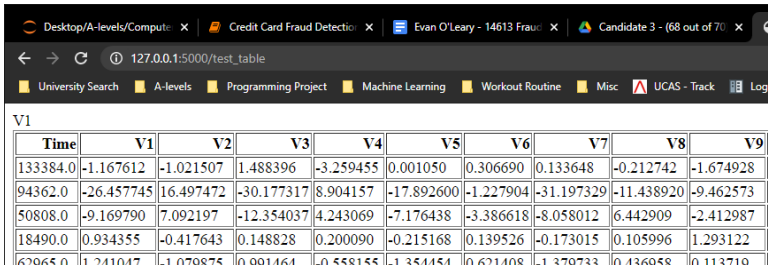
V2

V1	V2	V3
-2.387187	3.034337	-2.416180
1.159149	-0.895232	0.502863
1.009810	-0.926518	1.108154
-1.064468	0.986496	0.398228
-1.396204	2.618584	-6.036770

Success! So now lets universally apply this index to the transactions dataframe.

```
else:
    # doesnt exist so train/test the model
    transactions = pd.read_csv("dataset/creditcard.csv").set_index()
    transactions["V0"] = 1 # assign the 0th feature
```

Lets delete the testedCSV file and run the web app so we can activate this else clause. I also deleted the line in our balance the dataset code which dropped the time and amount columns.



V1

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
133384.0	-1.167612	-1.021507	1.488396	-3.259455	0.001050	0.306690	0.133648	-0.212742	-1.674928
94362.0	-26.457745	16.497472	-30.177317	8.904157	-17.892600	-1.227904	-31.197329	-11.438920	-9.462573
50808.0	-9.169790	7.092197	-12.354037	4.243069	-7.176438	-3.386618	-8.058012	6.442909	-2.412987
18490.0	0.934355	-0.417643	0.148828	0.200090	-0.215168	0.139526	-0.173015	0.105996	1.293122
62965.0	1.241047	-1.079875	0.991464	-0.558155	-1.354454	0.621408	-1.379733	0.436958	0.113719

Seemed to work.

The unique identifier hashing function (Primary Key)

The transactions dataframe is sliced and changed frequently throughout the runtime of our program but there is no index or ID which carries forward after it is segmented into new subsets. In the banking industry, transactions are given unique identifiers. Also if the fraud analyst needed to refer to a transaction and take further action with it, a unique identifier should be used. I am not going to use an integer index since there is no way we can order the transactions as most of the data is anonymised and time is not unique since more than one transaction can occur within the same second.

I am going to create a function which adds an identifier to every transaction record in the database. This way when the HTML record is interacted with, a function can be called from it and changes can be made directly to its attributes in the database.

I decided to use a hashing function to create this. The add unique identifier procedure takes in the names of the columns whose values the hash function should use to generate it's hash total

out of. It should add this ID column to a passed data frame. Here's the code for the first iteration of this function:

```
def addID(df, hashColumns):
    df['ID'] = df[hashColumns].sum(axis=1).map(hash)
```

The sum method takes a sum of the column values in that row. The map method applies this to all the rows but replaces it's sum with the output of the function passed as its argument. In our case we passed the python inbuilt hash function since we want to hash this value. I pass the procedure, the columnNames it should take values from to generate the hash amd apply this procedure to the transactions dataframe as soon as it's created:

```
else:
    # doesnt exist so train/test the model
    transactions = pd.read_csv("dataset/creditcard.csv")#.reset_
    transactions["V0"] = 1 # assign the 0th feature
    addID(transactions, ['Time', 'V1'])
```

Since I've called this function on the entire dataset, any subset will retain this ID and it should therefore be printed when we display the tested dataset:

Success! (Look at the highlighted text since border has not been added again since the new table printing methodology)

However I assumed the hash function would produce an int and wouldn't be so many digits long. Therefore to compress it into a shorted ID I am going to make it a hexadecimal number and use a lambda function in our map argument.

```
def addID(df, hashColumns):
    df['ID'] = df[hashColumns].sum(axis=1).map(lambda val: hex(hash(val)))
```

And delete the tested_set csv again and reload:

V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class
0.025587	0.136574	215.41	1	0x7c3269c0	0.162353	0
0.010164	-0.009007	200.00	1	0x7df972b9	0.065675	0
-1.865831	-0.442204	45.48	1	0x6e75cdfa	0.996732	1
-0.144774	-0.027184	0.77	1	0x693dc920	0.130900	0

Success! However I recall hexadecimal values only include digits 0-9 and characters A-F so am confused as to why every ID seems to start with '0x'. But I check the python documentation for the function which states:

Python `hex()` function takes one parameter.

- **integer** (required) – integer to be converted into the hexadecimal string.

The returned hexadecimal is prefixed with `0x`. Example, `0x3e2`.

I decided to remove this prefix due to not being unique and therefore redundant to our aim.

```
(axis=1).map(lambda val: hex(hash(val))[2:])
```

V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class
1.510206	-0.324706	1354.25	1	7bc10ba6	0.966427	1
-0.072319	0.119695	45.90	1	6dc36b6f	0.262888	0
1.101923	0.205958	1.52	1	21337686	0.999993	1
0.760542	0.386742	0.77	1	7907d466	0.999991	1
-0.918888	0.001454	60.00	1	7801e2fc	0.978822	1

We'll be finished once we can verify that there are no duplicate ID values.

So let's do this from inside the function. We'll use 'duplicated' method to confirm there are no duplicate ID values and add a selection statement to print a warning message if there are duplicate values:

```
def addID(df, hashColumns):
    IDCol = df[hashColumns].sum(axis=1).map(lambda val: hex(hash(val))[2:])
    if IDCol.duplicated().any():
        print("Duplicate ID Values exist.")
        df['ID'] = IDCol
    else:
        print("No duplicate ID Values exist.")
```

After running. The following statement was printed to the terminal:

```
 * Detected change in 'c:\\Users\\User\\Desktop\\A-levels\\Computer Science\\Unit 3 - Programming Project\\
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 776-070-565
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Duplicate ID Values exist.
492
127.0.0.1 - - [18/Dec/2020 21:59:36] "GET /test_table HTTP/1.1" 200 -
```

Duplicates exist. I'm going to check how many there were:

```
21
22  def addID(df, hashColumns):
23      IDCol = df[hashColumns].sum(axis=1).map(lambda val: hex(hash(val))[2:])
24      print(df[IDCol.duplicated()])
25
26
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 9 2: Python

```
33          26.0 -0.529912  0.873892  1.347247  0.145457  ... -0.073288  0.023307    6.14      0   1
35          26.0 -0.535388  0.865268  1.351076  0.147575  ... -0.078306  0.025427    1.77      0   1
113         74.0  1.038370  0.127486  0.184456  1.109950  ...  0.081247  0.001192    1.18      0   1
114         74.0  1.038370  0.127486  0.184456  1.109950  ...  0.081247  0.001192    1.18      0   1
115         74.0  1.038370  0.127486  0.184456  1.109950  ...  0.081247  0.001192    1.18      0   1
...          ...       ...       ...       ...       ...  ...       ...       ...     ...    ...  ..
282987  171288.0  1.912550 -0.455240 -1.750654  0.454324  ...  0.073133 -0.036020   11.99      0   1
283483  171627.0 -1.464380  1.368119  0.815992 -0.601282  ...  0.213565  0.119251    6.82      0   1
283485  171627.0 -1.457978  1.378203  0.811515 -0.603760  ...  0.219432  0.116772   11.93      0   1
284191  172233.0 -2.667936  3.160505 -3.355984  1.007845  ... -1.183743 -0.222200   55.66      0   1
284193  172233.0 -2.691642  3.123168 -3.339407  1.017018  ... -1.205466 -0.213020   36.74      0   1

[1101 rows x 32 columns]
492
127.0.0.1 - - [18/Dec/2020 22:07:38] "GET /test_table HTTP/1.1" 200 -
```

1101 out of the 280 thousand or so transactions were given duplicated IDs. By skimming the small section of shown records, you can see that the field values of duplicates are approximately the same values. This means that our algorithm is doing what I'm instructing it to without error. However I overestimated the sensitivity of Python's hash function. Similar values produce similar hashes and lead to collisions. Also I only gave it two fields to generate this value of, so let's try giving it every field containing continuous values. V1-V28, Time and Amount. This change is highlighted in the code screenshot.

```
36      else:
37          # doesnt exist so train/test the model
38          transactions = pd.read_csv("dataset/creditcard.csv") # Import the large datset locally
39          transactions["V0"] = 1 # assign the 0th feature
40          addID(transactions, transactions.columns.values[:-1])
41
42          dataset = model.balanceDataset(transactions) # balance/reduce dataset
43          features = ['V0', 'V14'] # define some test model
44
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS 9 2: Python

```
[1099 rows x 32 columns]
492
127.0.0.1 - - [18/Dec/2020 22:18:56] "GET /test_table HTTP/1.1" 200 -
 * Detected change in 'c:\\Users\\User\\Desktop\\A-levels\\Computer Science\\Unit 3 - Programming Project\\Web App\\app.py', reload
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 776-070-565
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
          Time        V1        V2        V3        V4        V5        V6  ...       V25       V26       V27       V28  Amount
33         26.0 -0.529912  0.873892  1.347247  0.145457  0.414209  0.100223  ...  0.098816 -0.552904 -0.073288  0.023307    6.14
35         26.0 -0.535388  0.865268  1.351076  0.147575  0.433680  0.086983  ...  0.098117 -0.553471 -0.078306  0.025427    1.77
113        74.0  1.038370  0.127486  0.184456  1.109950  0.441699  0.945283  ...  0.479120 -0.166937  0.081247  0.001192    1.18
114        74.0  1.038370  0.127486  0.184456  1.109950  0.441699  0.945283  ...  0.479120 -0.166937  0.081247  0.001192    1.18
115        74.0  1.038370  0.127486  0.184456  1.109950  0.441699  0.945283  ...  0.479120 -0.166937  0.081247  0.001192    1.18
...         ...       ...       ...       ...       ...       ...       ...  ...       ...       ...       ...       ...     ...
282987  171288.0  1.912550 -0.455240 -1.750654  0.454324  2.089130  4.160019  ... -0.283548 -1.084843  0.073133 -0.036020   11.99
283483  171627.0 -1.464380  1.368119  0.815992 -0.601282 -0.689115 -0.487154  ...  0.044127  0.639270  0.213565  0.119251    6.82
283485  171627.0 -1.457978  1.378203  0.811515 -0.603760 -0.711883 -0.471672  ...  0.044944  0.639933  0.219432  0.116772   11.93
284191  172233.0 -2.667936  3.160505 -3.355984  1.007845 -0.377397 -0.109730  ...  0.086719 -0.451128 -1.183743 -0.222200   55.66
284193  172233.0 -2.691642  3.123168 -3.339407  1.017018 -0.293095 -0.167054  ...  0.083693 -0.453584 -1.205466 -0.213020   36.74

[1099 rows x 32 columns]
492
127.0.0.1 - - [18/Dec/2020 22:19:42] "GET /test_table HTTP/1.1" 200 -
```

Okay the variation given was so small it only separated out two less transactions. So now is the time to use a completely unique method. This wouldn't be such a problem if the dataset wasn't so large and transactions are bound to be duplicate seeming through no error.

Now I'm going to number the transactions 1 to n. And assign this index as it's ID. This way we can use the index to directly access the original record data from the transactions csv.

```
def addID(df):
    df['ID'] = list(range(1, len(df) + 1))
```

V26	V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class
0.056643	-0.092482	-0.060673	14.95	1	154204	0.065879	0
0.172643	0.726781	0.234514	723.21	1	215133	0.997989	1
-0.345133	0.028813	0.013123	41.50	1	40792	0.076800	0
-0.220080	0.392338	-0.020089	22.04	1	154634	0.772499	1

Success! No possibility for duplicate IDs as collisions aren't possible since it is range function and not a hashing function. We might come back and change the value to a hex value to reduce the number of digits (space) while storing. This function shall only be called once and is a single line so I'm going to substitute it back into app.py and replace the df variable with the transactions variable.

Creating the data frame of fraud analyst decided Values

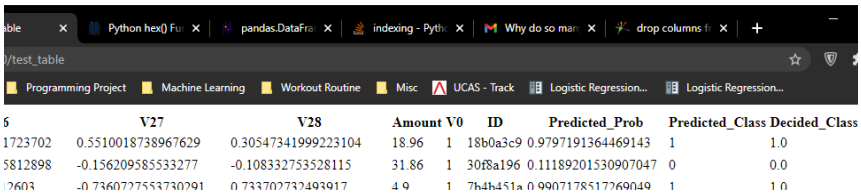
First I am going to copy the tested set into a new dataframe named finalClassifications. Then I am going to drop the column for the actual fraud classification since the fraud analyt's database would not have access. Next I am going to add a new column named Decided_Class by duplicating the predicted class column but if a transaction has a predicted probability of less than 0.05 away from the decision boundary (0.50), I will change it's Decided_Class value to None.

```
def createDecisionDB(testedSet):
    decisionDB = testedSet.copy()
    # Drop the column for the actual fraud classification
    decisionDB.drop('Class')
    # Add a new column named Decided_Class by duplicating the predicted class column
    decisionDB['Decidid_Class'] = decisionDB['Predicted_Class']
    # If a transaction has a predicted probability of less than 0.05 away from the decision boundary (0.50):
    for index, row in decisionDB.iterrows():
        if (decisionDB.loc[index,'Decided_Class'] - 0.50 < 0.05):
            decisionDB.at[index,'Decided_Class'] = None  # change it's Decided_Class value to None.
    return decisionDB
```

Now let's replace the testedSet for our newly created decision set.

```
else:
    # doesnt exist so train/test the model
    transactions = pd.read_csv("dataset/creditcard.csv") # Import the large datset locally
    transactions["V0"] = 1 # assign the 0th feature
    addID(transactions, ['Time', 'V1'])

    dataset = model.balanceDataset(transactions) # balance/reduce dataset
    features = ['V0', 'V14'] # define some test model

    testedSet = model.trainTest(dataset, features)
    decisionDB = model.createDecisionDB(testedSet) # make the decision database

    # Save to csv to prevent future wasted time
    decisionDB.to_csv("dataset/decisionDB.csv", index=False)
```

Let's test that the Class column has been dropped and Decided_Class has been added.



6	V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class
1723702	0.5510018738967629	0.30547341999223104	18.96	1	18b0a3c9	0.9797191364469143	1	1.0
5812898	-0.156209585533277	-0.1083327535528115	31.86	1	30f8a196	0.11189201530907047	0	0.0
2603	-0.7360727553730291	0.7337027324493917	4.9	1	7b4b451a	0.9907178517269049	1	1.0

Test passed successfully. Finally we'll teste if the Decided_Class value is modified to the value of the Decided_Class to none(nan) if the difference between decision boundary and predicted probability is less than 5% and else the other transactions values remain as their predicted class.

1.0	1	7890b882	0.9999971989087739	1	1.0
6.99	1	24175273	0.5033810003673934	1	nan

Test passed successfully.

The classification human decision function

We are going to create a function which takes the transaction's unique identifier key, and the fraud binary classification value (1 for fraud, 0 for non-fraud).

```
def classify(dataset, transactionID, classification):
    index = dataset.index[dataset['ID'] == transactionID]
    dataset.at[index, "Decided_Class"] = classification
```

This is the dataset we care about displaying. So this is the file I will save and replace it in app.py.

Chapter 22: Formatting the table - (Objective 4*)

Subtasks:

1. Filter the FinalClassifications DataFrame to only the cases which are undecided. (Those with a Decided_Class equal to None)
2. Only print the relevant columns to the fraud analyst.
3. Create a button for each row called classify which dropdowns two values 0 and 1. Pressing the button should trigger the record to disappear from the screen and it's Ddecided_Class value should be updated.
4. Create a button for each row called more information which opens a pop up of all the columns about the records

If I wanted to add a button to each row, I would have to create the table using a new method. The pandas provided to_HTML method pregenerates the table markup and populates it with data from the dataframe.

So I'm going to modify our HTML to print every cell with two loops. One main loop which iterates every row. And inside that one which prints all the cell values and finally the button. See below.

```
<table>
    <thead>
        <tr>
            <!-- Print every heading -->
            {% for header in headings %}
                <th> {{ header }} </th>
            {% endfor %}
        </tr>
    </thead>
    <tbody>
        <!-- Iterate over every row -->
        {% for row in row_data %}
            <tr>
                <!-- Iterate over every cell -->
                {% for cell in row %}
                    <td>
                        {{ cell }}
                    </td>
                {% endfor %}
                <td>
                    <!-- Classify particular row-->
                    <button>Classify</button>
                </td>
            </tr>
        {% endfor %}
    </tbody>
</table>
```

Now we need to modify the function in app.py which renders this html with the passed data frame variables. We need to make the rows a list.

```
row_data = list(testedSet.values.tolist())
```

We also need a separate list for header values (column names).

```
headings = testedSet.columns.values
```

Now these are assigned to variables. These can be accessed by the jinja 2 loops as variables while being rendered:

```
# render table
return render_template('test_table.html',  row_data = list(testedSet.values.tolist()), headings = testedSet.columns.values )
```

Now renders:

V25	V26	V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class	
80690790549	0.0566427032153314	-0.0924816502529371	-0.0606730109581279	14.95	1.0	154204.0	0.06587885252893985	0.0	0.0	Classify
097167708	0.172642921018111	0.7267810122814878	0.23451392249695896	723.21	1.0	215133.0	0.9979893849233422	1.0	1.0	Classify
708221606789	-0.345132759675864	0.0288128362137282	0.0131232129007104	41.5	1.0	40792.0	0.07679967366569435	0.0	0.0	Classify
5629075428	-0.220079647308722	0.392337584118175	-0.0200894870934853	22.04	1.0	154634.0	0.7724987085419271	1.0	1.0	Classify
213718346795	-0.176701082429858	0.504897671691263	0.0698824528615783	39.45	1.0	43625.0	0.9700070977598816	1.0	1.0	Classify

Which looks better without the previous styling given by the pandas function and also removes the random text which was generated at the top left of the page.

Button on every row added successfully.

Linking functionality to the button.

To simplify this process, I am going to click the classify button we just added, class the transaction as fraud (Class = 1). This means we only need to deal with the linking of the action since we can later have two buttons or an option to repeat this functionality for both fraud and non-fraud.

So I'll add extra text with the button to help clarify the above statement during development.

```
<td>
    <!-- Classify particular row-->
    <button>Classify as fraud</button>
</td>
```

So far with hyperlinks, we've been using routing to interact with our application from the browser. However this refreshes and reloads the page. Although we could use a route it would make the page unresponsive and we want the user to have a clean experience. Therefore we are going to be using Asynchronous JavaScript and XML (AJAX) to request data from a web server using a XMLHttpRequest object which is built-in to the browser.

Here's my plan for the code.

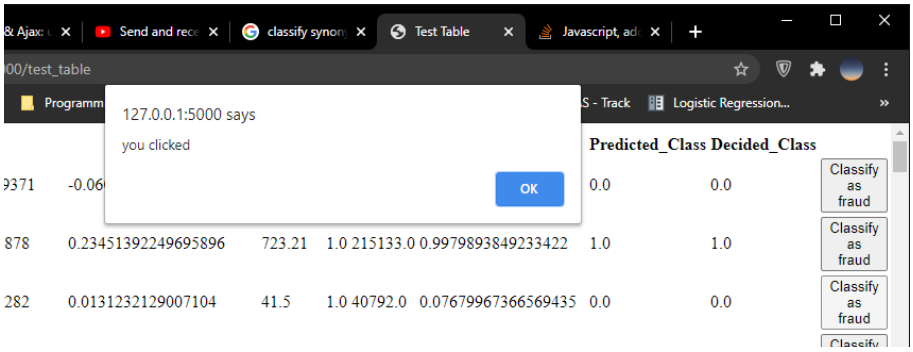
1. Add a script to the bottom of our HTML page.
2. Add an event listener for clicking on the classify button.
3. Make a request to our web server running the flask application. Carry the ID value of the record whose button was clicked.
4. The request should run the classify function in the python script which passes the carried data.
5. Table is updated and so is the decsionDB database.

Server Side verification that valid data is being passed in the Python function is very important since Client side JavaScript code can be amended to bypass checks.

This paragraph describes the code in the screenshot below. We're going to start by adding script tags into the body of our HTML file. I'm choosing to store this javascript internally while the codebase is small. We assign a collection of the button elements to a constant called 'buttons'. Next we loop over these elements and add an event listener which assigns a function to be called once a given action is triggered. In this case we want the function to be triggered when the button is clicked but the action could be hover or something else.

```
<script>
    const buttons = document.getElementsByTagName("button");
    for (i = 0; i < buttons.length; i++) {
        buttons[i].addEventListener("click", function() {
            alert("you clicked");
        });
    };
</script>
```

Now let's test that clicking these buttons activates the function which triggers an alert message.



Clicking each one would produce the above message. Success.

Now we need to use the row adding loop in the HTMl to add the transaction ID to each HTML button element. I'm going to add this by using the data attribute. It lets you create and name a new HTML attribute for elements. This is useful because we can access this attribute value from the element object in the javascript, and use it in our function, in our case, the classify function.

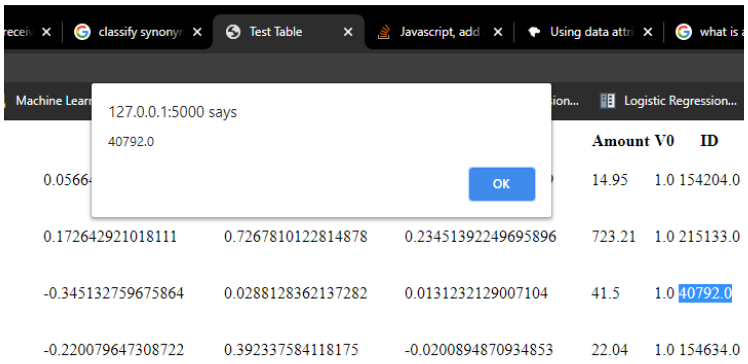
Let's demonstrate this by saving the required user id to an html attribute called 'data-transaction-id' and then accessing it in the event listener's callback function and print it to the screen with an alert.

```
<td>
    <!-- Classify particular row-->
    <button data-transactionid="{{ row[-4] }}">Classify as fraud</button>
</td>
```

The above code shows how we use the jinja 2 syntax to access an individual cell value from the row being looped over. All the rows were turned into lists in app.py before being passed to render, meaning we can use indexing to access a particular cell value. The ID column is in the fourth last index, so we use this index to access the cell value from the row and insert it into the attribute value.

```
<script>
    const buttons = document.getElementsByTagName("button");
    for (i = 0; i < buttons.length; i++) {
        buttons[i].addEventListener("click", function(e) {
            alert(e.target.dataset.transactionid);
        });
    };
</script>
```

The javascript above shows how we access this attribute value. In our callback function, our argument 'e' stands for event and is an object for representing the event of clicking on the button. The attribute called 'target' is the target element that the action "click" was made to. So from there we can access the 'data-transaction-id' value. Lets see if this works when we click any of the buttons:



Success!

The user classification function

Now we have assigned the unique IDS and are ready to create the function which lets the user decide the classification of a transaction, we must create and then independently test it locally in jupyter before connecting it to the flask app. Code is below.

```
# Classify case - responding to input
def classify(dataset, transactionID, classification):

    # get the index of the selected transaction in the wider dataset
    index = dataset.index[dataset['ID'] == transactionID]

    # now modify the class in this wider dataset
    dataset.at[index, "Decided_Class"] = classification

dataset["Decided_Class"] = None
```

We add the Decided_Class column to the balance dataset since it is similar to the DecisionDB dataset in our web application. This is what we'll be testing on:

```
# get an example to test on - the first row
exampleTransaction = dataset.iloc[0]

# obtain ID
ID = exampleTransaction.ID

# user decided classification before should be None
print(exampleTransaction["Decided_Class"])
```

```
None
```

```
# classify this example classification as 1
classify(dataset, ID, 1)

# see if it was updated
dataset.iloc[0]["Decided_Class"]
```

```
1
```

Successfully changed the transaction classification from null (none) to fraudulent (1). Now we have confirmed that the classify function works, we need to create the JavaScript code that sends the request to update the decisionDB database with the new classification once the button is clicked.

First we are going to define the url address we want to send the POST request to. And we'll temporarily let the classification be equal to 1 since our button so far only allows to change transaction classification to fraud.

```
const appdir = '/classify'
const server = 'http://127.0.0.1:5000/'
const URLaddress = server + appdir
let classification = 1
```

Next we need to alter the event listener so that it sends a POST request to the server. This is done through the fetch API, a simple interface which makes it easier to make web requests and handle responses. The await keyword before the fetch means this code will only be executed once the request has received a response. This is what asynchronous code is, not running alongside the single threaded execution of the program, since the time to send a request and work with external servers is variable and out of line with the sequential or synchronous running of the program.

In the fetch call, we specify that we are sending a POST request, along with the URL we want to make the request to. We also specify in the headers the type of content we want to send, and also, most importantly, the payload data we are sending for the server to process. This payload data is called the body. I interpreted the structure of the fetch arguments to be an abstraction of a packet hence why I'm referring to the body of data as the payload.

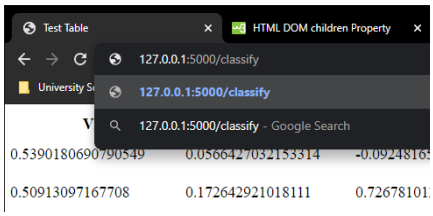
```
const buttons = document.getElementsByTagName("button");
for (i = 0; i < buttons.length; i++) {
    buttons[i].addEventListener("click", async function(e) {
        transactionID = e.target.dataset.transactionid
        data = {
            'transactionID': transactionID,
            'classification': classification
        };
        console.log(data)
        await fetch(
            URLaddress, {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify(data)
            }
        )
    });
};
```

We are going to send the body data in the JSON data format, which is a human-readable text which stores a data object consisting of key–value pairs. The data must be sent as one string, therefore the entire object must be "stringified".

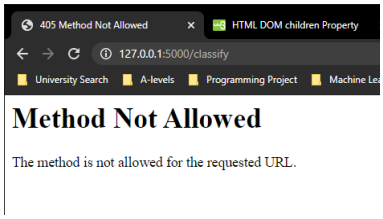
Next we're going to rework the classify function so that it can receive the request, access its data and update the database. First we need to name the methods it works with as a parameter in the route parameter. We only name the POST method so that if someone entered the classify ending as a URL in the search box, it won't try to perform a GET method, but instead show an error.

```
@app.route('/classify', methods = ['POST'])
def classify():
```

Before enter:



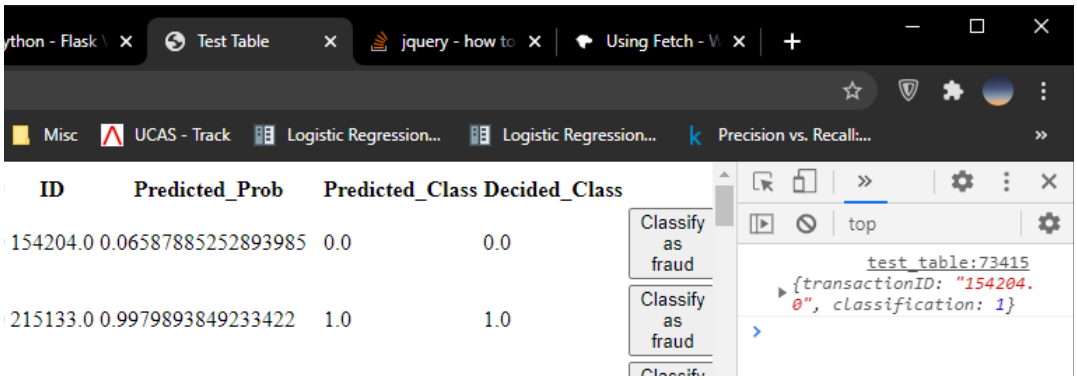
After enter:



Success. Now inside the function we must retrieve the decisionDB we are going to be altering while changing the classification of a given record. We cannot pass the data frame via an argument since the post request comes from the web app which doesn't have access to this information. So that's why we are going to read the decisionDB CSV file, edit the record and save it back as a csv file replacing the old file. This makes the change permanent. We use the request object to access the data in the payload of the POST request sent. The get_json method converts the json data into a dict which we can access the values from using keys. The code for the new classify function is shown below:

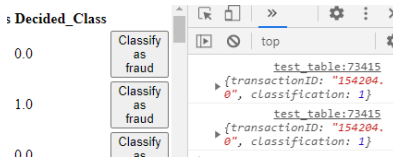
```
# Classify case - responding to input
@app.route('/classify', methods = ['POST'])
def classify():

    # Import the decisionsDB
    decisionDB = pd.read_csv("dataset/decisionDB.csv")

    if request.method == 'POST':

        # GET data from payload stored as JSON
        data = request.get_json()
        print(data)
        transactionID, classification = data['transactionID'], data['classification']

        # Update the classifcation in dataframe
        index = decisionDB.index[decisionDB['ID'] == transactionID]
        decisionDB.at[index, "Decided_Class"] = classification

        # Save to csv to prevent future wasted time
        decisionDB.to_csv("dataset/decisionDB.csv", index=False)

        return 'Transaction classified succesffuly'
```

We then launch app.py and click the classify as fraud button on the first row.



And proceed to check the terminal to see if the data was printed:

```
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 776-070-565
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [19/Dec/2020 20:21:59] "GET /test_table HTTP/1.1" 200 -
{'transactionID': '154204.0', 'classification': 1}
127.0.0.1 - - [19/Dec/2020 20:22:03] "POST //classify HTTP/1.1" 200 -
```



Success. The data was successfully transferred from the client side to the server side as you can see from the printed object with the correct ID and classification.

However when I refresh and even press again, I notice that the Decided_Class value remains at 0.0 As shown by the image on the right. So I inspect the flask code and print the index and indexed transaction since the data has been demonstrated to successfully be transmitted to the backend program.

```
print("transactionID: ", transactionID)

# Update the classifcation in dataframe
index = decisionDB.index[decisionDB['ID'] == transactionID]

print("index: ", index)
print("indexed record: ", decisionDB.iloc[index])
```

And this was the printed result in the terminal for the first record:

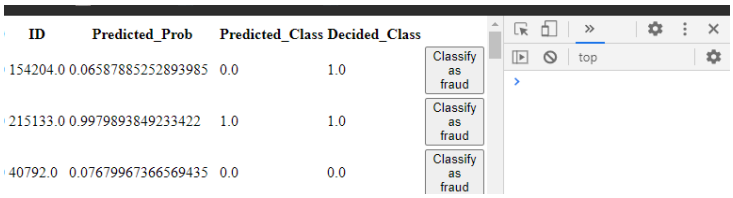
```
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
transactionID:  154204.0
index:  Int64Index([], dtype='int64')
indexed record:  Empty DataFrame
```

Which shows no index was matched with the transactionID, hence the indexed record came up empty. It was then I noticed that the printed transactionID in the terminal was a floating point number ending in .0. This was a pattern followed by all numerical values on the web page and I'm not sure what caused this. For now I'm just going to try and convert the float to an int in the

line where I save the ID to a variable. I'm also going to change from the tuple unpacking method to the standard method.

```
# GET data from payload stored as JSON
data = request.get_json()

transactionID = int(float(data['transactionID']))
classification = data['classification']
```

I then retry and refresh:



Success! Decision Class Value updated from 0 to 1 after refresh.

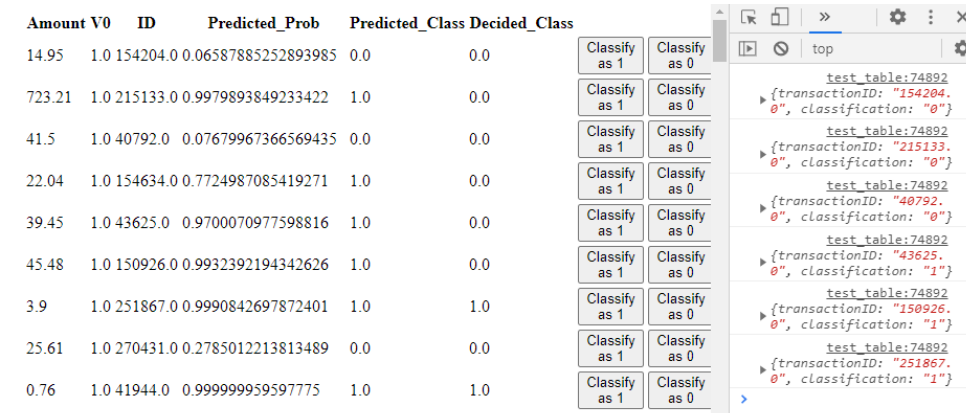
Let's add the option to classify as non-fraud. At the moment we'll simplify to 0 and 1 to represent the classifications in the HTML. We'll store the classifications in the data attribute of our button. This is shown in the code below.

```
{% for row in row_data %}
    <tr>
        <!-- Iterate over every cell -->
        {% for cell in row %}
            <td>
                {{ cell }}
            </td>
        {% endfor %}
        <td>
            <!-- Classify particular row-->
            <button data-transactionid = "{{ row[-4] }}" data-classification = 1>Classify as 1</button>
        </td>
        <td>
            <button data-transactionid = "{{ row[-4] }}" data-classification = 0>Classify as 0</button>
        </td>
    </tr>
{% endfor %}
```

In the code below, we represent this change and dynamically set the classification variable from the target elements classification variable. Adding this new button to each row won't affect its function since all are targeted by the same selection condition, by tag name: button.

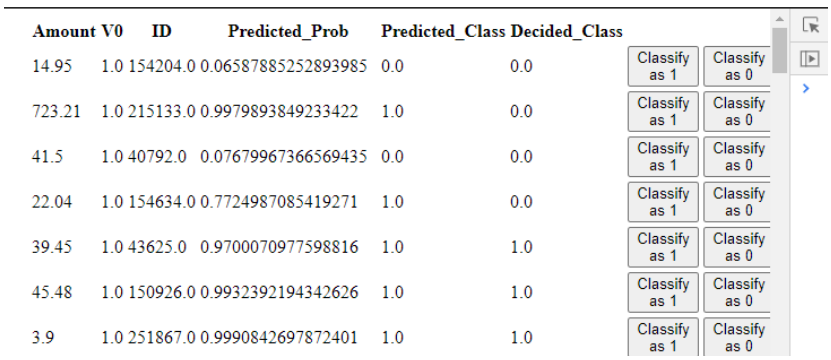
```
const buttons = document.getElementsByTagName("button");
for (i = 0; i < buttons.length; i++) {
    buttons[i].addEventListener("click", async function(e) {
        let transactionID = e.target.dataset.transactionid
        let classification = e.target.dataset.classification
        let data = {
            'transactionID': transactionID,
            'classification': classification
        };
        console.log(data)
        await fetch(
            URLaddress, {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify(data)
            }
        )
    });
};
```

Let's test this actually did work. I'm going to set the first three rows to non fraud (0) and the next three to fraud (1).



Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class		
14.95	1.0	154204.0	0.06587885252893985	0.0	0.0	Classify as 1	Classify as 0
723.21	1.0	215133.0	0.9979893849233422	1.0	0.0	Classify as 1	Classify as 0
41.5	1.0	40792.0	0.07679967366569435	0.0	0.0	Classify as 1	Classify as 0
22.04	1.0	154634.0	0.7724987085419271	1.0	0.0	Classify as 1	Classify as 0
39.45	1.0	43625.0	0.9700070977598816	1.0	0.0	Classify as 1	Classify as 0
45.48	1.0	150926.0	0.9932392194342626	1.0	0.0	Classify as 1	Classify as 0
3.9	1.0	251867.0	0.9990842697872401	1.0	1.0	Classify as 1	Classify as 0
25.61	1.0	270431.0	0.2785012213813489	0.0	0.0	Classify as 1	Classify as 0
0.76	1.0	41944.0	0.999999959597775	1.0	1.0	Classify as 1	Classify as 0

After refresh:



Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class		
14.95	1.0	154204.0	0.06587885252893985	0.0	0.0	Classify as 1	Classify as 0
723.21	1.0	215133.0	0.9979893849233422	1.0	0.0	Classify as 1	Classify as 0
41.5	1.0	40792.0	0.07679967366569435	0.0	0.0	Classify as 1	Classify as 0
22.04	1.0	154634.0	0.7724987085419271	1.0	0.0	Classify as 1	Classify as 0
39.45	1.0	43625.0	0.9700070977598816	1.0	1.0	Classify as 1	Classify as 0
45.48	1.0	150926.0	0.9932392194342626	1.0	1.0	Classify as 1	Classify as 0
3.9	1.0	251867.0	0.9990842697872401	1.0	1.0	Classify as 1	Classify as 0

Success! So the next step will be to make this automatically update on the front-end without having to refresh the page to load from the updated csv file.

Before moving on, I wanted to mention that I removed the selection statement in the classify function which allows the code to run if the request is a POST method. This was redundant since we showed that only POST methods were allowed anyway. Here's the code for the final version:

```python
# Classify case - responding to input
@app.route('/classify', methods = ['POST'])
def classify():

    # Import the decisionsDB
    decisionDB = pd.read_csv("dataset/decisionDB.csv")

    # GET data from payload stored as JSON
    data = request.get_json()

    transactionID = int(float(data['transactionID']))
    classification = data['classification']

    print("transactionID: ", transactionID)

    # Update the classifcation in dataframe
    index = decisionDB.index[decisionDB['ID'] == transactionID]

    print("index: ", index)
    print("indexed record: ", decisionDB.iloc[index])

    decisionDB.at[index, "Decided_Class"] = classification

    # Save to csv to prevent future wasted time
    decisionDB.to_csv("dataset/decisionDB.csv", index=False)

    return f'Success. Transaction with ID {transactionID } has been classified as {classification}.'
```

Update record on screen without refreshing

First we need to get the index of the column which holds the Decision Class values. The code for this is below:

```javascript
// iterate through header cell values find index of the DecisionValues
headerCellEls = document.querySelectorAll('th')
for (i = 0; i < headerCellEls.length; i++) {
    if (headerCellEls[i].innerText.trim() == 'Decided_Class') {
        decidedClassColIdx = i
        break
    }
}
```

So we select all the table headers and iterate through them incrementing an index 'i' until we find a match for the text being equal to our target. We use the trim method to remove the whitespace, the spaces between cell values in the HTML. Once we find a match, save it to a

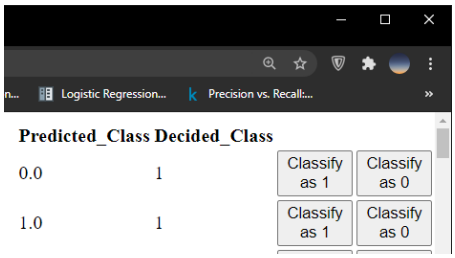
variable. The reason we use this method instead of just finding the value and not just using an absolute known index that we find, is that this method will hold even if we reduce the number of columns shown in the HTML table.

Next we want to use a selection statement to verify that we will only update the HTML if the status was actually successful. The way I can check that is by saving the output of the prior fetch to a variable called response and checking it's status property. We physically update the row by getting the targeted clicked button element's parent element which is the row containing it, getting its children cell values and indexing the one which contains the Decided_Class value with the Decided Class Column Index we obtained earlier. We change the innerText of this element to the classification that's been clicked.

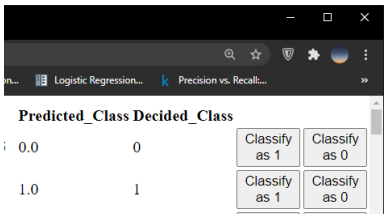
The code which is described by this is shown below:

```
// Send a post request to update database
const response = await fetch(
    URLaddress, {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify(data)
    }
)
// if request was succesful
if (response.status == 200) {

    // Physically update row
    const currCellEl = e.target.parentElement
    const currRowCellsEls = currCellEl.parentElement.children
    currRowCellsEls[decidedClassColIdx].innerText = e.target.dataset.classification

// if reuqest was unsuccessful
} else {
    alert("Couldn't update database");
}
```

Now let's test if clicking on the classify button dynamically changes the row HTML without needing to refresh the page:



Immediately after classifying first row as 0 and second row as 1:



Success!

Altering content displayed

We need to cut down the amount of columns we're showing. The purpose of the dashboard in our app is to show undecided fraud cases, so lets filter the content to display down to these rows.

We'll create a new variable called undecidedDB which will be a copy of the decisionDB data frame but only the subset of rows which have a DecisionClass equal to None. Once we formulated the decisionDB data frame, setting undecided rows Decision_Class value to None actually converted it to the Not a Number (NaN) type. There's a special series method called 'isna' which returns the indexes of all records whose column value is equal to NaN. We use this to select these particular rows.

The code below shows these changes to html_table in app.py:

```
# create the subset we are going to show
undecidedDB = decisionDB.loc[decisionDB['Decided_Class'].isna()]

# render table
return render_template('test_table.html',  row_data = list(undecidedDB.values.tolist()), headings = undecidedDB.columns.values)
```

We make these changes outside and below the decision statement which checks if the decisionDB has been created. Therefore we need not make any verification checks to see if saved data has been altered. This new data frame is purely to be viewed and to be used to copy transaction IDs to make changes on the wider data frame.

Here is the webpage after refreshing:

V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class		
-0.0360399641165051	0.016520010234727	1.0	1.0	281462.0	0.4904525325576094	0.0	nan	Classify as 1	Classify as 0
-0.015551488169813001	0.0418807020330689	276.17	1.0	192688.0	0.5465414276358201	1.0	nan	Classify as 1	Classify as 0
-0.3495805942313161	-0.0251710847123494	0.76	1.0	176703.0	0.4534143819759824	0.0	nan	Classify as 1	Classify as 0
0.0201823653167864	-0.0154697051692816	19.95	1.0	276072.0	0.4623945490244886	0.0	nan	Classify as 1	Classify as 0
0.0215269254860304	0.109191896780292	187.11	1.0	254345.0	0.4601100439416544	0.0	nan	Classify as 1	Classify as 0
0.0270429343018625	0.0632375727147923	0.0	1.0	93789.0	0.4774080452147546	0.0	nan	Classify as 1	Classify as 0
-0.13624338219167198	-0.00985225632281393	996.27	1.0	233259.0	0.4578741180703218	0.0	nan	Classify as 1	Classify as 0
0.7505305780403471	-0.532088128739401	8.97	1.0	102743.0	0.5323093707853493	1.0	nan	Classify as 1	Classify as 0

We must also test that after refreshing, rows that have been classified from none to 1 or 0 disappear since they do need to be shown. So we'll classify the first two rows as 1 and the next two as 0 by using the buttons we have just configured.

V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class		
-0.15250156201687698	-0.138866256312262	6.99	1.0	149146.0	0.47710344705361263	0.0	1	Classify as 1	Classify as 0
0.0270429343018625	0.0632375727147923	0.0	1.0	93789.0	0.4770154350448618	0.0	1	Classify as 1	Classify as 0
0.0201823653167864	-0.0154697051692816	19.95	1.0	276072.0	0.4628258007509152	0.0	0	Classify as 1	Classify as 0
0.0029875822434290698	-0.0153088128485981	42.53	1.0	281675.0	0.4946255087745168	0.0	0	Classify as 1	Classify as 0
-0.0334325568132471	-0.0302164791219711	1.0	1.0	259530.0	0.4553937118791329	0.0	nan	Classify as 1	Classify as 0
-0.0195787926354923	0.0061547052319409 79	108.51	1.0	204080.0	0.4785248495575275	0.0	nan	Classify as 1	Classify as 0
-0.13624338219167198	-0.00985225632281393	996.27	1.0	233259.0	0.45855272329208435	0.0	nan	Classify as 1	Classify as 0
-0.34063914871381895	-0.6977811718104651	350.35	1.0	264523.0	0.5195257500029132	1.0	nan	Classify as 1	Classify as 0

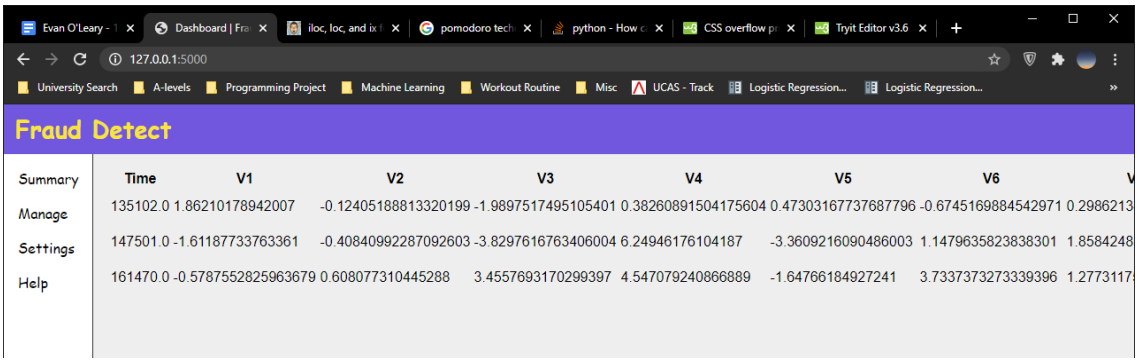
After refresh:

V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class		
-0.0334325568132471	-0.0302164791219711	1.0	1.0	259530.0	0.4553937118791329	0.0	nan	Classify as 1	Classify as 0
-0.0195787926354923	0.0061547052319409785	108.51	1.0	204080.0	0.4785248495575275	0.0	nan	Classify as 1	Classify as 0
-0.13624338219167198	-0.00985225632281393	996.27	1.0	233259.0	0.4585527232920844	0.0	nan	Classify as 1	Classify as 0
-0.3406391487138189	-0.6977811718104651	350.35	1.0	264523.0	0.5195257500029132	1.0	nan	Classify as 1	Classify as 0

Success. Brilliant. This is a major milestone in our project.

Chapter 23: Configuring the dashboard - (Objective 4*)

We transported this Python code for our test table view into the index view, essentially making this the home page (dashboard), like how we planned in the Screen Design section of our design phase. Also we'll need to move the HTML for this into index.htm then move this javascript into a new file called script.js in the static folder.

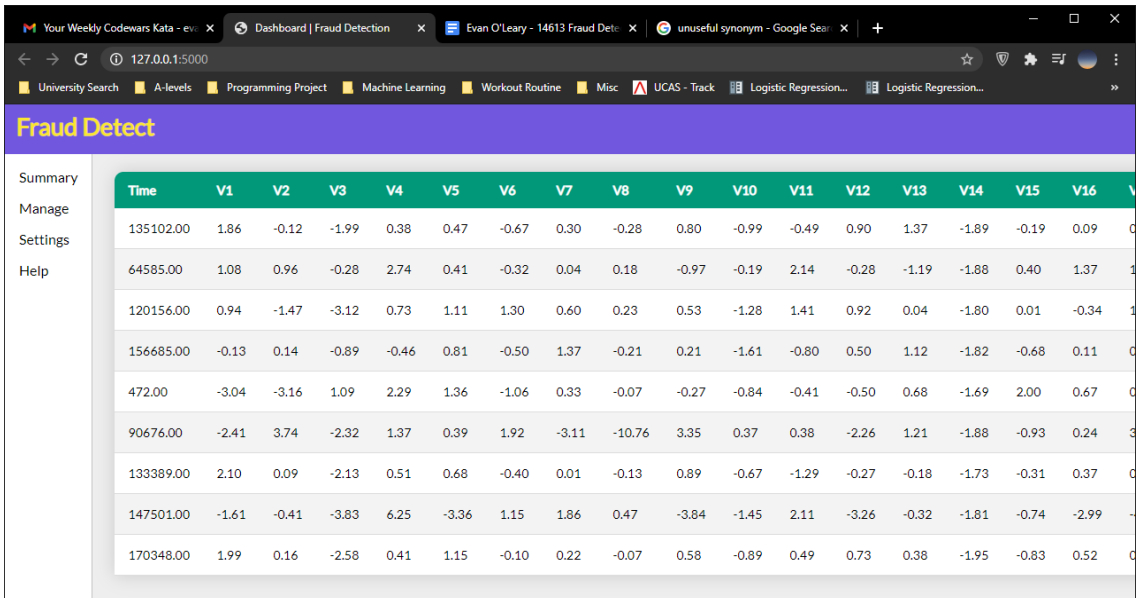


We run into a problem we could have easily predicted. There's too many columns to display, and the actual values are shown to an unhelpfully high degree of accuracy.

Let's make the numerical cell values closer to 2.dp. Jinja 2 has an inbuilt format function to help accomplish this.

```
<!-- Iterate over every cell -->
{% for cell in row %}
    <td>
        {{ '%0.2f'| format(cell|float) }}
    </td>
{% endfor %}
```

I am also going to add styling throughout to help the readability and professionalism of our web app. I won't go into detail. Here's the page after the formatting change:



Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
135102.00	1.86	-0.12	-1.99	0.38	0.47	-0.67	0.30	-0.28	0.80	-0.99	-0.49	0.90	1.37	-1.89	-0.19	0.09
64585.00	1.08	0.96	-0.28	2.74	0.41	-0.32	0.04	0.18	-0.97	-0.19	2.14	-0.28	-1.19	-1.88	0.40	1.37
120156.00	0.94	-1.47	-3.12	0.73	1.11	1.30	0.60	0.23	0.53	-1.28	1.41	0.92	0.04	-1.80	0.01	-0.34
156685.00	-0.13	0.14	-0.89	-0.46	0.81	-0.50	1.37	-0.21	0.21	-1.61	-0.80	0.50	1.12	-1.82	-0.68	0.11
472.00	-3.04	-3.16	1.09	2.29	1.36	-1.06	0.33	-0.07	-0.27	-0.84	-0.41	-0.50	0.68	-1.69	2.00	0.67
90676.00	-2.41	3.74	-2.32	1.37	0.39	1.92	-3.11	-10.76	3.35	0.37	0.38	-2.26	1.21	-1.88	-0.93	0.24
133389.00	2.10	0.09	-2.13	0.51	0.68	-0.40	0.01	-0.13	0.89	-0.67	-1.29	-0.27	-0.18	-1.73	-0.31	0.37
147501.00	-1.61	-0.41	-3.83	6.25	-3.36	1.15	1.86	0.47	-3.84	-1.45	2.11	-3.26	-0.32	-1.81	-0.74	-2.99
170348.00	1.99	0.16	-2.58	0.41	1.15	-0.10	0.22	-0.07	0.58	-0.89	0.49	0.73	0.38	-1.95	-0.83	0.52

First we should order the rows ID which is essentially sorting by time in ascending order (oldest first). This timestamp may not actually be as useful to a fraud analyst.

Chapter 24: Adding a time and date column - (Objective 4*)

A timestamp is a sequence of characters or encoded information identifying when a certain event occurred, usually giving date and time of day, sometimes accurate to a small fraction of a second. We'll need to use the time module in python to convert our timestamp into a time in the standard form: the number of milliseconds from the Unix Epoch: the point in time from which computers use as the absolute zero value for time 00:00:00 1st of January 1970. The time column of our dataset contains the number of seconds elapsed between this transaction and the first transaction in the dataset.

The following description explains the code below for our function which returns a list of timedate strings. The function takes in a dataframe's Time column values in the form of a list. To create a standard timestamp, I obtained the timestamp of the start of the dataset. Kaggle says it was recorded in September 2013, I am going to say it began on the 1st. Then I added all the times in our dataset to the starting time value and proceeded to convert it back into a datetime object. I appended the stringified version of the object to our datetimes list. A string of a timedate object returns the date and time in the ISO format. This list is returned at the end of subprograms runtime.

```python
def generateDatetimes(timesList):

    # get starting point in time of dataset
    startingDate = "01/09/2013"
    startTimestamp = time.mktime(datetime.strptime(startingDate, "%d/%m/%Y").timetuple())

    datetimes = []
    for timestamp in timesList:

        # use a list comprehension to create a list of datetimes
        datetimeObj = datetime.fromtimestamp( startTimestamp + timestamp)
        datetimes.append(str(datetimeObj))

    return datetimes
```

I assign this datetimes list to a new column in undecidedDB, the dataframe we are going to display on our web page with all of our undecided cases. I add it to this data frame because it is the only one the fraud analyst is able to interact with so far.

```python
# add datetimes
undecidedDB['Datetime'] = model.generateDatetimes(list(undecidedDB['Time']))
```

Lets test if they were displayed correctly.

V22	V23	V24	V25	V26	V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class	Datetime	Classify
-0.51	0.08	0.39	0.01	-0.12	-0.02	0.01	108.51	1.00	204080.00	0.53	1.00	nan	0.00	1 0
-0.06	-0.05	-0.03	0.40	0.07	0.03	0.06	0.00	1.00	93789.00	0.53	1.00	nan	0.00	1 0
-0.18	-0.36	-1.09	-0.13	-0.27	-0.04	0.06	510.36	1.00	170389.00	0.51	1.00	nan	0.00	1 0
-0.25	0.48	0.36	-0.44	-0.25	0.02	0.11	187.11	1.00	254345.00	0.51	1.00	nan	0.00	1 0
0.44	1.38	-0.29	0.28	-0.15	-0.25	0.04	529.00	1.00	624.00	0.48	0.00	nan	0.00	1 0

Test failed. Let's check the terminal to see if the datetimes printed to it correctly:

```
 * Debugger is active!
 * Debugger PIN: 776-070-565
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
['2013-09-02 13:31:42', '2013-09-01 17:56:25', '2013-09-02 09:22:36', '2013-09-02 19:31:25', '2013-09-01 00:07:52', '2013-09-02 01:11:16', '2013-0
9-02 13:03:09', '2013-09-02 16:58:21', '2013-09-02 23:19:08']
```

Successfully created datetimes list. I suspect it is a formatting error in the HTML Jinja 2 where we rounded every value to 2.d.p in the previous chapter. So I comment this out and temporarily switch back to our first method of displaying cell values:

```
<!-- Iterate over every cell -->
{% for cell in row %}
    <td>
        {{ cell }}
        <!-- {{ '%0.2f'| format(cell|float) }} -->
    </td>
{% endfor %}
```

Let's retest our web page to see if the datetime strings appear instead of 0.00:

	V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class	Datetime	Classify
01	-0.0195787926354923	0.0061547052319409785	108.51	1	204080	0.5310976267210572	1	nan	2011-12-02 13:31:42	1 0
5	0.0270429343018625	0.0632375727147923	0.0	1	93789	0.5294395476238009	1	nan	2011-12-01 17:56:25	1 0
	-0.0357845238420802	0.06073434717573201	510.36	1	170389	0.5058439578929184	1	nan	2011-12-02 09:22:36	1 0
1	0.0215269254860304	0.109191896780292	187.11	1	254345	0.5114101210746065	1	nan	2011-12-02 19:31:25	1 0
	-0.252773122530705	0.0357642251788156	529.0	1	624	0.476275728044045	0	nan	2011-12-01 00:07:52	1 0

Success (with compromise)! Now we must selectively decide to only round to 2.d.p if cell data type is a float and else just display it as normal.

```
<td>
    <!-- Selection statememnt to format floating point values to 2.d.p -->
    {% if cell is float %}
        {{ '%0.2f'| format(cell|float) }}
    {% else %}
        {{ cell }}
    {% endif %}
</td>
```

Lets test:

V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class	Datetime	Classify
-0.02	0.01	108.51	1	204080	0.53	1	nan	2013-09-02 13:31:42	1 0
0.03	0.06	0.00	1	93789	0.53	1	nan	2013-09-01 17:56:25	1 0
-0.04	0.06	510.36	1	170389	0.51	1	nan	2013-09-02 09:22:36	1 0
0.02	0.11	187.11	1	254345	0.51	1	nan	2013-09-02 19:31:25	1 0
-0.25	0.04	529.00	1	624	0.48	0	nan	2013-09-01 00:07:52	1 0

Success. No compromise! Finally let's split this column into two columns: date and time. To accomplish this I modified the function to return a list of dates and times.

```
def generateDatetimes(timestamps):

    # get starting point in time of dataset
    startingDate = "01/09/2013"
    startTimestamp = time.mktime(datetime.strptime(startingDate, "%d/%m/%Y").timetuple())

    dates = []
    times = []
    for timestamp in timestamps:

        # create datetime object from timestamp
        datetimeObj = datetime.fromtimestamp( startTimestamp + timestamp)

        # retrieve date and time and add them to their respective lists
        dates.append(datetimeObj.date())
        times.append(datetimeObj.time())

    return dates, times
```

And now create two new columns for this data:

```
# add datetimes
dates, times = model.generateDatetimes(list(undecidedDB['Time']))
undecidedDB['Date'] = dates
undecidedDB['TimeStr'] = times
```

Result after reloading:

V27	V28	Amount	V0	ID	Predicted_Prob	Predicted_Class	Decided_Class	Date	TimeStr	Classify
-0.02	0.01	108.51	1	204080	0.53	1	nan	2013-09-02	13:31:42	1 0
0.03	0.06	0.00	1	93789	0.53	1	nan	2013-09-01	17:56:25	1 0
-0.04	0.06	510.36	1	170389	0.51	1	nan	2013-09-02	09:22:36	1 0
0.02	0.11	187.11	1	254345	0.51	1	nan	2013-09-02	19:31:25	1 0
-0.25	0.04	529.00	1	624	0.48	0	nan	2013-09-01	00:07:52	1 0

Success.

Chapter 25: Ordering rows - (Objective 4*)

As you can see from the diagram on the right, you might notice that there is no order with respect to time. I thought I sorted this out in this second line of code below but apparently not.

Time	V1	V2	V3	V4	V5
135102.00	1.86	-0.12	-1.99	0.38	0.47
64585.00	1.08	0.96	-0.28	2.74	0.41
120156.00	0.94	-1.47	-3.12	0.73	1.11
156685.00	-0.13	0.14	-0.89	-0.46	0.81
472.00	-3.04	-3.16	1.09	2.29	1.36
90676.00	-2.41	3.74	-2.32	1.37	0.39

```
# create the subset we are going to show
undecidedDB = decisionDB.loc[decisionDB['Decided_Class'].isna()]
undecidedDB.sort_values('ID') # sort records with respect to time
```

After reading the Pandas documentation, it stated that the sort values method only returns a new sorted list. Therefore I should reassign the undecidedDB back to the sorted data frame:

```
# create the subset we are going to show
undecidedDB = decisionDB.loc[decisionDB['Decided_Class'].isna()]
undecidedDB = undecidedDB.sort_values('ID') # sort records with respect to time
```

Lets see the results. Below is a compressed version of the screenshot.

Time	V1	V2	V3	V4	V5	V6	Predicted_Class	Decided_Class	Date	TimeStr	Classify
472.00	-3.04	-3.16	1.09	2.29	1.36	-1.06	0	nan	2013-09-01	00:07:52	1 0
64585.00	1.08	0.96	-0.28	2.74	0.41	-0.32	1	nan	2013-09-01	17:56:25	1 0
90676.00	-2.41	3.74	-2.32	1.37	0.39	1.92	1	nan	2013-09-02	01:11:16	1 0
120156.00	0.94	-1.47	-3.12	0.73	1.11	1.30	1	nan	2013-09-02	09:22:36	1 0
133389.00	2.10	0.09	-2.13	0.51	0.68	-0.40	0	nan	2013-09-02	13:03:09	1 0

Absolute success since the fraud analyst would want to see the earliest transaction first (ascending timestamp order).

Chapter 26: Restructuring our dashboard content - (Objective 4*)

As you can see from our previous screenshots of the table, it is simply too wide since it contains too many columns to be shown. First we should remove the completely redundant columns. We can change how we show the data. We have already removed ‘Class’ since a transaction’s actual fraudulency wouldn’t be known to anyone except the fraudster or account holder. We performed this action in the function in which we created decisionDB.

Removing redundant columns

We must remove the V0 column as this column contains the same value for all rows, it is completely redundant, hence we can remove it from the saved decisionDB which is a superset of undecidedDB. This will save file space as we are removing a column of text. This won’t actually affect our program’s functionality since this data frame is only used as a reference to obtain the transactionID from where we update the actual data frame.

```
# Drop the column for the actual fraud classification
decisionDB = decisionDB.drop(columns=['Class', 'V0'], axis=1)
```

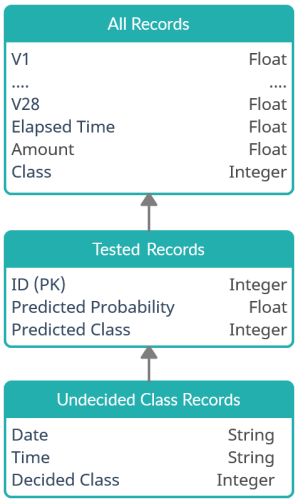
V0 is removed. I do admit using a column to store all the data is a bit of a hack, we should modify the machine learning algorithm so that it recognises the value instead of having to obtain it from the dataframe. Once we've used the timestamps to generate a time and date and ordered our transactions according to this, it is not very beneficial to the analyst since it is just a number of seconds which bear no meaning to the bank.

This means we can replace the new list of times in the form of strings over our previous number of elapsed seconds since the beginning of data. This reassignment is performed in the last line of code in the image below:

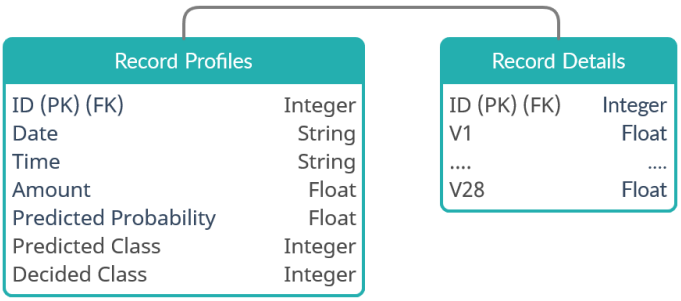
```
# add datetimes
dates, times = model.generateDatetimes(list(undecidedDB['Time']))
undecidedDB['Date'] = dates
undecidedDB['TimeStr'] = times
```

Segmenting the table

Even with the previous column reductions, sticking to the screen design plan, we must split the undecidedDB table into tables. To keep track of the attributes we should be reorganising during the, I have been updating a database inheritance diagram below which is shown below.



Undecided Class Records (undecidedDB) is going to be split into two dataframes: one which shows the transaction profile which the analyst might need to directly reference values, such as date and amount, and the other table being the details of each transaction which will be useful to the analyst in deciding whether the transaction is fraudulent, e.g. V1-V28. I made a diagram to represent how these tables will be split.



The entities have a one-to-one relationship. Therefore it exists that the ID is both the foreign and a primary key in both tables.

Instead of splitting the original copy of undecidedDB, I create two new data frames called profilesTable and detailsTable. I choose the columns I require for each table by listing their names. I use a loop to create the strings ‘V1’ to ‘V28’. I assure the ID is in the first position for both tables so that while looping over cell values I can also know that this column will be first and its value can be inserted into the classify button.

```
# splitting table into column values we will show in the modal window and those that will be shown in the table
detailsColNames = ['ID'] + [f'V{num}' for num in range(1, 29)]
profilesColNames = ['ID', 'Date', 'Time', 'Amount', 'Predicted_Prob', 'Predicted_Class', 'Decided_Class']
profilesTable, detailsTable = undecidedDB[profilesColNames], undecidedDB[detailsColNames]
```

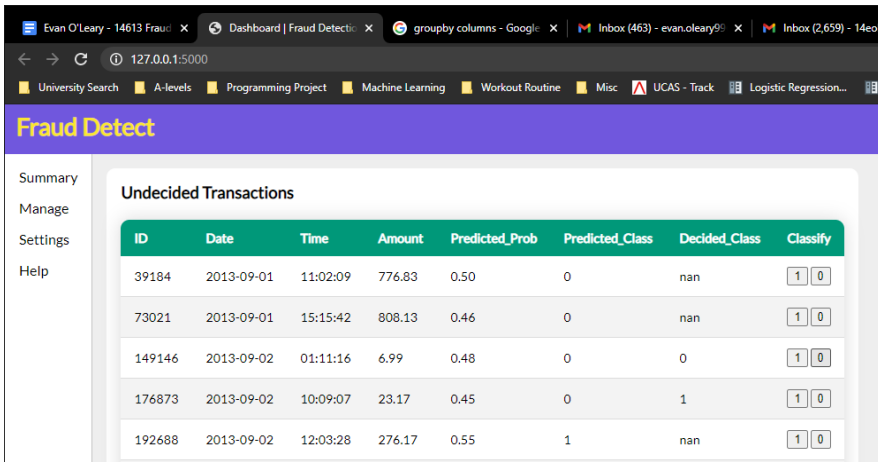
I replaced the profilesTable for undecidedDB in our render function.

```
# render table
return render_template('index.html',  row_data = list(profilesTable.values.tolist()), headings = profilesTable.columns.values)
```

I remembered to substitute the 0th position to insert the transaction ID for our button.

```
<td>
    <!-- Classify particular row-->
    <button data-transactionid = "{{ row[0] }}" data-classification = 1>1</button>
    <button data-transactionid = "{{ row[0] }}" data-classification = 0>0</button>
</td>
```

Reloaded the page:



Success. Let's also confirm that the detailsDB has been created and contains the ID and fields V1-V28 using a print statement in app.py:

```
65      profilesTable, detailsTable = undecidedDB[profilesColNames], undecidedDB[detailsColNames]
66
67      # test to see if detailsTable has been created.
68      print(detailsTable)
69
```

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS                    2: Python

 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
        ID        V1        V2        V3        V4        V5        V6  ...       V22       V23       V24       V25       V26       V27       V28
68    39184 -0.964567 -1.643541 -0.187727  1.158253 -2.458336  0.852222  ...  0.536204  1.634061  0.203839  0.218749 -0.221886 -0.308555 -0.164500
167   73021 -1.197573 -0.321733  2.616074  2.502472 -4.131824  4.262354  ...  1.114847 -0.258551  0.235556 -0.132293  1.422617  0.070305 -0.642796
360  192688  1.522080 -0.519429 -2.581685  0.774741  0.206722 -1.431020  ... -0.211678 -0.247452 -0.279472  0.239646 -0.508398 -0.015551  0.041881
317  204080  1.862102 -0.124052 -1.989752  0.382609  0.473032 -0.674517  ... -0.511441  0.077874  0.388335  0.007896 -0.120980 -0.019579  0.006155
368  233259 -1.611877 -0.408410 -3.829762  6.249462 -3.360922  1.147964  ...  0.616383  2.251439 -0.066096  0.538710  0.541325 -0.136243 -0.009852
484  254345 -0.129778  0.141547 -0.894702 -0.457662  0.810608 -0.504723  ... -0.246526  0.484108  0.359637 -0.435972 -0.248480  0.021527  0.109192
43   260942  2.121415 -0.724698 -1.352614 -0.858305 -0.365159 -0.071412  ...  0.805426  0.083720  0.195716 -0.120595 -0.113102  0.023593 -0.018358

[7 rows x 29 columns]
127.0.0.1 - - [05/Jan/2021 11:52:43] "GET / HTTP/1.1" 200 -
```

Expected outcome achieved.

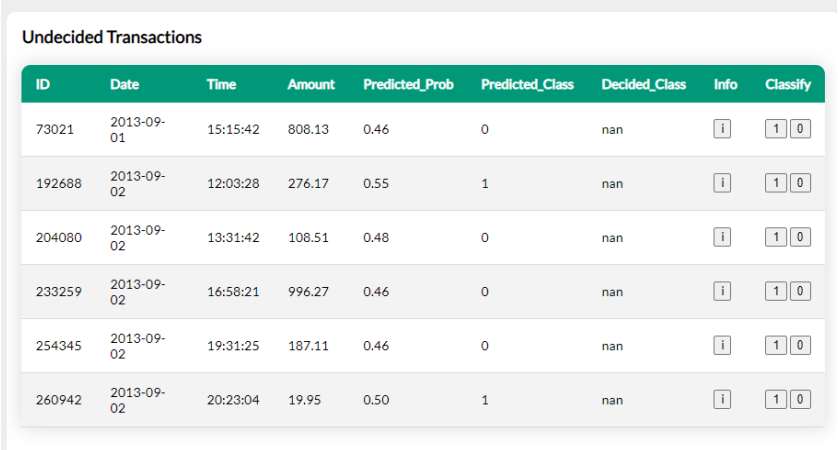
Creating the modal window

I want to display the record profiles in the table we are showing on the page already. Then I am going to create a button on each row which launches a modal window containing all the record details. This modal window will also contain a button to classify the transaction's fraudulency classification to save the user time from exiting from the window and clicking the classify buttons on the webpage.

I want to launch this window by placing a button next to each record in the table. It should find the right record from the javascript by passing the button the transactionID and then from there it can be matched to the variable.

I added the button with the transactionID in the data field set again.

```
        {% endfor %}
        <td>
            <button class="launch-modal-button" data-transactionid = "{{ row[0] }}">i</button>
        </td>
        <td>
            <!-- Classify particular row-->
            <button class="classify-button" data-transactionid = "{{ row[0] }}" data-classification = 1>1</button>
            <button class="classify-button" data-transactionid = "{{ row[0] }}" data-classification = 0>0</button>
        </td>
    </tr>
{% endfor %}
```



Undecided Transactions

ID	Date	Time	Amount	Predicted_Prob	Predicted_Class	Decided_Class	Info	Classify
73021	2013-09-01	15:15:42	808.13	0.46	0	nan	i	1 0
192688	2013-09-02	12:03:28	276.17	0.55	1	nan	i	1 0
204080	2013-09-02	13:31:42	108.51	0.48	0	nan	i	1 0
233259	2013-09-02	16:58:21	996.27	0.46	0	nan	i	1 0
254345	2013-09-02	19:31:25	187.11	0.46	0	nan	i	1 0
260942	2013-09-02	20:23:04	19.95	0.50	1	nan	i	1 0

Then I passed the detailsTable in app.py in the render function.

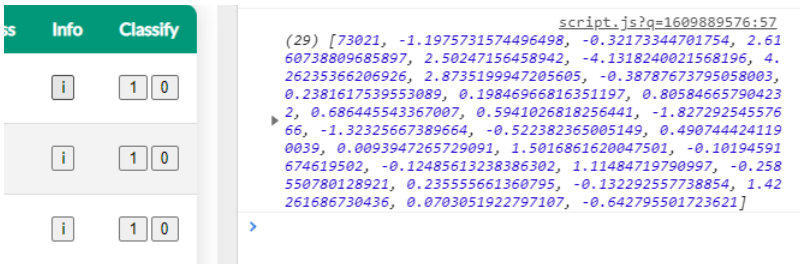
```
# render table
return render_template('index.html',  row_data = list(profilesTable.values.tolist()), table_headings = profilesColNames,
                                      modal_data_table = list(detailsTable.values.tolist()), modal_data_headings = detailsColNames)
```

Then I parsed this data JSON to a global array of records in script tags in index.html since I don't know how to import flask variables into external JS files yet.

```
<script> detailsTable = JSON.parse('{{ modal_data_table}}')</script>
<script src="{{ url_for('static', filename='js/script.js') }}"></script>
```

```
const launchModalButtons = document.getElementsByClassName("launch-modal-button");
for (i = 0; i < launchModalButtons.length; i++) {
    launchModalButtons[i].addEventListener("click", function(e) {

        // Retrieve the data of the record to update from the HTML attrs
        const transactionID = e.target.dataset.transactionid

        // Details yable declared gloabally from index.html script file
        console.log(typeof detailsTable)
        const record = detailsTable.find(record => record[0] == transactionID );
        console.log(record);
    });
};
```

The screenshot below shows that data of the record whose info button we clicked on has successfully been printed to the console.



Now we must create the modal pop up window which this data is going to be loaded into. For now it'll just be a static part of the page so we can just test if the field data is correctly inputted after the info button has been clicked. The jinja 2 double curly brackets syntax commands the HTML page to create a header for every one of the record's column names and also an empty set of paragraph tags which we will populate with the selected record's field values.

```
<!-- Modal content -->
<div class="bg-modal">
    <div class="modal-content">
        {% for header in modal_data_headings %}
            <div class="details-field-conatiner">
                <h4> {{ header }} </h4>
                <p class="details-field-value"></p>
            </div>
        {% endfor %}
    </div>
</div>
```

The javascript code which fills in these field values is in the loop over every detailsField. Learning from the mistakes I made when creating the previous loop which created the table of records, I am going too preemptively round all floating point values to 2.dp for features V1-V28. The question mark in the assignment statement is known as the ternary operator. It's purpose is to choose which code to run like an if/else statement. It rounds the number only if it is a floating point number. Lastly this value is added to the page in the p tags for each field.

```
const launchModalButtons = document.getElementsByClassName("launch-modal-button");
for (i = 0; i < launchModalButtons.length; i++) {
    launchModalButtons[i].addEventListener("click", function(e) {

        // Retrieve the data of the record to update from the HTML attrs
        const transactionID = e.target.dataset.transactionid

        // Details yable declared gloabally from index.html script file
        const record = detailsTable.find(record => record[0] == transactionID );

        // Select elements where the field values will
        const detailsFieldEls = document.querySelectorAll(".details-field-value")

        for (i = 0; i < detailsFieldEls.length; i++) {
            // If floating point, round value to 2.d.p
            fieldDataValue = Number.isInteger(record[i]) ? record[i] : record[i].toFixed(2)
            // Update records values
            detailsFieldEls[i].innerText = fieldDataValue
        };
    });
};
```

The screenshot below shows the modal window before the info button has been pressed.

ID	Date	Time	Amount	Predicted_Prob	Predicted_Class	Decided_Class	Info	Classify
192688	2013-09-02	12:03:28	276.17	0.55	1	nan	i	1 0
204080	2013-09-02	13:31:42	108.51	0.48	0	nan	i	1 0
233259	2013-09-02	16:58:21	996.27	0.46	0	nan	i	1 0
254345	2013-09-02	19:31:25	187.11	0.46	0	nan	i	1 0
260942	2013-09-02	20:23:04	19.95	0.50	1	nan	i	1 0

ID V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22

V23 V24 V25 V26 V27 V28

And the next screenshot shows how these values update when I click the info button of the first record. As you can see it successfully updated these values and printed the correct record. This is evident from the transactionID (192688) being the same in the table and the modal content.

ID	Date	Time	Amount	Predicted_Prob	Predicted_Class	Decided_Class	Info	Classify
192688	2013-09-02	12:03:28	276.17	0.55	1	nan	i	1 0
204080	2013-09-02	13:31:42	108.51	0.48	0	nan	i	1 0
233259	2013-09-02	16:58:21	996.27	0.46	0	nan	i	1 0
254345	2013-09-02	19:31:25	187.11	0.46	0	nan	i	1 0
260942	2013-09-02	20:23:04	19.95	0.50	1	nan	i	1 0

ID	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17
192688	1.52	-0.52	-2.58	0.77	0.21	-1.43	0.76	-0.44	1.00	-1.43	-0.38	0.18	-0.15	-2.14	-0.04	-0.06	1.78

V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28
0.67	-0.00	0.36	0.02	-0.21	-0.25	-0.28	0.24	-0.51	-0.02	0.04

Tests pass successfully and when other records info buttons are pressed, the data in the modal is updated dynamically without the need to refresh the page. Now we need to make the window pop up, darken the background and disappear when another part of the page is clicked.

So in the CSS we are going to make the modal darkened opaque background. It will initially be hidden (display=none). Once activated it must take up 100% of the screen's real estate. And finally it must sit on top of all the other pages elements (z-index=1):

```
/* Modal Popup */
.bg-modal {
    display: none; /* Hidden by default */
    position: fixed; /* Stay in place */
    z-index: 1; /* Sit on top */
    left: 0;
    top: 0;
    width: 100%; /* Full width */
    height: 100%; /* Full height */
    overflow: auto; /* Enable scroll if needed */
    background-color: rgb(0,0,0); /* Fallback color */
    background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
}
```

The actual pop up content itself is above the modal background, so will be the only thing the user can interact with once the modal is activated. It should take up 50% of the screen's width. And it will have a white background colour for clarity.

```
.modal-content {
    width: 50%;
    padding: 0 10px;
    margin: 15% auto;
    background-color: white;
}
```

Now we must configure the JavaScript code so that this pop up is launched and updated with the correct information only when clicked. The modal style is changed from none to block at the end of the info button event listener action.

```
        for (i = 0; i < detailsFieldEls.length; i++) {
            // If floating point, round value to 2.d.p
            fieldDataValue = Number.isInteger(record[i]) ? record[i] : record[i].toFixed(2)
            // Update records values
            detailsFieldEls[i].innerText = fieldDataValue
        };
        // Launch Modal
        modal.style.display = "block";
    });
};

// When the user clicks anywhere outside of the modal, close it
window.onclick = function(event) {
  if (event.target == modal) {
    modal.style.display = "none";
  }
}
```

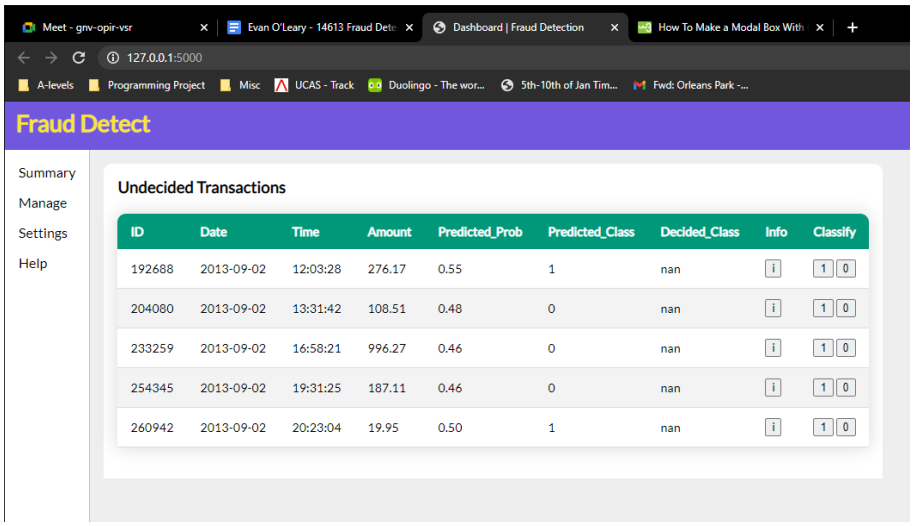
The window.onclick commands the modal style to be set back to none (closed) if the pop up window is active and anywhere apart from the pop up window is clicked. I also added a close button in the modal content.

```
<!-- Close Pop up Button -->
<span class="close">&times;</span>

<!-- Data Fields for transaction -->
{% for header in modal_data_headings %}
```

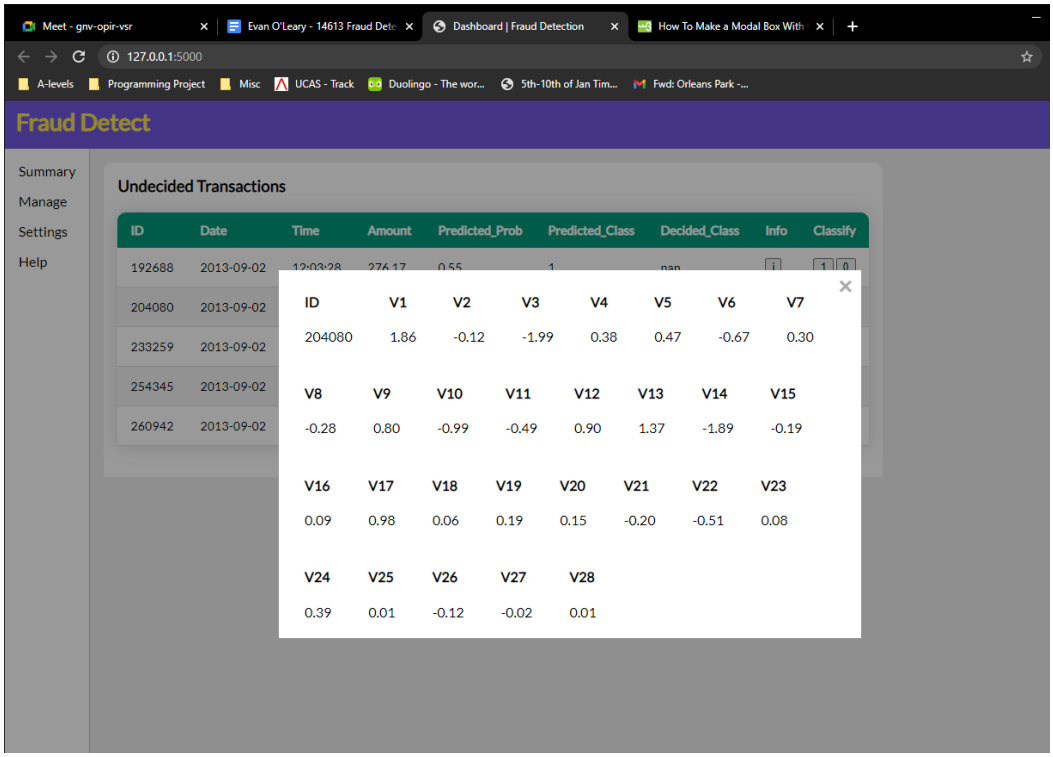
This javascript again closes the modal window once the close button is clicked.

```
// When the user clicks on <span> (x), close the modal
document.querySelector('.close').onclick = function() {
    modal.style.display = "none";
}
```

Now let's test if this worked. So we are going to test if the modal with the correct information opened after clicking the info button on the second record. Here’s the screen before clicking:



And after:



Success. The second records ID matched that shown in the modal window. Clicking on the cross in the top right corner or anywhere else on the screen. The window closed. It went back to the first screenshot. Success!

27. Managing the database page - (Objective 4*)

The fraud analyst should have access to all transactions and be able to change any of their decided fraud classifications. This is the purpose of the second page on our site, it'll be called 'manage'.

Time efficiency

First I moved the code which added the dates and times into the function which created decisionDB. Therefore they will now be saved to the file itself since we do not want to execute this code for hundreds of transactions every time the page is refreshed.

Creating the table

I was able to reuse the code from our dashboard HTML page. However I added a third classification to set the fraud class to none (undecided). The purpose of this is so that the fraud analyst can select the transactions that they've chosen not to classify yet. This means it will display on the dashboard again with the undecided transactions so the analyst can reevaluate them.

```
<!-- Iterate over every row -->
{% for row in row_data %}
    <tr>
        <!-- Iterate over every cell -->
        {% for cell in row %}
        <td>
            <!-- Selection statememnt to format floating point values to 2.d.p -->
            {% if cell is float %}
                {{ '%0.2f'| format(cell|float) }}
            {% else %}
                {{ cell }}
            {% endif %}
            </td>
        {% endfor %}
            <td>
                <!-- Classify particular row-->
                <button class="classify-button" data-transactionid = "{{ row[0] }}" data-classification = 1>1</button>
                <button class="classify-button" data-transactionid = "{{ row[0] }}" data-classification = 0>0</button>
                <button class="classify-button" data-transactionid = "{{ row[0] }}" data-classification = nan>None</button>
            </td>
    </tr>
{% endfor %}
```

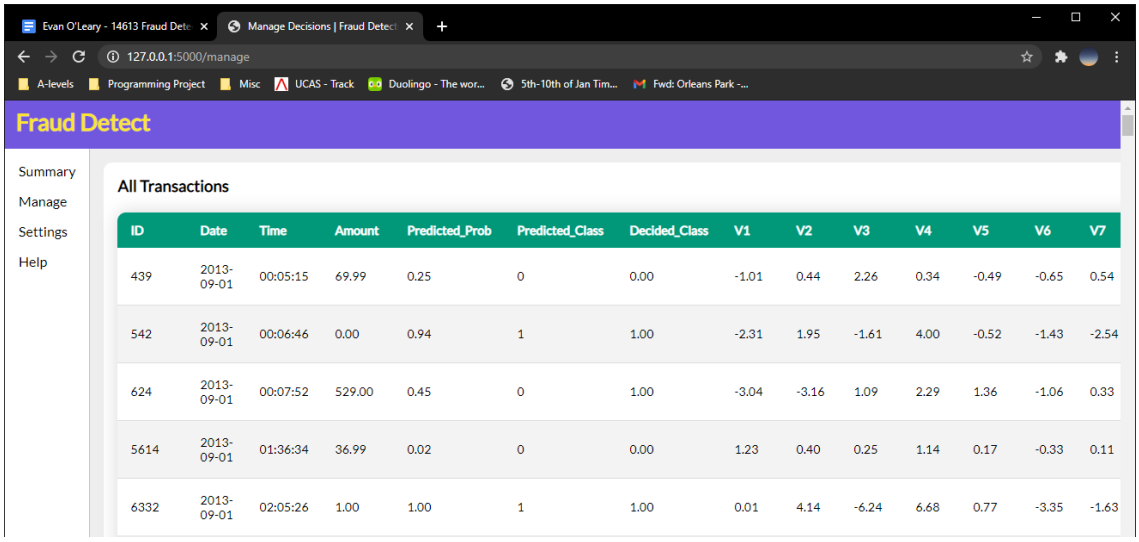
I also moved the JavaScript code which enables the modal content functionality to separate script tags in index.html since this is the only page it is required. I kept the JavaScript code which enabled the POST requests to change the classifications of records in the database in the external file so we can reuse the functionality again in this page:

```
<script src="{{ url_for('static', filename='js/classify.js') }}"></script>
```

This time I substituted the table data and columns for the entire set of tested transactions. I listed the columns again in the order to be displayed and left out Elapsed_Time since it is not needed.

```
@app.route('/manage')
def manage():

    # bring in pre-tested dataset
    decisionDB = pd.read_csv("dataset/decisionDB.csv")

    # get the columns in the order to be presented
    columnsToDisplay = ['ID', 'Date', 'Time', 'Amount', 'Predicted_Prob', 'Predicted_Class', 'Decided_Class'] + [f'V{num}' for num in range(1, 29)]
    decisionDB = decisionDB[columnsToDisplay]

    return render_template('manage.html', row_data = list(decisionDB.values.tolist()), table_headings = decisionDB.columns.values)
```

This is the rendered manage.html page with all the transactions in decesionDB:



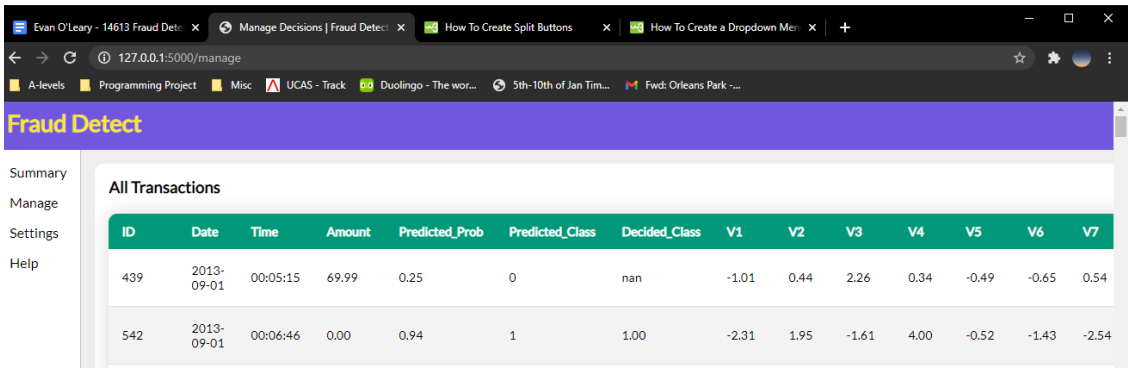
ID	Date	Time	Amount	Predicted_Prob	Predicted_Class	Decided_Class	V1	V2	V3	V4	V5	V6	V7
439	2013-09-01	00:05:15	69.99	0.25	0	0.00	-1.01	0.44	2.26	0.34	-0.49	-0.65	0.54
542	2013-09-01	00:06:46	0.00	0.94	1	1.00	-2.31	1.95	-1.61	4.00	-0.52	-1.43	-2.54
624	2013-09-01	00:07:52	529.00	0.45	0	1.00	-3.04	-3.16	1.09	2.29	1.36	-1.06	0.33
5614	2013-09-01	01:36:34	36.99	0.02	0	0.00	1.23	0.40	0.25	1.14	0.17	-0.33	0.11
6332	2013-09-01	02:05:26	1.00	1.00	1	1.00	0.01	4.14	-6.24	6.68	0.77	-3.35	-1.63

Success! All ordered and with date and time. Scroll depth shows the large scale of how many transactions are shown. You may also notice that these rows are about a line in height taller. This is due to the new button being stacked below the other two classification buttons:

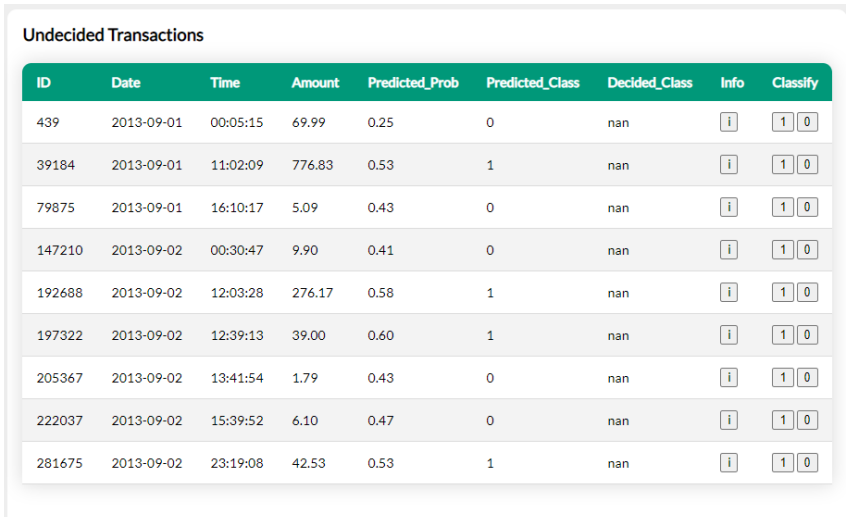
V23	V24	V25	V26	V27	V28	Classify
-0.11	1.01	-0.10	0.40	0.31	0.10	1 0 None

I'm going to test if classifying the transaction shown in the first row as None (nan) from its current classification (0) will make sure it is displayed as an undecided transaction in the dashboard. Note that the record's ID was 439.

This is the manage page after clicking classify as None:



Decided_Class updated instantaneously and remained after page refresh. Success so far! Now let's check if it has appeared in the undecided transactions table on the dashboard:



It's present as the top row!! Success.

Chapter 28: Adding graphs - (Objective 4*)

I want to add the graphs to the dashboard page to track the success of the program and inform the fraud analyst about fraudulent activity.

Here are the following graphs I'm considering adding:

1. Pie-chart or bar graph for amount of frauds versus non-fraudulent transactions across the day so far.
2. Box-plot of amount (£) of each of the transaction classifications.
3. Time series for amount of transactions of each classification per hour.

4. A two variable plot and show where fraud transactions cluster. Users can select the variables.

Second interview with the stakeholder

I have space for at least 2 plots to inform the fraud analyst how similar transactions should be classified due to their characteristics. I decided to arrange a second interview with the stakeholder so I could propose to him the possible graphs and see which ones he thinks will be the most useful and why.

I also hosted my web application on a live server called PythonAnywhere and sent him the link so that he could interact with it and give feedback on what features were missing and any important changes needed to be made.

I also informed the project is still in progress and that the settings and help sections hadn't started their development. I also requested that feedback on the design (colours, positioning) was not required yet and that the focus should be on the functionality and usability of the application.

Transcript

Yorke was emailed all but the second questions prior to the interview to give him time to properly articulate his response. The interview itself was conducted in-person and the audio was recorded. I have transcribed this recording below.

Question 1

Interviewer: Out of the five possible graphs I listed,
Which two did you think would be most useful to a fraud analyst?

Yorke: Personally, I think the more data the better but also the quality of that data is important too. The more granular the better. That's why I liked the two variable plots, where variables can be chosen by the analyst. The pie chart would be pretty unhelpful, fraud cases are nearly always going to be a tiny fraction of the data which the analyst is aware of. Moreover the fraction at that scale will seem near constant every day so shows no variation so is unhelpful. However, that's the reason I thought that the time series of the different classifications per hour would be very useful, even for a more realistic high ratio of the quantity of non-frauds to fraud to fraud cases. There will be a high variation of cases per hour which will reflect the trends of fraud throughout the day.

Question 2

Interviewer: What time intervals should I use for the time series?

Yorke: If it's a bank with millions of customers then you would assume a constant stream of fraud or red flags but if it's a small business or very bespoke then the frequency might be less so you'd perhaps need larger time intervals, e.g. every hour.

Question 3

Interviewer: Did you have any trouble using the application?

Yorke: Nope. I didn't encounter any bugs or defects while using the application. I was able to navigate between pages using the sidebar. Changing the fraudulency classifications of records in the table worked as expected. However the sheer amount of records in the manage section made it difficult to navigate through them and find the one Iwas looking for. I wasn't quite sure what the difference between decided class and predicted class was. I don't think you need both. Getting rid of some of the unhelpful details will make it easier to read.

Question 4

Interviewer: Were there any features missing that you expected?

Yorke: Like I mentioned, it was difficult to find particular transactions due to the scale of the data. This should be the main improvement: the manage page needs some sort of search or filtering option. It doesn't need to be incredibly advanced and complex but it should provide enough filters to nail down certain transactions.

Interview Analysis

Data granularity refers to the level of detail of our data. The more granular your data, the more information contained in a particular data point.

I'm going to do a number of frauds per hour in line with the number of non frauds per hour, going back 48 hours. So there should be 48 data points which is quite a large number

Chapter 29: Adding the time-series graph - (Objective 4*)

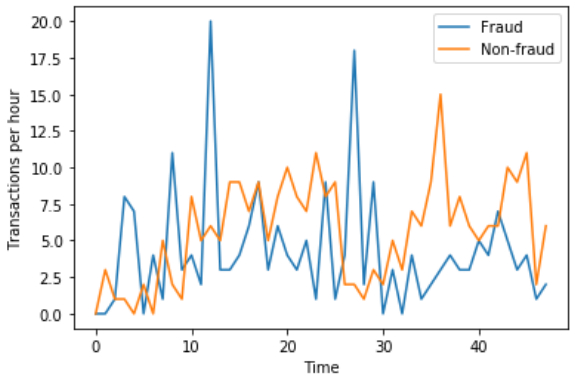
This graph will record the number of cases of fraud and non-fraud in hourly intervals. We are going to create this data by iterating over every transaction in decisionDB. Every time the transaction time is 3600 seconds (1 hour) over the current hours amount of elapsed seconds then we are going to add our recording and start a new data point for the number of fraud/non-fraud transactions per hour. There should be 48 data points over the 2 days.

I decided to return to Jupyter to create this graph since I can show plots as output so it will be easier to debug.

```
def createPlot(decisionDB):

    # intialise variables
    fraudsPerHour, nonFraudsPerHour = [], []
    currHour = 0
    fraudsThisHour = nonFraudsThisHour = 0

    # we are going to iterate over every transaction in decisionDB -  already ordered by time
    for index, transaction in decisionDB.iterrows():

        # check if we've moved on to next hour
        if transaction['Elapsed_Time'] >= currHour * 3600 :
            # add current measurement
            fraudsPerHour.append(fraudsThisHour)
            nonFraudsPerHour.append(nonFraudsThisHour)

            # reset count and move onto next hour
            currHour += 1
            fraudsThisHour = nonFraudsThisHour = 0

        # add transaction data if fraud
        if transaction['Decided_Class'] == 1:
            fraudsThisHour += 1

        # add transaction data if not-fraud
        elif transaction['Decided_Class'] == 0:
            nonFraudsThisHour += 1

    # create time x-axis
    hours = [hour for hour in range(48)]

    # Input features to plot
    plt.plot(hours, fraudsPerHour)
    plt.plot(hours, nonFraudsPerHour)

    # label the axis
    plt.xlabel('Time')
    plt.ylabel('Transactions per hour')

    # Create legend
    plt.legend(["Fraud", "Non-fraud"])

    # Display
    plt.show()
```

I imported decisionDB into our workspace and ran the function.

```
In [57]: createPlot(decisionDB)
```



I decided these were reasonable numbers since there was an average of about 10 total transactions an hour in our database over 48 hours. So in total 480 transactions which was close to the actual value of 492.

Therefore I can continue. My next step was to save this data into a format which could be passed as a variable by the render function and displayed as an image by the browser. I chose to use the Base64 encoding scheme to convert binary data into ASCII, from which I could transmit the image data without loss or modification of the contents itself. In the HTML document I will then decode this data using a built in flask command and display the image.

Here's the code which saves the image data and translates it into base64.

```
# label the axis
plt.xlabel('Time (Hours)')
plt.ylabel('No. transactions per hour')

# Create legend
plt.legend(["Fraud", "Non-fraud"])

# You can load your input data you have into the BytesIO before giving it to the library.
# After it returns, you can get any data the library wrote to the file from the BytesIO using the getvalue() method.
figfile = BytesIO()

# load the image data into temporary file (IO)
plt.savefig(figfile, format='png')

# rewind to beginning of file
figfile.seek(0)

# Base64 encoding is a type of conversion of bytes into ASCII characters
figdata_png = base64.b64encode(figfile.getvalue())
return figdata_png
```

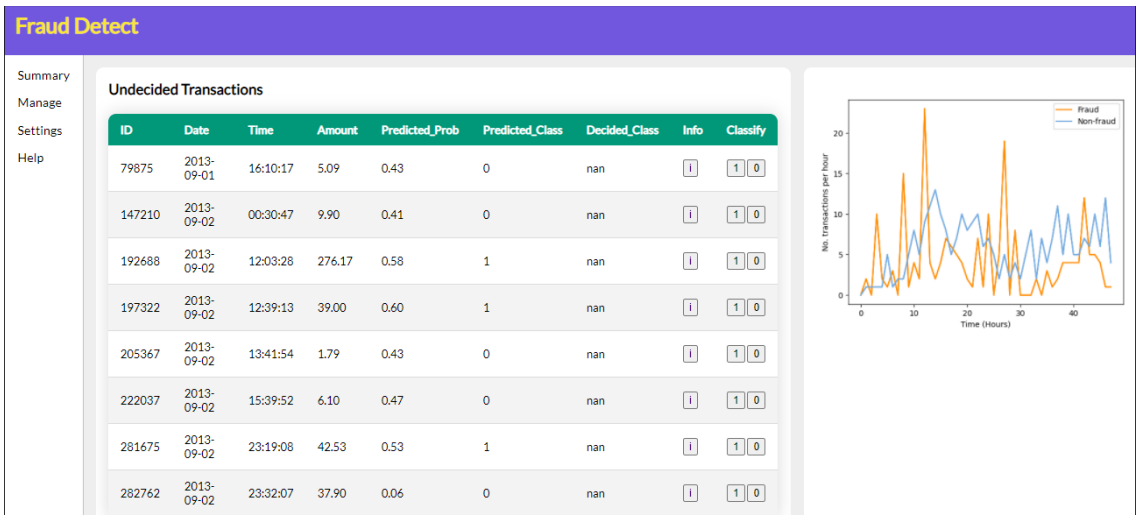
In the function which returns the index view in app.py we call the newly mate createPlot function to return this image in the base 64 representation, and then we decode it into the unicode format which can be understood by the browser:

```
# select these columns from the databse
profilesTable, detailsTable = undecidedDB[profilesColNames], undecidedDB[detailsColNames]

figdata_png = model.createPlot(decisionDB)

# render table and dashboard page
return render_template('index.html',  row_data = list(profilesTable.values.tolist()), table_headings = profilesColNames,
                       modal_data_table = list(detailsTable.values.tolist()), modal_data_headings = detailsColNames,
                       timeseries = figdata_png.decode('utf8'))
```

On the browser's end, we create an if statement which checks if the image was successfully passed and if so converts it from base64 into an image we can see:

```
<!-- Timeseries graph -->
<div class="graph-container">
    <!-- Print graph if it is successfully passed-->
    {% if timeseries != None %}
        <img src="{{ timeseries }}" id="timeseries">
    {% endif %}
</div>
```

Now let's reload the web app and check if it loads as expected



Success!

Tests added for post-development phase

No	Type	What is being tested?	Input / Action Taken	Expected Outcome
Choose date on time series				
12	Valid	Cursor clicks on	Mouse click on	New time series for

		a date for which data in the table exists.	calendar date.	that day is loaded.

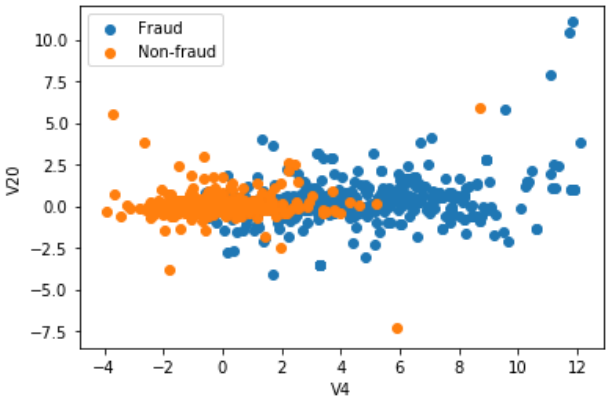
Chapter 30: Adding the 2 variable scatter plot selector - (Objective 4*)

I previously created the function which takes in the two variables (features) and displays a scatter graph for the data of a given dataset.

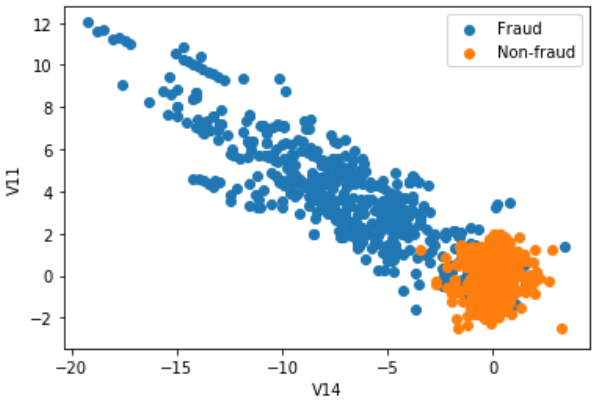
```
# Plotting the data - .plot(...) for line & .scatter(...) for scatter

def plotFeatures(trainingSet, feature1, feature2):

    fraud = trainingSet[trainingSet.Class == 1]
    non_fraud = trainingSet[trainingSet.Class == 0]

    # Input features to plot
    plt.scatter(fraud[feature1], fraud[feature2])
    plt.scatter(non_fraud[feature1], non_fraud[feature2])

    # label the axis
    plt.xlabel(feature1)
    plt.ylabel(feature2)

    # Create legend
    plt.legend(["Fraud", "Non-fraud"])

    # Display
    plt.show()
```

These two previously performed and recorded test results screenshots show that this function can be reused as it works properly:

```
plotFeatures(dataset, 'V4', 'V20')
```




```
plotFeatures(dataset, 'V14', 'V11')
```



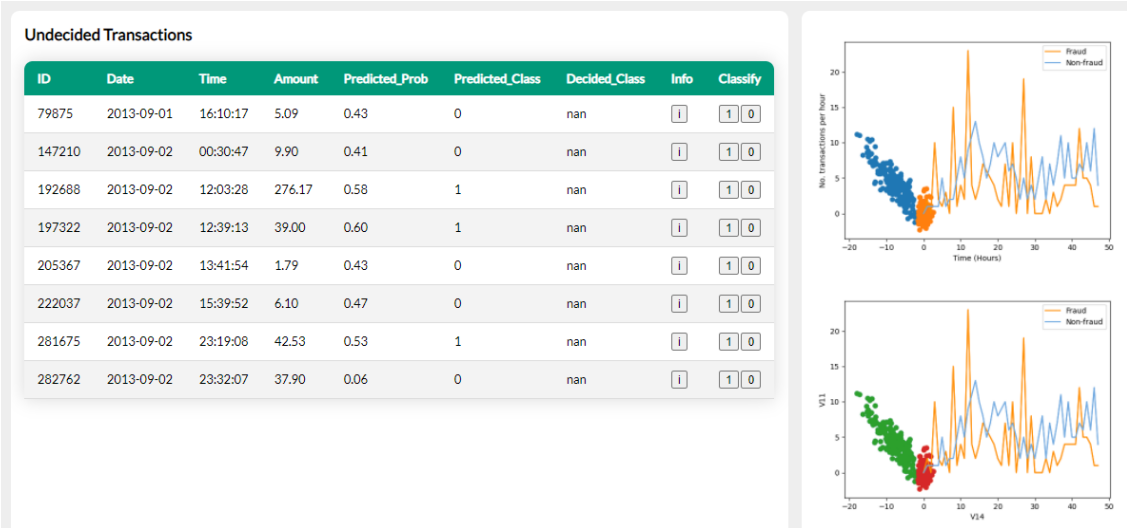
I'm going to modify the function so that it returns the base64 image data instead of instantly printing the image to the terminal. I'm also going to rename it 'createScatter' to avoid having a similar name to the time series creating function, which we will rename 'createTimeseries'.

```
def createScatter(dataset, feature1, feature2):

    fraud = dataset[dataset.Decided_Class == 1]
    non_fraud = dataset[dataset.Decided_Class == 0]

    # Input features to plot
    plt.scatter(fraud[feature1], fraud[feature2])
    plt.scatter(non_fraud[feature1], non_fraud[feature2])

    # label the axis
    plt.xlabel(feature1)
    plt.ylabel(feature2)

    # Create legend
    plt.legend(["Fraud", "Non-fraud"])

    # You can load your input data you have into the BytesIO before giving it to the library.
    # After it returns, you can get any data the library wrote to the file from the BytesIO using the getvalue()
    figfile = BytesIO()

    # load the image data into temporary file (IO)
    plt.savefig(figfile, format='png')
    plt.clf()

    # rewind to beginning of file
    figfile.seek(0)

    # Base64 encoding is a type of conversion of bytes into ASCII characters
    figdata_png = base64.b64encode(figfile.getvalue())
    return figdata_png
```

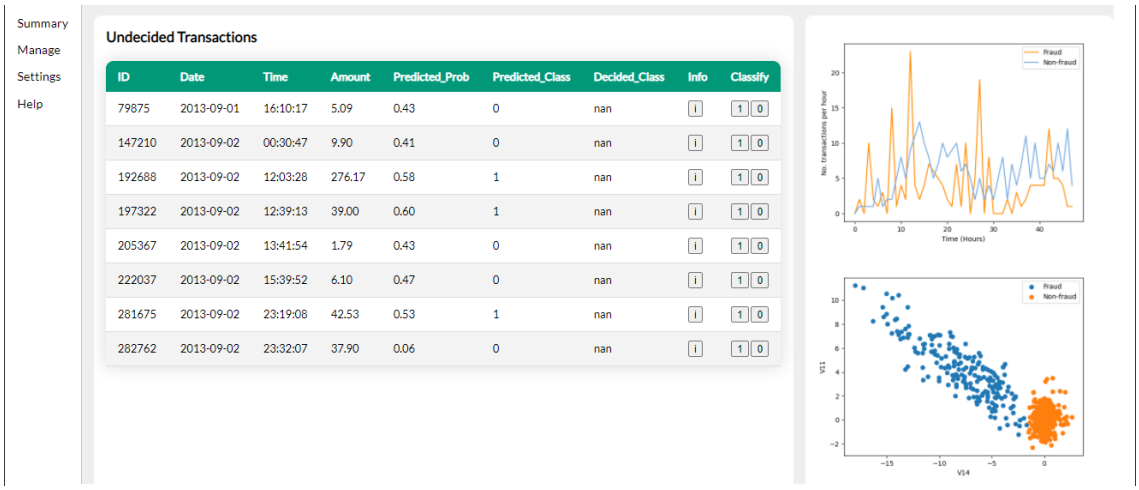
In app.py we created the variable for it to be passed into our application:

```
# select these columns from the databse
profilesTable, detailsTable = undecidedDB[profilesColNames], undecidedDB[detailsColNames]

# create the image data for the plots
timeseriesB64 = model.createTimeseries(decisionDB)
scatterB64 = model.createScatter( decisionDB,'V14', 'V11')

# render table and dashboard page
return render_template('index.html',  row_data = list(profilesTable.values.tolist()), table_headings = profilesColNames,
                       modal_data_table = list(detailsTable.values.tolist()), modal_data_headings = detailsColNames,
                       timeseriesImg = timeseriesB64.decode('utf8'), scatterImg = scatterB64.decode('utf8'))
```

In the body of the createScatter and createTimeseries functions in model.py I had to add the plt.clf() method call below the line where we save the figure. The purpose of this is to clear the figure after it's called. That's so that it doesn't overlap over the next created figure. Using the 'V11' and 'V14' variables to test, lets launch the graph without the clear figure call to demonstrate the defect it causes when the plt.clf method is commented out:



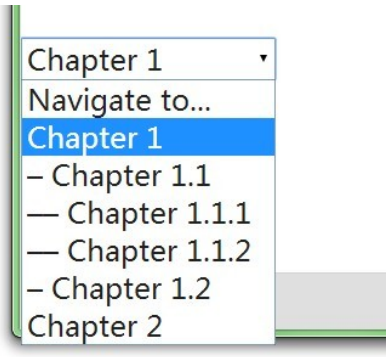
Fails the test that the graph can be read clearly!, so lts uncomment the clear figure call line of code and relaunch:



Success!

Chapter 31: Scatter Plot Variable Axis - (Objective 4*)

I want to give the user a select menu for each axis using a select menu:

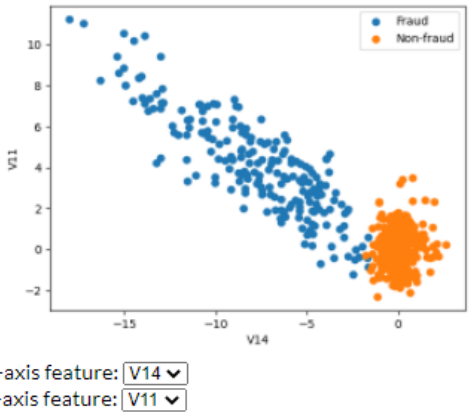


I want to keep v14 and v11 as our default preselected options. I'm going to use a jinja 2 loop over the names of the features which we want to graph (V1-V28). We can reuse the list of these features from the modal section. We don't want to include the ID attribute which is in the first position so we will only do after the 1st index:

```
<!-- Print graph if it is successfully passed-->
{% if scatterImg != None %}
    <img src="{{ scatterImg }}" id="scatter">
{% endif %}
<div class="axis-selectors-container">

    <!-- X-axis of 2 variable scatter plot -->
    <label for="x-axis">X-axis feature: </label>
    <select name="x-axis" id="x-axis" class="axis-selector">
        {% for feature in modal_data_headings[1:] %}
            {% if feature == 'V14' %}
                <option value = "V14" selected="selected"> V14 </option>
            {% else %}
                <option value = "{{ feature }}"> {{ feature }} </option>
            {% endif %}
        {% endfor %}
    </select>

    <br>

    <!-- Y-axis of 2 variable scatter plot -->
    <label for="y-axis">Y-axis feature: </label>
    <select name="y-axis" id="y-axis" class="axis-selector">
        {% for feature in modal_data_headings[1:] %}
            {% if feature == 'V11' %}
                <option value = "V11" selected="selected"> V11 </option>
            {% else %}
                <option value = "{{ feature }}"> {{ feature }} </option>
            {% endif %}
        {% endfor %}
    </select>

</div>
```

Lets run a test to see if this worked:

Success. V14 and V11 are the axes.



Next we are to create a function in app.py called ‘getScatter’ which sends the image data for a scatter graph with the requested axes with the data fields (V1-V28) selected by the user.

Like in the function which handled the ‘classify transaction’ HTTP request, we read the request data and assign it’s parameters to local variables, this time however, we are sending a GET request since we are not making a change to the database we are asking for a resource processed on the server. This is different to a POST request which has the purpose of making a change to the database.

The code for this function is below:

```
# Classify case - responding to input
@app.route('/getScatter', methods = ['GET'])
def getScatter():

    # Import the decisionsDB
    decisionDB = pd.read_csv("dataset/decisionDB.csv")

    print(request)

    # GET querystring arguments from request
    x_axis = request.args['x_axis']
    y_axis = request.args['y_axis']

    # Plot the graph and generate the encoded image data for it
    scatterB64 = model.createScatter( decisionDB, x_axis, y_axis).decode('utf8')

    # Return image data
    return jsonify({'imgB64': scatterB64})
```

The ‘methods’ argument for routing our application specifies that GET is the only available action we can make to this URL; all other types will be denied. The ‘args’ attribute is a map of the variable names to their values in the sent querystring. The jsonify function is provided by the flask framework and is different from a standard JSON formatter. This is because it actually returns a HTTP response object which returns the usual status and headers as well as the payload data preformatted in JSON.

Now we have the server-side function which handles the request and returns the image data, let's create the client-side JavaScript code which listens for the action of changing the selected option, obtains the current axes variables and sends the GET request for their scatter graph image data.

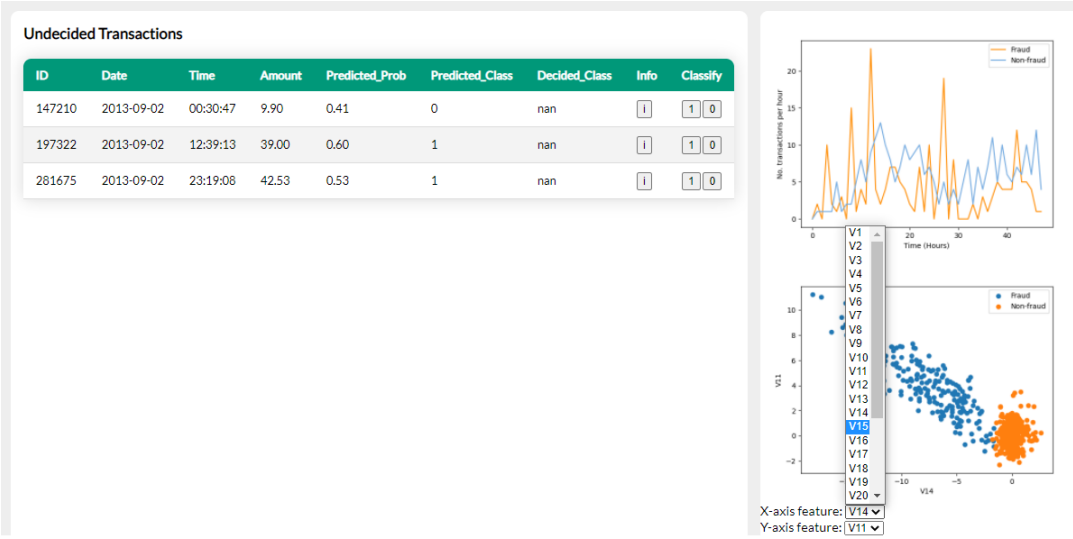
First I created a variable for the scatterPlot image element already present on the page outside of the event listener function in the global scope since it would waste time to call it every time the event is triggered.

```
// Get image to load it in
const scatterPlot = document.querySelector('#scatter')

document.querySelectorAll('.axis-selector').forEach(selectorEl => {
    selectorEl.addEventListener('change', async e => {

        // Get axis variable values from DOM
        if (e.target.id == 'x-axis') {
            var x = e.target.value;
            var y = document.querySelector('#y-axis').value;
        } else {
            var x = document.querySelector('#x-axis').value;
            var y = e.target.value;
        }

        // Data to create the graph
        const queryString = '?x_axis=' + x + '&y_axis=' + y;
        const URLaddress = server + '/getScatter' + queryString;

        // Make POST Request to receive the image data for the new graph
        fetch( URLaddress)
            .then(response => {
                // GET Request successful - update image
                return response.json();
            })
            .then( data => {
                scatterPlot.src = '' + data.imgB64;
                console.log("Image updated.");
            })
            .catch(error => {
                // GET Request unsuccessful
                console.log(error);
            });
        })
})
```

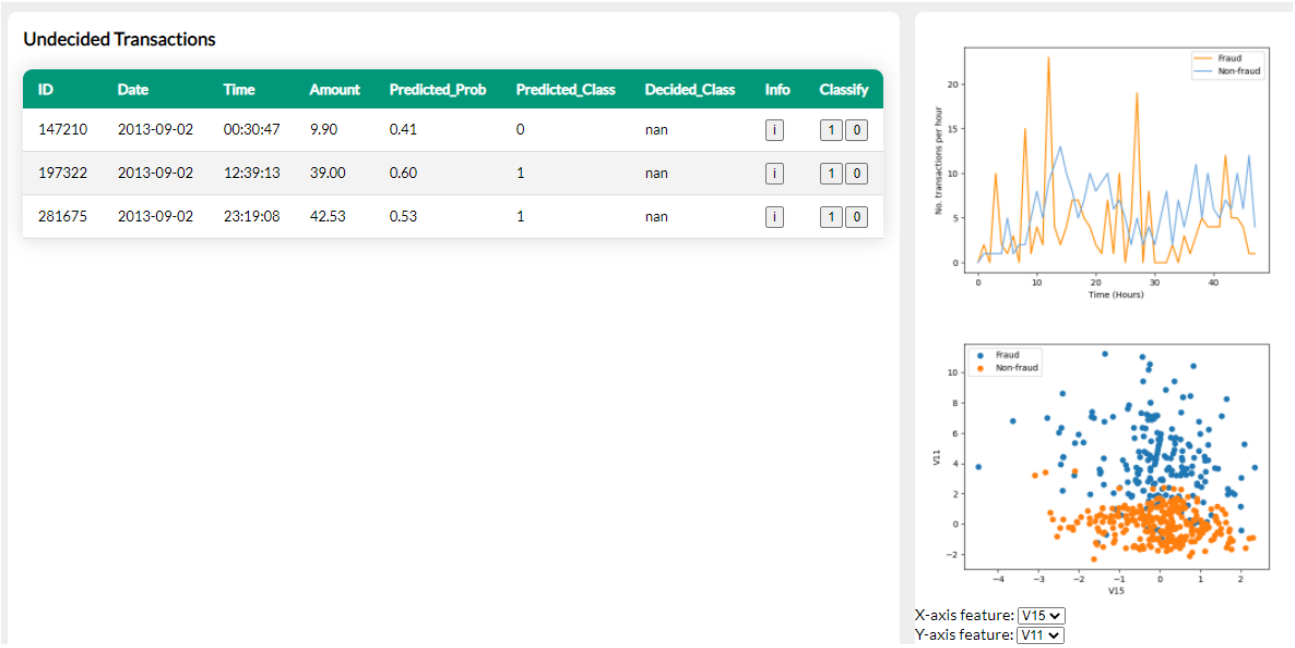
The first selection block figures out whether the x-axis selected option was changed, and if so sets the value of the feature equal to this changed value, and then gets the value in the y-axis selector from the DOM. And vice versa when the y-axis selected option is changed.

I then used a query string to input this data into the URL for our GET request. The catch command which prints the error message is there to help us debug if any errors are produced by the server side code. Otherwise the request is then made and the response JSON is converted and the base64 image encoded data is the new source for our scatterPlot image to display.

Lets test that the request is successfully sent and the response changes the image correctly. Here's the screenshot of the graph while hovering over the variable of the x-axis select menu we want to change:



All variables successfully shown to the user. Below is the screenshot of the result of clicking down and changing the x-axis variable from v14 to v15:



Success!

Tests added for post-development phase

No	Type	What is being tested?	Input / Action Taken	Expected Outcome

Choose date on time series				
13.1	Valid	Users can look for clustering of fraud / non-fraud cases between V14 and V16.	Mouse click on the menu.	Scatter plot (colour coded by orange and blue for fraud and non-fraud) is displayed for two variables.
13.2	Valid	Users can look for clustering of fraud / non-fraud cases between V4 and V10.	Mouse click on the menu.	Scatter plot (colour coded by orange and blue for fraud and non-fraud) is displayed for two variables.
13.3	Boundary	Users can find the clustering for fraud / non-fraud cases for one variable by looking at the same axes.	Mouse click on the menu.	Should display a straight line and fraud and hopefully non-fraud are grouped.

Chapter 32: Responding to interview feedback - (Objective 4*)

Removing the Predicted Class From Table

In the previous interview, Yorke fed back that the predicted class column was not necessary for the fraud analyst to see since they can see the predicted probability of the transaction. However I will need this column for the prediction model's success analysis, so I cannot out right remove this column from the database. Instead I will remove it from the list of columns which index the data frame and rendered.

Undecided Transactions

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	Info	Classify
77588	2013-09-01	15:51:58	40.00	0.44	nan	i	1 0
79875	2013-09-01	16:10:17	5.09	0.45	nan	i	1 0
93487	2013-09-01	17:54:03	0.00	0.45	nan	i	1 0
93789	2013-09-01	17:56:25	0.00	0.51	nan	i	1 0
149146	2013-09-02	01:11:16	6.99	0.51	nan	i	1 0
151731	2013-09-02	02:42:15	18.96	0.59	nan	i	1 0
166538	2013-09-02	08:49:02	14.99	0.42	nan	i	1 0
179385	2013-09-02	10:27:32	110.14	0.41	nan	i	1 0
192688	2013-09-02	12:03:28	276.17	0.57	nan	i	1 0
241743	2013-09-02	17:59:22	3.57	0.51	nan	i	1 0
247674	2013-09-02	18:40:53	247.86	0.43	nan	i	1 0
250449	2013-09-02	19:01:53	90.18	0.50	nan	i	1 0
254345	2013-09-02	19:31:25	187.11	0.49	nan	i	1 0
281675	2013-09-02	23:19:08	42.53	0.52	nan	i	1 0

It looks much cleaner now. Removing unnecessary detail is abstraction. This should improve the fraud analysts experience using the software.

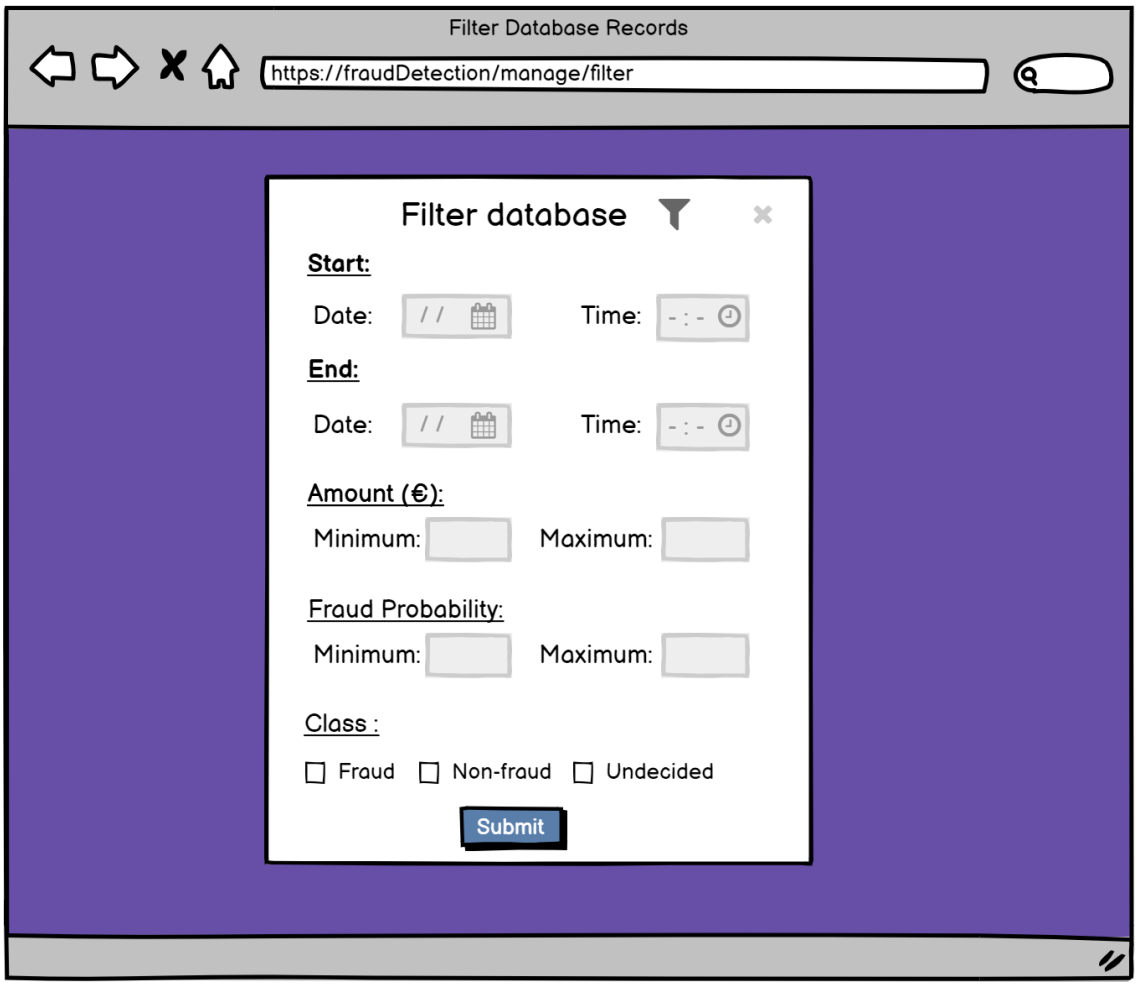
Chapter 33: Filtering the manage table - (Objective 4*)

Identifying the attributes a fraud analyst might filter rows by.

The fraud analyst is unlikely to want to filter by features V0-V28. Yorke suggested that an analyst would like to filter by date and time so that they can investigate transactions in a period of time of frequent fraudulent activity. York also stated that the project should have the bank's priorities in mind. That's why the user should be able to filter by financial significance, e.g amount (€) in each transaction, and also the classification type. This is because the analyst would most likely want to be able to investigate and review high cost fraudulent transactions. The analyst would also like to review the level of risk (predicted probability of fraudulency) which additionally be another concern of the banks priorities.

Finally, the dataset is very large and this filtering process needs to be precise. Therefore the form should offer a minimum and maximum bound of all the data we want to search. E.g. for predicted fraudulency probability, could be searched between 0.2 and 0.5.

I created a mockup to demonstrate what this should look like:



Creating the filter form

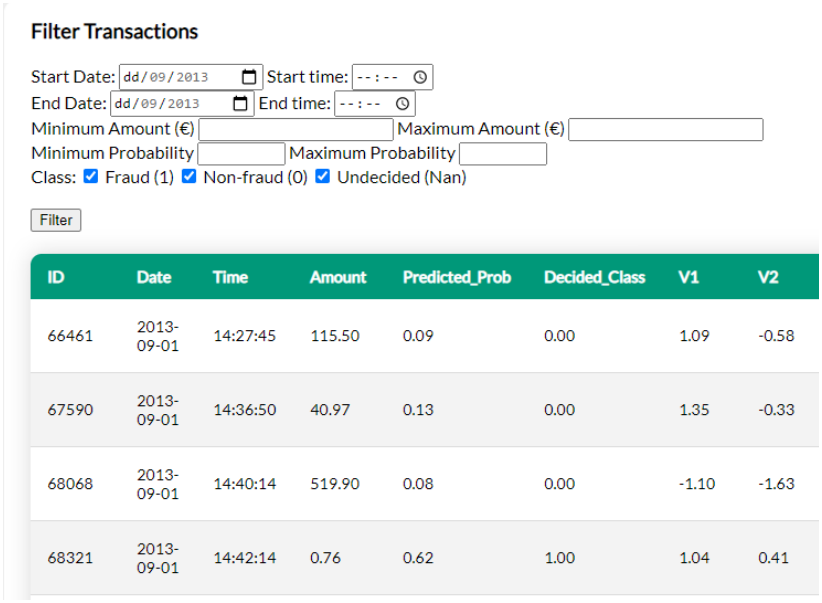
We are making a GET request because we are pulling filtered data processed on the server and printing it to the screen. We are going to send the request to the manage view in app.py which originally renders the template for the table with all tested data. What differentiates this request to our original one is that the URL will contain a query string of arguments to filter the database and return in place of the original complete set of transactions.

Here's the HTML code for this form:

```
<!-- Filter Table Form -->
<form action="/manage" method="GET" id="filter-table-form">

    <!-- Start Date Time -->
    <label for="start_date">Start Date:</label>
    <input type="date" name="start_date" min="2013-09-01" max="2013-09-02">

    <label for="start_time">Start time:</label>
    <input type="time" name="start_time"> <br>

    <!-- End Date Time -->
    <label for="end_date">End Date:</label>
    <input type="date" name="end_date" min="2013-09-01" max="2013-09-02">

    <label for="end_time">End time:</label>
    <input type="time" name="end_time"> <br>

    <!-- Amount -->
    <label for="min_amount">Minimum Amount (&#128;)</label>
    <input type="number" name="min_amount" min="0" step="50">

    <label for="max_amount">Maximum Amount (&#128;)</label>
    <input type="number" name="max_amount" min="0" step="50"> <br>

    <!-- Predicted Probability -->

    <label for="min_probability">Minimum Probability</label>
    <input type="number" name="min_probability" min="0" max="1" step="0.05">

    <label for="max_probability">Maximum Probability</label>
    <input type="number" name="max_probability" min="0" max="1" step="0.05"> <br>

    <!-- Decided Class -->

    <span>Class:</span>
    <input type="checkbox" name="fraud" data-classification = 1 checked>
    <label for="fraud">Fraud (1)</label>

    <input type="checkbox" name="non_fraud" data-classification = 0 checked>
    <label for="non_fraud">Non-fraud (0)</label>

    <input type="checkbox" name="undecided" data-classification = nan checked>
    <label for="undecided">Undecided (Nan)</label><br>

    <!-- Submit form button -->
    <input type="submit" value="Filter">
</form>
```

I've constrained the range of possible values to between 0 and 1 using the min and max attributes for the probability input elements. Minimum is 0 for amount so that users cannot enter minus values. For now we are not going to add form validation, we are going to focus on making a successful request and receiving the correct response.

This is the form the above HTML renders:



Method justification

Unlike using the manual HTTP request with the fetch API, where actually had to do a lot of work to create a valid request from input data, the form API does most of the work for us, especially as a successful response will send us back to the current page but just with a change in table data.

Handling request data

Since the filter with render request is mostly the same as the request, in the ‘manage’ view, we must check if the request has a query string attached to the URL. We can accomplish this by obtaining the arguments as a dictionary and seeing if it has any keys. If it has no keys Python will class it as a falsy value and the if statement will not be run.

The first transaction attribute I will create the filter for is ‘Decided_Class’.

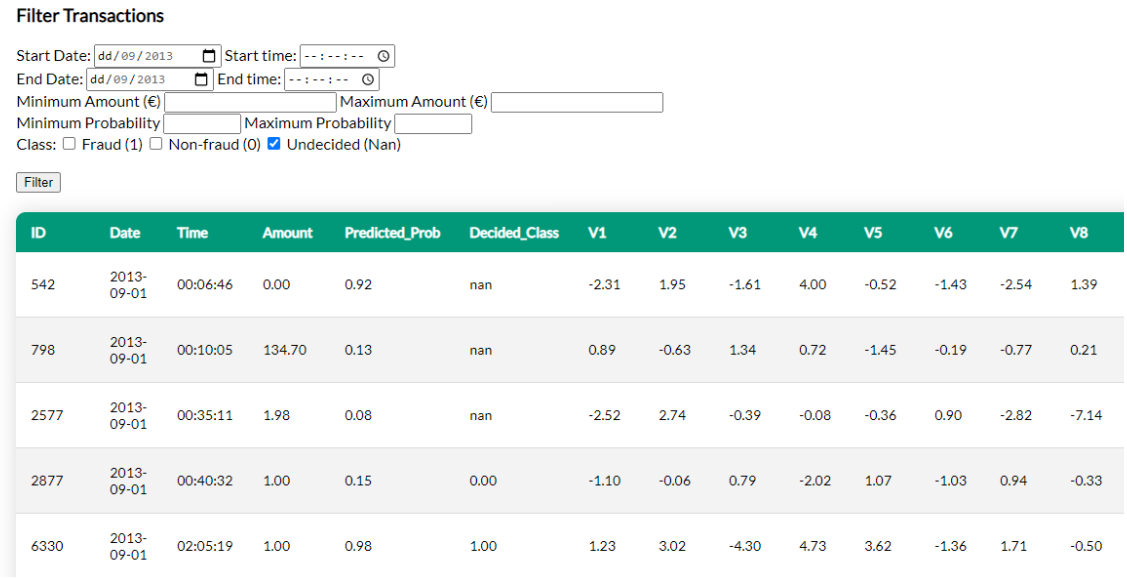
```python
@app.route('/manage')
def manage():
    # bring in pre-tested dataset
    filteredDB = pd.read_csv("dataset/decisionDB.csv")

    # test to see if querystring loaded in
    arguments = request.args.to_dict()
    if arguments:

        # filter out unwanted clasifcations
        if 'fraud' not in arguments:
            filteredDB = filteredDB[filteredDB["Decided_Class"] != 1]

        if 'non_fraud' not in arguments:
            filteredDB = filteredDB[filteredDB["Decided_Class"] != 0]

        if 'undecided' not in arguments:
            filteredDB = filteredDB[filteredDB["Decided_Class"] != numpy.nan]

    # get the columns in the order to be presented
    columnsToDisplay = ['ID', 'Date', 'Time', 'Amount', 'Predicted_Prob', 'Decided_Class'] + [f'V{num}' for num in range(1, 29)]
    filteredDB = filteredDB[columnsToDisplay]

    # render table and manage page
    return render_template('manage.html', row_data = list(filteredDB.values.tolist()), table_headings = filteredDB.columns.values)
```

Testing success

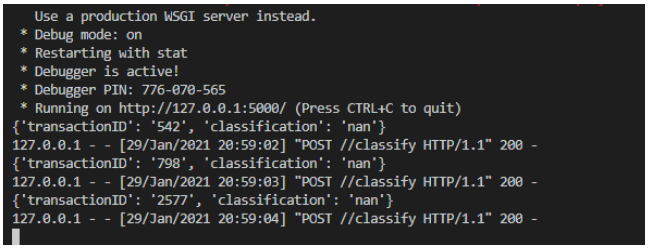
Let's test whether this worked. To do this I am going to assign the classification of the first 3 records to nan. All three records should be present when we only filter by nan. Note: all other queries will be left blank, they will be sent as an empty string in the request parameters however they will not be processed.

The screenshot below shows what the 'manage' table looks like after assigning their classification to undecided (nan) and before making the undecided class filter request. And the screenshot below that shows that the server received these requests to change their classifications for the purpose of testing.

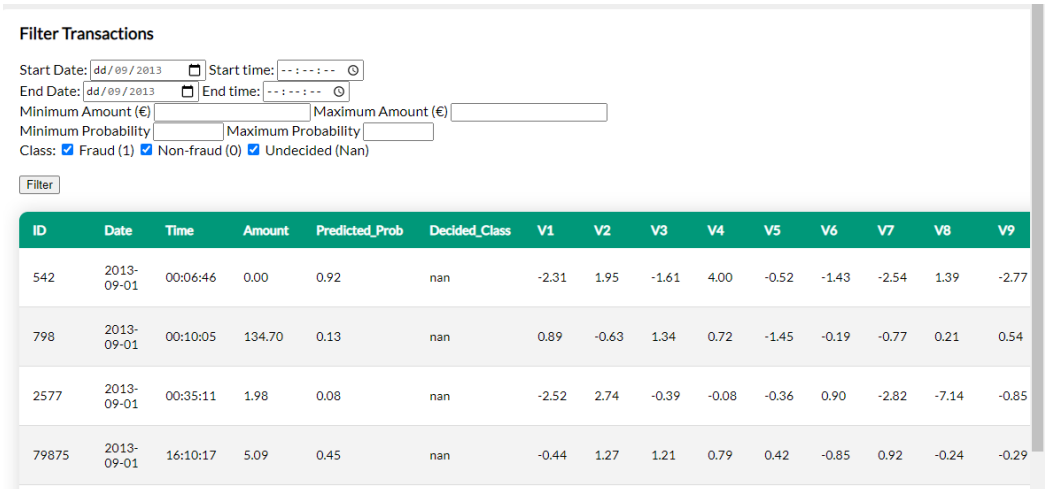
Also lets track the IDs of these three test records: 542, 798 and 2577 in the results.



ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2	V3	V4	V5	V6	V7	V8
542	2013-09-01	00:06:46	0.00	0.92	nan	-2.31	1.95	-1.61	4.00	-0.52	-1.43	-2.54	1.39
798	2013-09-01	00:10:05	134.70	0.13	nan	0.89	-0.63	1.34	0.72	-1.45	-0.19	-0.77	0.21
2577	2013-09-01	00:35:11	1.98	0.08	nan	-2.52	2.74	-0.39	-0.08	-0.36	0.90	-2.82	-7.14
2877	2013-09-01	00:40:32	1.00	0.15	0.00	-1.10	-0.06	0.79	-2.02	1.07	-1.03	0.94	-0.33
6330	2013-09-01	02:05:19	1.00	0.98	1.00	1.23	3.02	-4.30	4.73	3.62	-1.36	1.71	-0.50



Lets click the filter button and apply the filter. Result below:



ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2	V3	V4	V5	V6	V7	V8	V9
542	2013-09-01	00:06:46	0.00	0.92	nan	-2.31	1.95	-1.61	4.00	-0.52	-1.43	-2.54	1.39	-2.77
798	2013-09-01	00:10:05	134.70	0.13	nan	0.89	-0.63	1.34	0.72	-1.45	-0.19	-0.77	0.21	0.54
2577	2013-09-01	00:35:11	1.98	0.08	nan	-2.52	2.74	-0.39	-0.08	-0.36	0.90	-2.82	-7.14	-0.85
79875	2013-09-01	16:10:17	5.09	0.45	nan	-0.44	1.27	1.21	0.79	0.42	-0.85	0.92	-0.24	-0.29

All three transactions with the IDs 542, 798 and 2577 are present in the results. Success!

Proof that the server processed this request and this filtering wasn't client-side manipulation:

```
 * Debugger is active!
 * Debugger PIN: 776-070-565
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [29/Jan/2021 21:35:47] "GET /manage?start_date=&start_time=&end_date=&end_time=&min_amount=&max_amount=&min_probability=&max_probability=&undecided=on HTTP/
1.1" 200 -
```

Chapter 34: Performing more complex queries. - (Objective 4*)

Filtering by amount (€)

If no parameters are entered for min and max amount in our filter request, then we will not filter any records using this field.

However if either min or max is not null, then we need to manually fix the range for the value not entered. E.g if only a max value is entered, then we assume there is no minimum so it is set to the smallest value (0). But if a max value is not entered, there is no type of value to represent infinity numerically. However when we convert the string “inf” to a floating point number, Python generates an unbounded upper value for comparison operations.

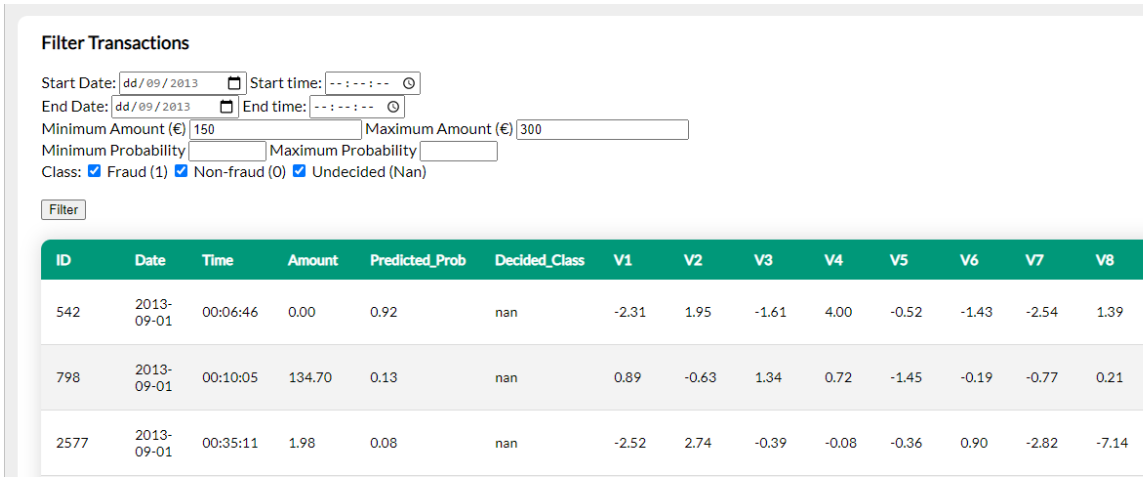
This is useful because it lets us fix a lower and upper bound so we can perform the filtering in one operation rather than in multiple cases which is more difficult to implement. In the previous chapter, we created a local variable called arguments for the the args object passed in the request which we converted into a dictionary. We utilise this variable to save our bounds into it without creating redundant new variables to store this same data.

The need for the float () function is that the values in the request are passed as strings which would throw a type comparison error in during the filtering. Finally like in the last chapter, we replace our current filteredDB with the new filter applied which removes any transactions which failed the query.

```
# check if amount filter enetered
if arguments["min_amount"] != "" or arguments["max_amount"] != "":

    # if min or max amount isn't present:
    # set it to the start or end point of range of possible values

    if arguments["min_amount"] == "":
        arguments["min_amount"] = 0

    elif arguments["max_amount"] == "":
        arguments["max_amount"] = "inf"

    min_amount = float(arguments["min_amount"])
    max_amount = float(arguments["max_amount"])

    # filter by amount
    filteredDB = filteredDB[filteredDB["Amount"].between(min_amount, max_amount, inclusive = True)]
```

Test 1

Lets filter all transactions between €150 and €300.

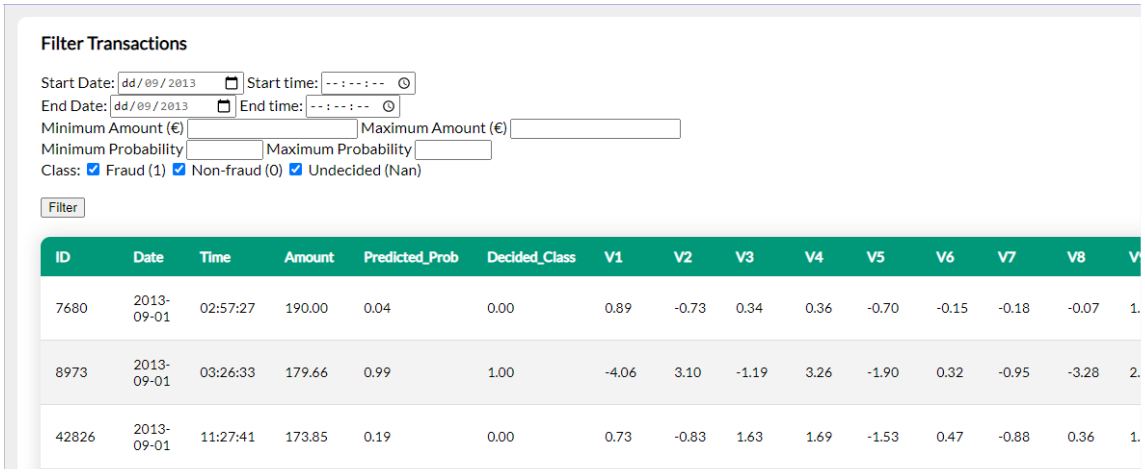


Filter Transactions

Start Date: dd/09/2013 Start time: --:--:-- End Date: dd/09/2013 End time: --:--:--
Minimum Amount (€) 150 Maximum Amount (€) 300
Minimum Probability Maximum Probability
Class: Fraud (1) Non-fraud (0) Undecided (Nan)

Filter

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2	V3	V4	V5	V6	V7	V8
542	2013-09-01	00:06:46	0.00	0.92	nan	-2.31	1.95	-1.61	4.00	-0.52	-1.43	-2.54	1.39
798	2013-09-01	00:10:05	134.70	0.13	nan	0.89	-0.63	1.34	0.72	-1.45	-0.19	-0.77	0.21
2577	2013-09-01	00:35:11	1.98	0.08	nan	-2.52	2.74	-0.39	-0.08	-0.36	0.90	-2.82	-7.14



Filter Transactions

Start Date: dd/09/2013 Start time: --:--:-- End Date: dd/09/2013 End time: --:--:--
Minimum Amount (€) Maximum Amount (€)
Minimum Probability Maximum Probability
Class: Fraud (1) Non-fraud (0) Undecided (Nan)

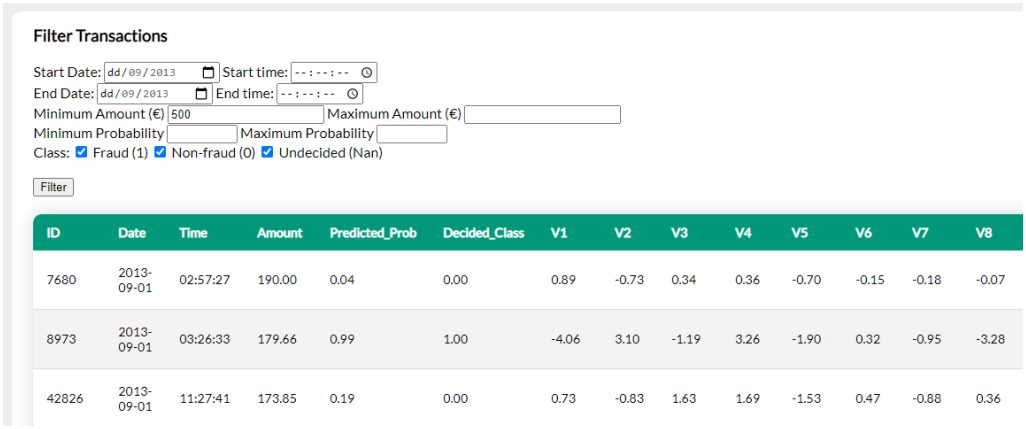
Filter

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2	V3	V4	V5	V6	V7	V8	V
7680	2013-09-01	02:57:27	190.00	0.04	0.00	0.89	-0.73	0.34	0.36	-0.70	-0.15	-0.18	-0.07	1.
8973	2013-09-01	03:26:33	179.66	0.99	1.00	-4.06	3.10	-1.19	3.26	-1.90	0.32	-0.95	-3.28	2.
42826	2013-09-01	11:27:41	173.85	0.19	0.00	0.73	-0.83	1.63	1.69	-1.53	0.47	-0.88	0.36	1.

Amounts in transactions are all between that range. Success.

Test 2

Lets filter all transactions with a minimum amount of €500.

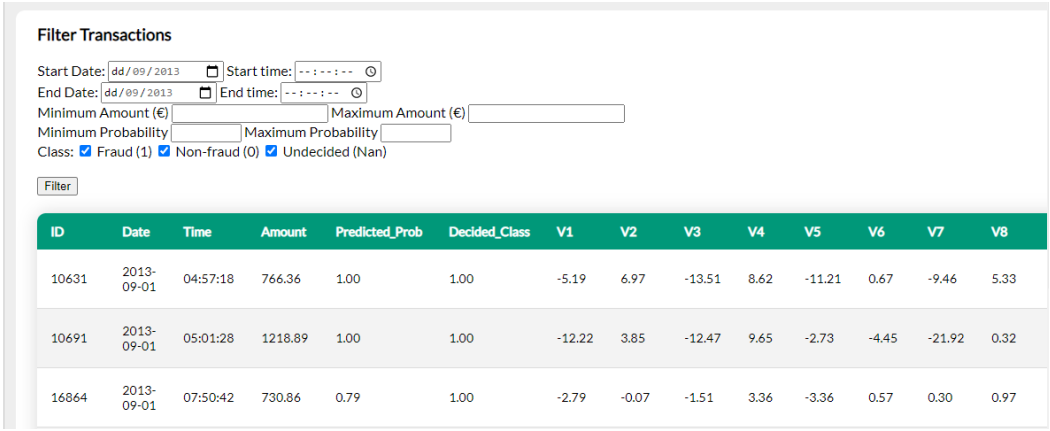


Filter Transactions

Start Date: dd/09/2013 Start time: --:--:--
End Date: dd/09/2013 End time: --:--:--
Minimum Amount (€) 500 Maximum Amount (€)
Minimum Probability Maximum Probability
Class: Fraud (1) Non-fraud (0) Undecided (Nan)

Filter

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2	V3	V4	V5	V6	V7	V8
7680	2013-09-01	02:57:27	190.00	0.04	0.00	0.89	-0.73	0.34	0.36	-0.70	-0.15	-0.18	-0.07
8973	2013-09-01	03:26:33	179.66	0.99	1.00	-4.06	3.10	-1.19	3.26	-1.90	0.32	-0.95	-3.28
42826	2013-09-01	11:27:41	173.85	0.19	0.00	0.73	-0.83	1.63	1.69	-1.53	0.47	-0.88	0.36



Filter Transactions

Start Date: dd/09/2013 Start time: --:--:--
End Date: dd/09/2013 End time: --:--:--
Minimum Amount (€) Maximum Amount (€)
Minimum Probability Maximum Probability
Class: Fraud (1) Non-fraud (0) Undecided (Nan)

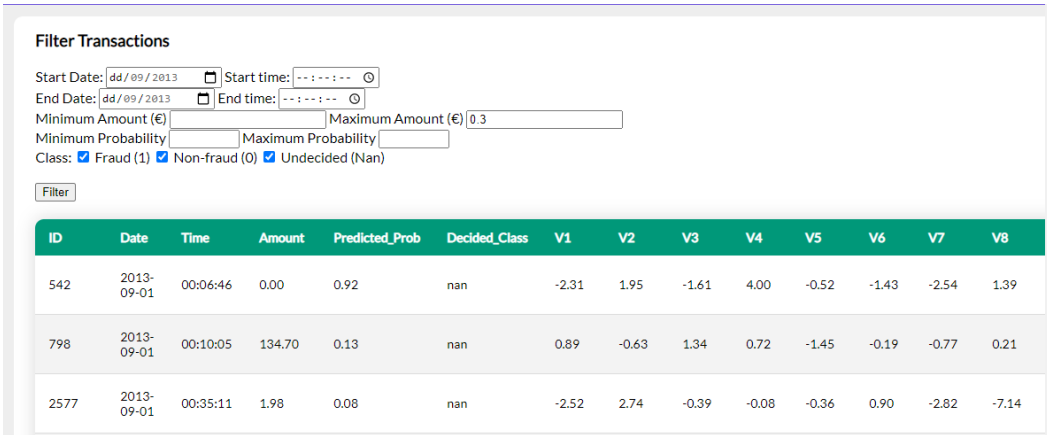
Filter

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2	V3	V4	V5	V6	V7	V8
10631	2013-09-01	04:57:18	766.36	1.00	1.00	-5.19	6.97	-13.51	8.62	-11.21	0.67	-9.46	5.33
10691	2013-09-01	05:01:28	1218.89	1.00	1.00	-12.22	3.85	-12.47	9.65	-2.73	-4.45	-21.92	0.32
16864	2013-09-01	07:50:42	730.86	0.79	1.00	-2.79	-0.07	-1.51	3.36	-3.36	0.57	0.30	0.97

Success.

Test 3

Lets filter all transactions with a maximum amount of €0.30.

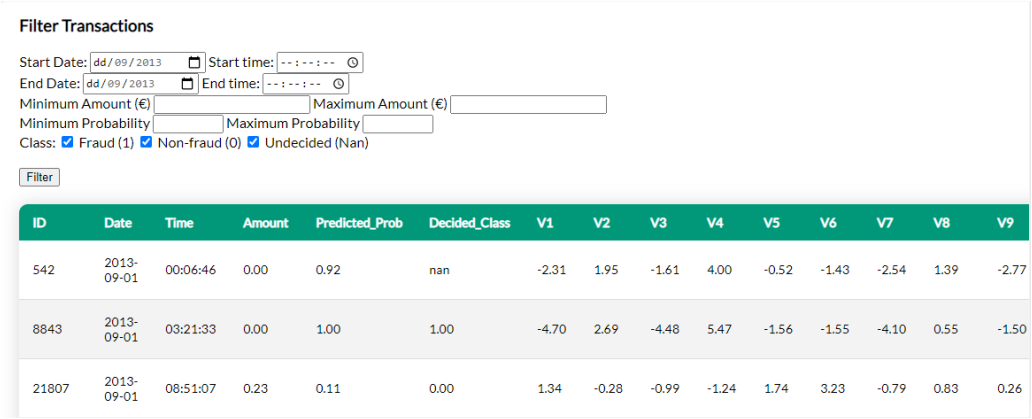


Filter Transactions

Start Date: dd/09/2013 Start time: --:--:--
End Date: dd/09/2013 End time: --:--:--
Minimum Amount (€) Maximum Amount (€) 0.3
Minimum Probability Maximum Probability
Class: Fraud (1) Non-fraud (0) Undecided (Nan)

Filter

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2	V3	V4	V5	V6	V7	V8
542	2013-09-01	00:06:46	0.00	0.92	nan	-2.31	1.95	-1.61	4.00	-0.52	-1.43	-2.54	1.39
798	2013-09-01	00:10:05	134.70	0.13	nan	0.89	-0.63	1.34	0.72	-1.45	-0.19	-0.77	0.21
2577	2013-09-01	00:35:11	1.98	0.08	nan	-2.52	2.74	-0.39	-0.08	-0.36	0.90	-2.82	-7.14



Success.

Filtering by Predicted Fraudulency Probability

Like probability we are dealing with a continuous range of values so can repeat exactly the same method but instead replacing the max to 1.0 instead of infinity.

```
# check if probability filter enetered
if arguments["min_probability"] != "" or arguments["max_probability"] != "":

    if arguments["min_probability"] == "":
        arguments["min_probability"] = 0

    elif arguments["max_probability"] == "":
        arguments["max_probability"] = 1.0

    min_probability = float(arguments["min_probability"])
    max_probability = float(arguments["max_probability"])

    # filter by probability
    filteredDB = filteredDB[filteredDB["Predicted_Prob"].between(min_probability, max_probability, inclusive = True)]
```

No need to retest this functionality.

Tests added for post-development phase

No	Type	What is being tested?	Input / Action Taken	Expected Outcome
Filtering Transactions				
14.1	Valid	Search for transactions with fraudulent probability between 0.75	Max prob: 0.75 and min prob: 0.90.	All transactions with fraudulent probability between 0.75 and 0.90 are displayed to the

		and 0.90.		table.
14.2	Valid	Search for all transactions with an amount more than €500.00.	Min amount: 500.	All transactions with an amount more than €500.00 are displayed to the table.
14.3	Valid	Search for all undecided classification transactions.	Uncheck fraud and non-fraud.	All transactions that are of the undecided classification are displayed in the table.

Chapter 35: Filtering by date and time. - (Objective 4*)

Filtering by date and time is a more complex operation. Lets pose three example scenarios to demonstrate this:

1. The fraud analyst is doing a general review of high risk transactions over the last two days and enters the 1st and 2nd of September 2013 as the start and end dates.
2. The analyst has observed that fraudulent activity peaks at midday and wants to filter out any transactions not between 11:50 and 12:30 for every day in the dataset. The analyst would not enter a start nor end date but is still making a valid filter request.
3. The fraud analyst was notified that after the 1st of September 2013, many false positives were falsely stopped between 21:45 and 22:00. There would be no end date in that request as that is assumed to be the present (or the latest in the dataset).

In conclusion, when the analyst entered different combinations of inputs, it implied the type of logical filtering operation they wanted. In Examples 1 and 2, the analyst wanted to search a set daily interval of time (e.g. from 11:50 - 12:30) over multiple selected days. And in Example 3 the analyst requires to see all transactions over a continuous period of time (e.g the 1st of September to the end of the dataset).

Below I created a table to demonstrate which combinations of inputs would map to the type of datetime filter used. For the input columns, 1 represents a valid input present and 0 (blank) represents no input present and for the output column, 1 represents using the daily interval search and 0 represents the continuous period search.

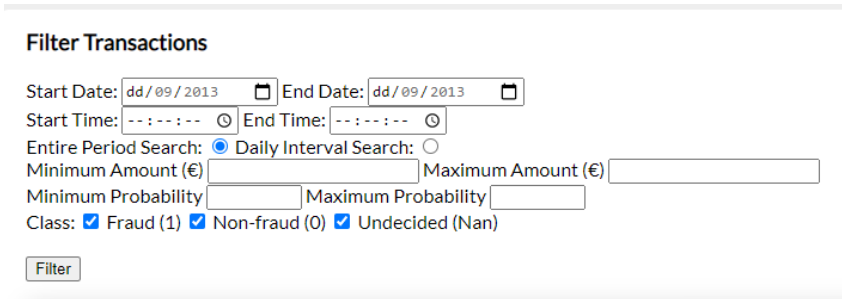
Start Time, T1	End Time, T1	Start Date, D1	End Date, D2	**Output**
		1	1	**0**
		1		**0**
			1	**0**
1		1		**0**
	1		1	**0**
1				**1**
	1			**1**
1	1			**1**
1	1	1		**1**
1	1		1	**1**

However I can list other combinations where it is not contextually clear which inputs map to which inputs map to which output:

Start Time, T1	End Time, T1	Start Date, D1	End Date, D2	**Output**
1	1	1	1	**0 OR 1**
	1	1	1	**0 OR 1**
1		1	1	**0 OR 1**

These possibilities are clearly quite convoluted and are open to interpretation. Therefore we are going to let the user choose which type of search they desire using a radio button on the form. Here's the code for this:

```html
<!-- Start Date Time -->

<label for="start_date">Start Date:</label>
<input type="date" name="start_date" min="2013-09-01" max="2013-09-02">

<label for="end_date">End Date:</label>
<input type="date" name="end_date" min="2013-09-01" max="2013-09-02"> <br>

<label for="start_time">Start Time:</label>
<input type="time" name="start_time" step="1">

<label for="end_time">End Time:</label>
<input type="time" name="end_time" step="1"> <br>

<!-- Select the type of datetime range search -->

<label for="entire_period">Entire Period Search:</label>
<input type="radio" name="time_search_type" value="entire_period" checked>

<label for="daily_interval">Daily Interval Search:</label>
<input type="radio" name="time_search_type" value="daily_interval"> <br>
```



First we are going to check if the user wants to filter by date/time and then prepare incoming data for either one of these searches.

Checking for data

If a user wants to filter by datetime, they will have entered any data into the start/end date/time fields. We are not checking if they have selected a type of search because one of the radio buttons will always be selected by default.

```
# filter by datetime

# check if any datetimes filters enetered
if arguments["start_date"] != '' or arguments["end_date"] != '' or arguments["start_time"] != '' or arguments["end_time"]:

    # variably set the start and end of the dataset
    dataset_start, dataset_end = "2013-09-01", "2013-09-02"

    # if min or max amount isn't present - set it to the start or end point of range of possible values
    if arguments["start_date"] == '':
        arguments["start_date"] = dataset_start # start of dataset

    if arguments["end_date"] == '':
        arguments["end_date"] = dataset_end # start of dataset

    if arguments["start_time"] == '':
        arguments["start_time"] = "00:00:00" # start of day

    if arguments["end_time"] == '':
        arguments["end_time"] = "23:59:59" # end of day

    # create a new temporary datetime column for filtering - needed in both searches
    filteredDB["Datetime"] = pd.to_datetime(filteredDB['Date'] + ' ' + filteredDB['Time'])
```

Similar to other fields, blank fields in the presence of valid start or end fields can be assumed to be the start or end values of the range of dates or times. The pandas filter by datetime method requires we have a start and end datetime to filter by. We assign these changes back to the arguments dict and there will be no blank fields to corrupt the search.

If the datasets start or end dates change we can change it once and be sure all functionality will update inline with this change. Although we can't define constants in Python, I'm going to declare two variables to act as such. I only had to do this for the dataset start/end dates since the times "00:00:00" and "23:59:59" will always be the start and end times of any day. We are also going to add a Pandas own data type of datetime to every record by concatenating each record's date and time. This lets Pandas own filtering methods to operate on the data frame without having to implement our own.

The continuous period search

The radio input collects the type of search type - this is the argument it passes in its request. Because this search type filters a continuous range of times between two timestamps, it is simple in nature and can be executed using a mask.

We use the start/end datetimes arguments to create a start and an end timestamps and convert them into Pandas own type of timestamp so that they are also valid operands for the comparison operations (≥ and ≤) in the filtering mask.

```
# use the correct search type.
if arguments["time_search_type"] == "entire_period":

    # create the start and end datetimes of the period we want to gett
    startDtStr = arguments["start_date"] + ' ' + arguments["start_time"]
    start_datetime = pd.to_datetime(startDtStr)

    endDtStr = arguments["end_date"] + ' ' + arguments["end_time"]
    end_datetime = pd.to_datetime(endDtStr)

    # create mask to filter between two datetimes
    mask = (start_datetime <= filteredDB["Datetime"]) & (filteredDB['Datetime'] <= end_datetime)

    # filter by datetime
    filteredDB = filteredDB[mask]
```

The daily interval search

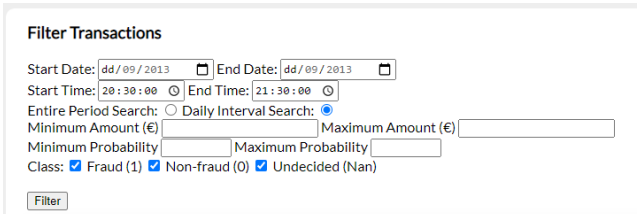
Our first step is to solely use the start/end dates to remove any unwanted dates using a mask, then we create a timeIndex so we can query any transactions within a given time interval every day.

```
else: # daily interval search

    # first filter out the dates we're not interested in
    start_date, end_date = arguments["start_date"], arguments["end_date"]

    #greater than the start date and smaller than the end date
    mask = (start_date <= filteredDB["Date"] ) & (filteredDB["Date"] <= end_date)

    # get rid of unwanted days
    filteredDB = filteredDB[mask]

    start_time, end_time = arguments["start_time"], arguments["end_time"]

    # create an index of times Pandas can look at.
    timeIndex = pd.DatetimeIndex(filteredDB['Datetime'])

    # filter by the daily interval of time
    filteredDB = filteredDB.iloc[timeIndex.indexer_between_time(start_time, end_time)]
```

Now let's test if these two search types have been implemented correctly.

Test 1

The first test will search for all transactions between 20:30 and 21:30 across all days of the dataset.



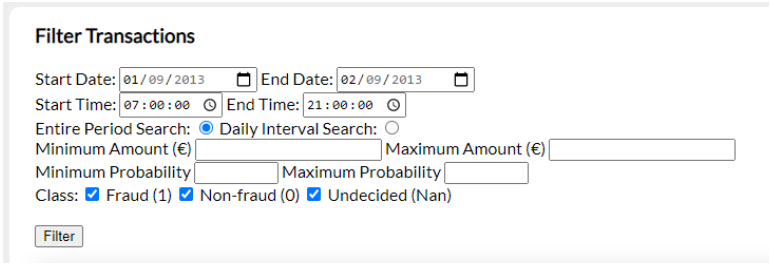
Here's part of a screenshot of the result:

123271	2013-09-01	21:21:07	1.00	0.93	1.00	1.08	1.09	-1.37
123302	2013-09-01	21:21:16	1.00	0.96	1.00	-1.30	1.08	-0.18
124037	2013-09-01	21:25:54	129.00	0.32	0.00	-0.72	0.61	1.16
262561	2013-09-02	20:35:37	4.69	1.00	1.00	0.57	3.31	-6.63
262777	2013-09-02	20:37:19	30.00	0.32	0.00	-0.63	-0.39	2.03
263085	2013-09-02	20:39:52	300.00	0.09	0.00	1.37	-1.21	-0.69

Test successful.

Test 2

The second test will search for all transactions starting at 7am on the 1st of September and 9pm on the 2nd of September.



Here's the resulting list of selected transactions:

- [Start]: first transaction on the 1st of September.

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2
14198	2013-09-01	07:00:31	99.99	1.00	1.00	-16.60	10.54
14212	2013-09-01	07:00:54	99.99	1.00	1.00	-17.28	10.82

....

- [End]: last transaction 20:54 on the 2nd of September.

264318	2013-09-02	20:49:30	10.59	0.13	0.00	1.89	0.30
264942	2013-09-02	20:54:22	13.99	0.06	0.00	-2.89	2.96

Test Successful.

Tests added for post-development phase

No	Type	What is being tested?	Input / Action Taken	Expected Outcome
Filtering Transactions				
14.4	Valid	Perform a daily interval search to search for all transactions between 13:00 and 14:00 across the dataset.	Daily interval checked. Start time: 13:00. End time: 14:00.	All transactions between 13:00 - and 14:00 on both days are displayed in the table.
14.5	Valid	Perform a periodic search to find all transactions between 21:00 - 01/09/2013 to 3:00 - 02/09/2013.	Start time: 21:00. Start date: 01/09/2013 End time: 03:00. End date: 02/09/2013.	All transactions between 21:00 - 01/09/2013 to 3:00 -02/09/2013. are displayed in the table.

Chapter 36: Form validation. - (Objective 4*)

In this chapter we use client-side processing to ensure that a valid filter request is sent.

Our first step is to create a paragraph element so that we can add any error messages thrown as text that the user can read. Its inner text will initially remain empty until an error is triggered and it can be added/removed by the JavaScript DOM when necessary. Here's the HTML code for it:

```
    <!-- Submit form button -->
    <input type="submit" value="Filter">
</form>

<p id="errorMessage"></p>
```

Next we need to create the form validation function. We need to create an array for error messages to be added to, and we need to also obtain a list of the input elements in the form.

In the body of the function triggered, our aim is to scour all fields and find logical errors in the data inputted. We do not need to look for data type errors thanks to the way we configured our inputs so that they have less freedom to put invalid values.

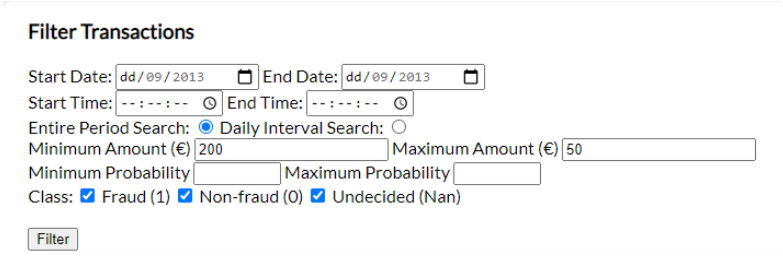
An example of a logical error could be a minimum transaction amount being greater than a maximum probability. This will be the first logical error of its nature we will tackle. This is how we will detect it: check if a minimum and maximum amount have in fact been entered then check if the minimum is greater than the maximum. If these two conditions are met, append the error message to the array of error messages.

Below is the code which describes this:

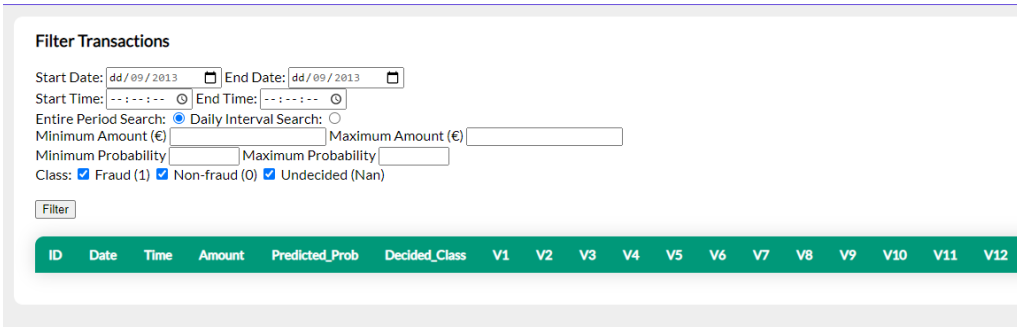
```
const errorEl = document.querySelector('#errorMessage')

function validate () {
    let inputs = document.forms[0].elements
    let errors = []

    if (inputs.min_amount.value && inputs.max_amount.value) {
        // throw an error if minimum value is more than maximum for our range functions
        if (inputs.min_amount.value > inputs.max_amount.value) {
            errors.push('Maximum amount value in range cannot be smaller than the minimum.')
        }
    }
    if (errors.length > 0) {
        errorEl.innerHTML = errors.join('<br>') // print errors to screen
        return false // stop form from submitting
    } else {
        return true
    }
}
```

```
<!-- Filter Table Form -->
<form action="/manage" method="GET" id="filter-table-form" onsubmit="return validate()">
```

Let's test if this worked by entering a minimum amount of €200.00 and a maximum amount of €50.00.



And here's the results:



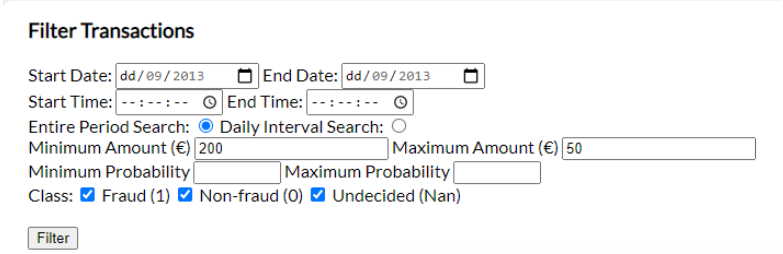
Test failed. The purpose of form validation is to stop invalid or redundant requests. Instead it submitted regardless and wasted server processing on a void request.

Why did it fail? The purpose of the onsubmit method is to stop the action if the function returns false.

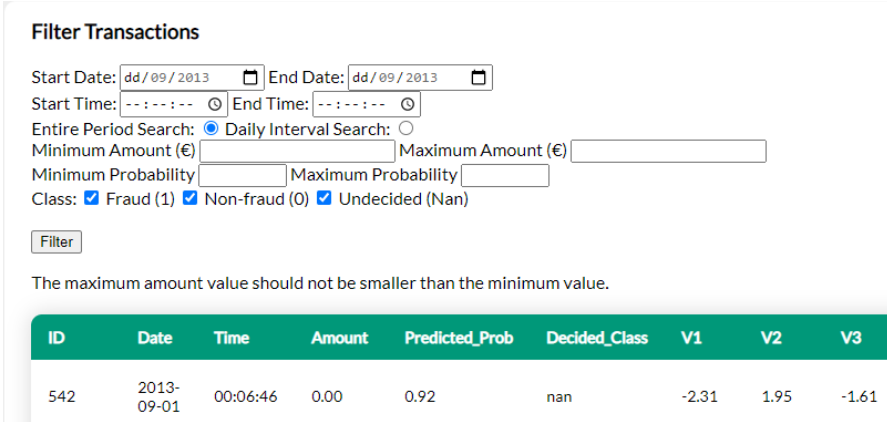
How am I going to make it work? By using an event handler instead. After doing some reading online I learned that the submit event has a special method called preventDefault() which should stop it from preventing the form submitting.

```
document.getElementById('filterTableForm').addEventListener ("submit", e => {

    const inputs = e.target.elements
    let errors = []

    // throw an error if minimum value is more than maximum for our range functions
    if (inputs["min_amount"].value > inputs["max_amount"].value) {
        errors.push('The maximum amount value should not be smaller than the minimum value.')
    }

    if (errors.length > 0) {
        errorEl.innerHTML = errors.join('<br>') // print errors to screen
        e.preventDefault() // stop form from submitting
        e.target.reset() // reset the inputs
    }

});
```

Let's repeat the previous test:



Here's the results:



Test **Successful**. Error message prevented void request.

Validating by Predicted Fraud Probability and datetime.

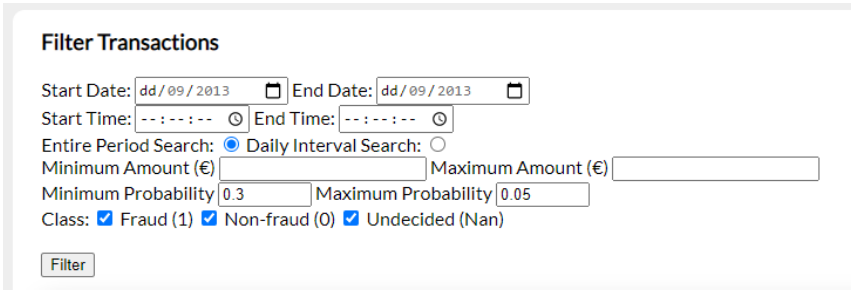
Like the min/max amount fields we only need ensure the minimum isn't greater than the maximum. But for date and time the nature of the checks is slightly more complex like the operation itself. We need to perform two separate checks:

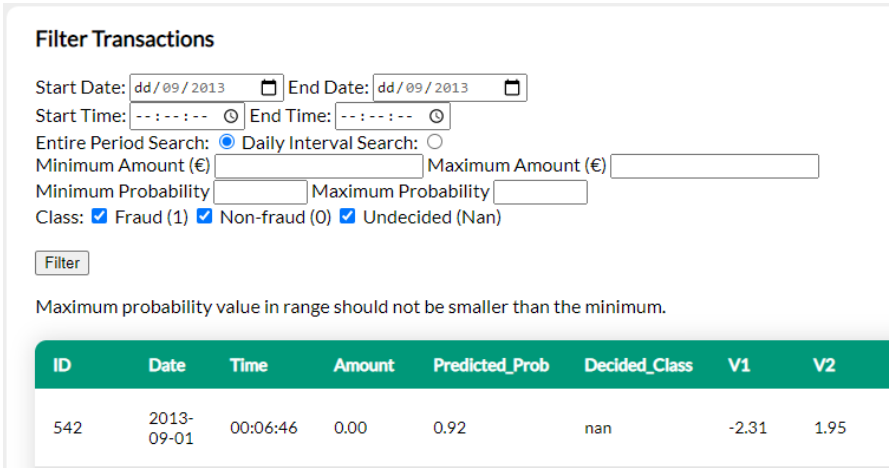
1. If two dates are entered. The start date must be before the end date in time.
2. Then if the dates check passes and two times are also entered. The end datetime must be after the start datetime if it is a period search.

```
// throw an error if minimum value is more than maximum for our range functions
if (inputs["min_probability"].value > inputs["max_probability"].value) {
    errors.push('Maximum probability value in range should not be smaller than the minimum.')
}

// first check: end date must be equal to or larger than start date
if (Date.parse(inputs["start_date"].value) > Date.parse(inputs["end_date"].value)) {
    errors.push("The starting date should equal to or before the end date.")
}
// second check: if end date = start date then the end datetime must be after the start datetime
else if (inputs["time_search_type"].value == "entire_period" && inputs["start_date"].value == inputs["end_date"].value
        && inputs["start_time"].value && inputs["end_time"].value) {

    // create the timestamps
    let date = inputs.start_date.value
    let start_timestamp = Date.parse(date + ' ' + inputs.start_time.value)
    let end_timestamp = Date.parse(date + ' ' + inputs.end_time.value)

    // check if the start timestamp is greater than the end timestamp
    if (start_timestamp > end_timestamp) {
        errors.push("The end datetime must not be before the starting datetime.")
    }
}
```

Test 1

Min probability is greater than max probability. Should throw an error.

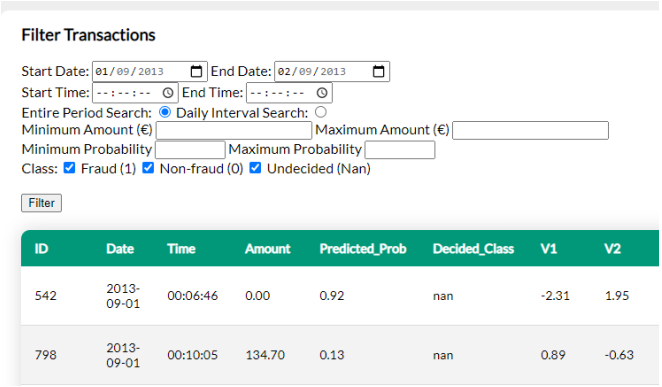


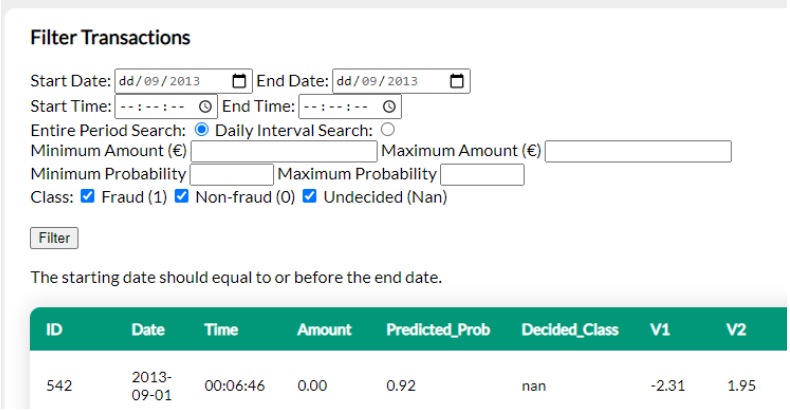


Test **successful.** Appropriate Error message produced.

Test 2

We're going to test if the first datetime check works as expected by entering the end date as the 1st and start date as the 2nd of September.

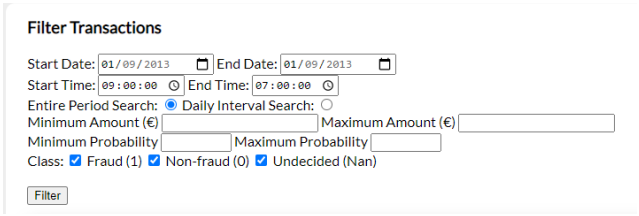


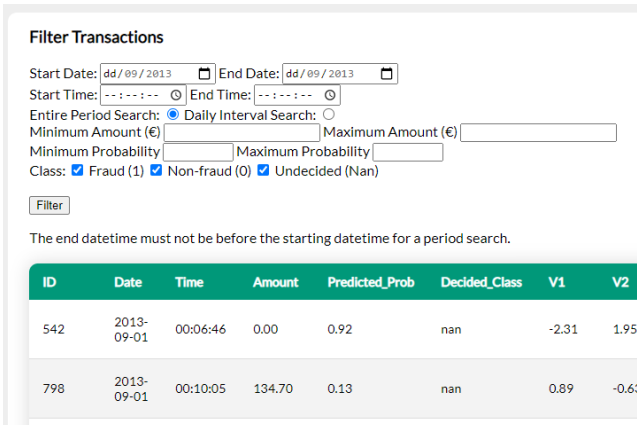


Test **successful.** Appropriate Error message produced.

Test 3

In our last test, we are going to perform a period search on a particular day (start_date = end_date). However the end time is going to be before the start datetime. Specifically, we will be attempting to search the 1st of September between the times start_time = 9:00 and end_time = 7:00.





Test **successful.** Appropriate Error message produced.

We have now carried out all the requests of the stakeholder. Lets get feedback on these features.

Tests added for post-development phase

No	Type	What is being tested?	Input / Action Taken	Expected Outcome
Filtering Transactions				
14.6	Invalid / Absent	No classification categories are attempted to be selected.	Mouse clicks to uncheck all classifications.	A message should pop up warning the user of this and cancel the filter request.
14.7	Invalid	Minimum probability is	Max prob: 0.45 and min prob:	A message should pop up warning the user of

		greater than the maximum probability causes error.	0.10.	this and cancel the filter request.
14.8	Invalid	Minimum amount is greater than the maximum amount causes the error.	Min amount: 50 Max amount: 10.	A message should pop up warning the user of this and cancel the filter request.
14.9	Invalid	Start date is after the end date entered causes error.	E.g start date: 02/09/2013 and end date: 01/09/2013.	A message should pop up warning the user of this and cancel the filter request.
14.10	Invalid	Min probability is greater than the max probability and start date is after the end date entered. Causes multiple error messages.	E.g start date: 02/09/2013 and end date: 01/09/2013. Max prob: 0.10 and min prob: 0.40.	Multiple warning messages should pop up and cancel the filter request.

Stakeholder Alpha Testing

Alpha Testing simulates the targeted environment for the software product along with the real-time users actions. The primary purpose of this is to detect defects, especially in design and functions at an early stage of testing. The secondary purpose of this in our case is to see if the Stakeholder's requirements have been met.

Yorke is the stakeholder and target audience for our end product, so I sent him the most recent version of our project which contained all the features he requested. I asked him to carry out the following tasks to demonstrate how the analyst would interact with the web application's new feature and report back with screenshots via email as evidence alongside explanations to clarify his results.

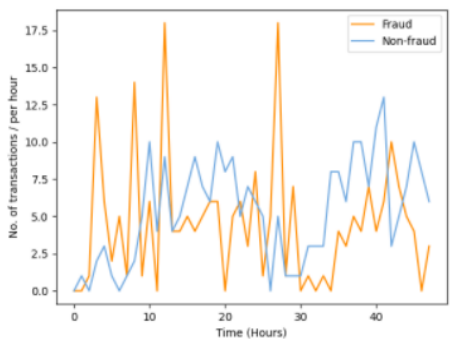
Alpha Testing Evidence

I listed 5 complex tasks which encompassed the usage of all new features and core functionality of the application to demonstrate it hasn't worsened during development. The evidence will be laid out below in a question-response transcript style. I also instructed Yorke to record any and every error encountered in the notes and with evidence.

Test 1

Task: Read off the time-series which hours had the highest concentration of fraudulent activity each day.

Response:



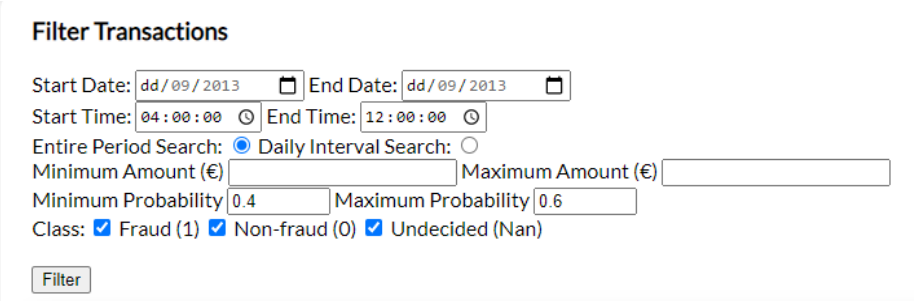
No. of fraudulent transactions per hour seems to to peak, after 11 and 28 hours i The past few hours trailing these peak also had high fraudulent activity.

So on the 1st of September at 11:00 and the 2nd of september at 4:00. Hence 4:0 12:00 seems to have the highest amount activity each day.

Test 2

Task: Make a daily interval search across the entire duration of the dataset for the interval of highest fraudulent activity and select transactions which has a predicted probability between 0.4 and 0.6.

Response: Filter Query:



Result:

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2
14339	2013-09-01	07:03:46	3.76	0.59	1.00	1.13	1.13
34327	2013-09-01	10:26:52	515.82	0.53	1.00	-1.44	-0.98
166538	2013-09-02	08:49:02	14.99	0.42	0.00	-0.57	-0.08
179385	2013-09-02	10:27:32	110.14	0.41	0.00	1.80	-0.25

4 transactions met the filter criteria and were displayed.

Test 3

Task: Classify at least 3 of these filtered transactions within 0.1 of the decision boundary (0.5) as undecided (NaN). Note down their transaction IDs.

Response:

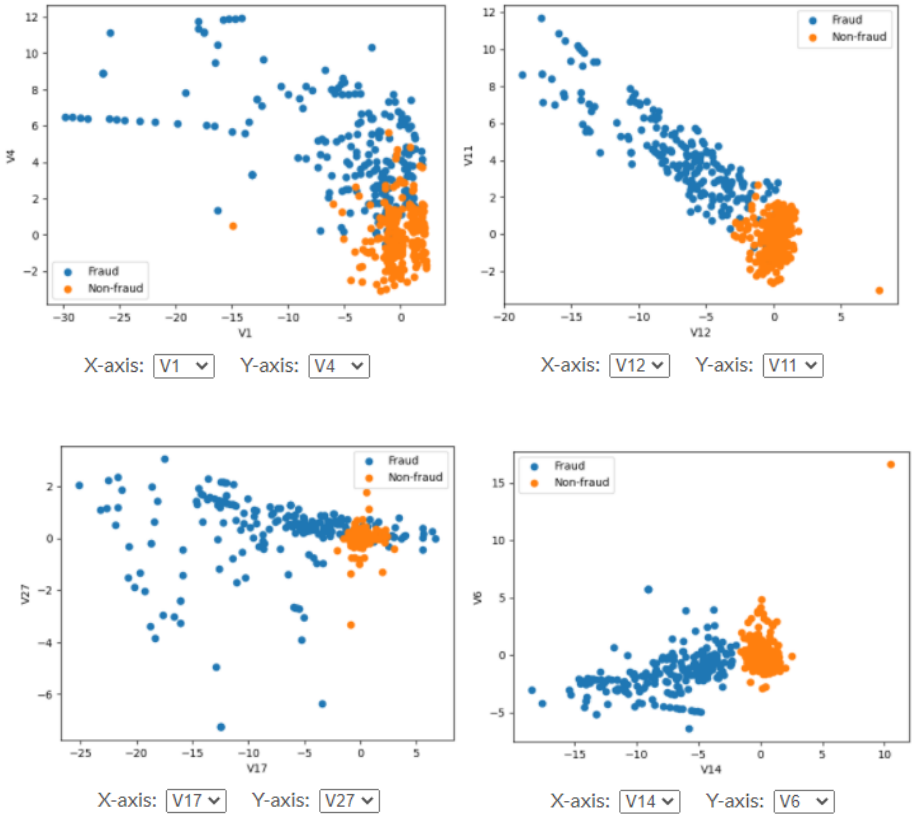
ID	Date	Time	Amount	Predicted_Prob	Decided_Class	V1	V2
14339	2013-09-01	07:03:46	3.76	0.59	nan	1.13	1.13
34327	2013-09-01	10:26:52	515.82	0.53	nan	-1.44	-0.98
166538	2013-09-02	08:49:02	14.99	0.42	nan	-0.57	-0.08
179385	2013-09-02	10:27:32	110.14	0.41	nan	1.80	-0.25

4 transactions which fit the filter criteria. The two fraudulent transactions had IDS 14339 and 34327. The non-fraudulent transactions had IDS 166538 and 17985.

Test 4

Task: First go to the dashboard and browse through different combinations of 2-variable plots and identify which ones show the most obvious fraudulent activity.

Response:



I chose four 2-var plots with different variables. V12 and V11 showed the highest clustering of fraudulency in the bottom right corner of the plot. Mean fraudulency had an average value of V12 = 0, V11 = 0.

Test 5

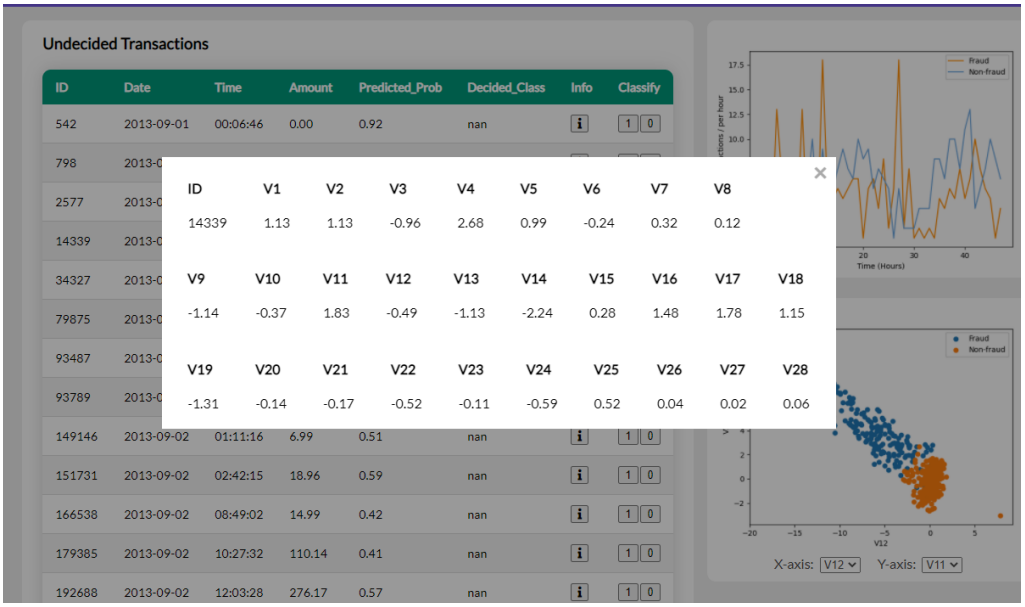
Task: On the dashboard page find the first transaction you classified as NaN in test 3 by using its ID. Click its info button to see its characteristics and obtain the values for the two variables we are interested in.

Response:

Undecided Transactions

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	Info	Classify
542	2013-09-01	00:06:46	0.00	0.92	nan	i	1 0
798	2013-09-01	00:10:05	134.70	0.13	nan	i	1 0
2577	2013-09-01	00:35:11	1.98	0.08	nan	i	1 0
14339	2013-09-01	07:03:46	3.76	0.59	nan	i	1 0
34327	2013-09-01	10:26:52	515.82	0.53	nan	i	1 0

Picture of when I found the transaction and before I opened the info window.



For the transaction with ID 14339, V11 = 1.83 and V12 = -0.49.

Test 6

Task: Finally, compare the average fraudulency var values with the transactions var values and determine the classification.

Response:

It seems that the nearest transaction to (-0.49, 1.83) is right on the border of being classified as fraudulent. Its two nearest neighbour data points are fraudulent. Therefore I am going to classify this transaction from undecided to fraudulent.

Undecided Transactions

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	Info	Classify
542	2013-09-01	00:06:46	0.00	0.92	nan	i	1 0
798	2013-09-01	00:10:05	134.70	0.13	nan	i	1 0
2577	2013-09-01	00:35:11	1.98	0.08	nan	i	1 0
14339	2013-09-01	07:03:46	3.76	0.59	1	i	1 0
34327	2013-09-01	10:26:52	515.82	0.53	nan	i	1 0

After refresh:

Undecided Transactions

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	Info	Classify
542	2013-09-01	00:06:46	0.00	0.92	nan	i	1 0
798	2013-09-01	00:10:05	134.70	0.13	nan	i	1 0
2577	2013-09-01	00:35:11	1.98	0.08	nan	i	1 0
34327	2013-09-01	10:26:52	515.82	0.53	nan	i	1 0

The record has now been classified and removed from the undecided transacti table.

Stakeholder Feedback Interview

During testing, Yorke reported back no errors, defects or bugs so the primary purpose of alpha testing has been fulfilled.

I am conducting this interview because I need to gain an insight into whether the stakeholder's requirements for the program has been met and if not, what the next steps should be to fulfill this and complete the iterative development of the program. I want to hear feedback on any issues he had using the program even if they didn't necessarily cause any bugs. After we complete these requirements, the development will be finished and the final product will be evaluated.

Transcript

Yorke was emailed all the questions prior to the interview to give him time to properly articulate his response. The interview itself was conducted over skype and the audio was recorded. I have transcribed this recording below.

Question 1

Interviewer: How was your experience using the software to complete the tasks?

Yorke: It was relatively easy to use if you know what you are doing. I assume analysts would have some sort of manual, forum or training to use the software initially. The web app was fast to make classifications and there was no latency switching tasks or making filter requests.

Question 2

Interviewer: Did you feel like the new requested features met your needs? If not what improvements need to be made to fix that?

Yorke: Yes. I was very happy with the filter system. It was quite sophisticated and entuitive but didn't add any complications using the software. The graphs were informative and 2-variable plots were very helpful and data was easy to read.

However, I had a few minor issues with the graphs. The x-axis on the time-series seemed to be an elapsed time? It wasn't clear what it meant and I had to perform manual calculations to see what 24hr time that would be. would let the user select which day they want to see. In addition, the time series showed that there were 17.5 average transactions per hour for some hours. This is not possible. It should be a

discrete number being that it is a count per hour of a certain type.

Also why did the different graphs use opposing colour coding for the classifications of data points. The timeseries used orange for fraud while the 2-variable plot used blue for fraud. It confused me and only noticed it at the end while I was closing it down. I used this information to classify the transaction as fraudulent instead of non-fraudulent. Once these changes are made it'll be a lot easier to read.

Question 3

Interviewer: Were there any wider last changes would you make to the program that weren't initially requested before the end of development?

Yorke: Yes. Firstly, I don't understand why the dashboard and dashboard can't just be one page: they both have usual features and now that the manage page can filter transactions by class it does the job of the dashboard table which exclusively shows transactions whose classifications aren't decided on. Moreover it would be really helpful to have the graphs on the dashboard page on the manage page. Lastly, the manage page design sort of breaks down and feels uncomfortable since there are so many columns. I'll send an example after.

Also, I'd like some sort of end simulation option which shows the final metrics of accuracy and the performance of the program and analyst. I don't think you need a help page, I think it's safe to assume the analyst will be trained to use it. Then the page will only consist of two pages. Lastly, on this table I'd like a number to show how many transactions there are filtered out of how many there are in total. Nothing more detailed than that. Otherwise the program's ready and looking good. I'm excited to see the final version.

Interview Analysis

The stakeholder is satisfied with the main functionality of the program. However he has requested 4 changes to give the user a better experience for the final program:

1. Switch the colours of fraudulent and non-fraudulent data points on the 2-variable scatter plot to be consistent with the timeseries colour coding.

2. Change the x-axis of the time series so that the ticks show the 24 hour time instead of the current hours elapsed since the beginning of the dataset.
3. Create a select input to allow the user to choose which day of the timeseries they want to see.
4. Ensure that the y-axis displays a discrete number of transactions per hour. Not decimal/continuous numbers.
5. Combine the dashboard page and the manage page into one page.
6. Display a figure above the table to show how many transactions in the table are being displayed out of the total number in the database.
7. Delete the help page.
8. Create a reset page which allows the user to end the simulation and reveal the performance of the program.

Once all 8 of these tasks are complete and are ready for submission we are going to conduct a final interview to ensure that these changes meet the requirements of the stakeholder.

Chapter 37: Improving the usability of the graphs - (Objective 4*)

In this chapter we are going to address the first 4 bullet points of the tasks outlined in the analysis of the interview with the stakeholder.

1. Switching the colours of fraud and non-fraud in the 2-variable plot

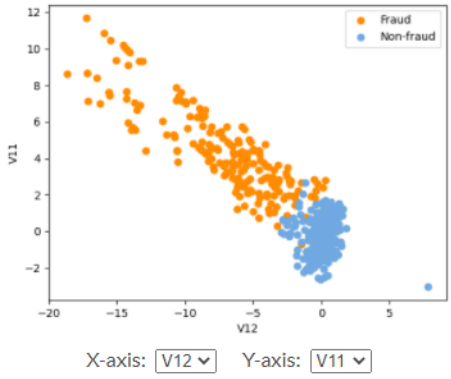
Justification: During alpha testing the analyst actually failed a test by getting the colour of the classifications the wrong way round and misclassified a transaction as fraudulent. This was because the timeseries used orange to indicate fraudulency while the 2-var used blue.

To fix this I copied the colours used for the time series line plot and added it as the parameter for the scatter plots of both classifications below:

```
# Input features to plot
plt.scatter(fraud[feature1], fraud[feature2], color = "#FF8C00") # orange
plt.scatter(non_fraud[feature1], non_fraud[feature2], color = "#70a9e1") # light blue
```

Here is a picture of a sample plot before the color reversal:

And after:

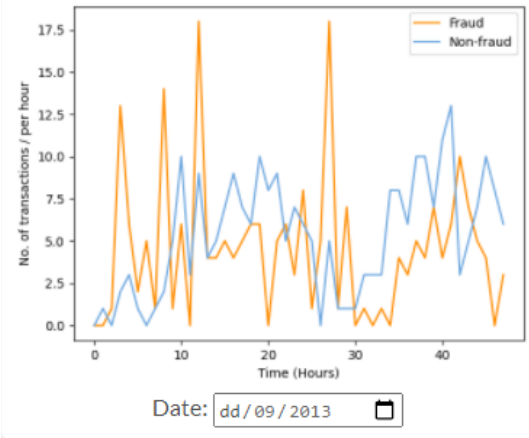


Colour reversal is complete.

2/3. Letting the user choose which day is shown on the timeseries and changing the x-axis.

The first step is creating a date input to select which days the time series shall show. The range of times has been restricted using the min and max attributes to stop the user choosing dates for which data doesn't exist.

```
<!-- Timeseries graph -->
<div class="graph-container">

    <!-- Print graph if it is successfully passed-->
    {% if timeseriesImg != None %}
        <img src="{{ timeseriesImg }}" id="timeseries">
    {% endif %}

    <div class="date-selector-container">
        <label for="x-axis">Date: </label>
        <input type="date" min="2013-09-01" max="2013-09-02">
    </div>
</div>
```



Now we need to configure the event listener which sends the request to get this image data for a given day. We will repeat the code used to send POST requests when the feature selected for the x or y-axis has changed.

```
// Script which changes the graph when axis value is changed
const timeseriesPlot = document.querySelector('#timeseries') // Get image to load it in

document.querySelector('.date-selector').addEventListener('change', async e => {

    // Get date values via DOM and add it to querystring.
    const queryString = '?date=' + e.target.value;
    const URLaddress = server + '/getTimeseries' + queryString;

    // Make POST Request to receive the image data for the new graph
    fetch(URLaddress)
        .then(response => {
            // GET Request successful - update image
            return response.json();
        })
        .then( data => {
            timeseriesPlot.src = '' + data.imgB64;
            console.log("Image updated.");
        })
        .catch(error => {
            // GET Request unsuccessful
            console.log(error);
        });
});
```

Now we're going to create the view on the backend which handles and responds to this request.

```python
# Classify case - responding to input
@app.route('/getTimeseries', methods = ['GET'])
def getTimeseries():

    # Import the decisionsDB
    decisionDB = pd.read_csv("dataset/decisionDB.csv")

    # GET querystring arguments from request
    selectedDate = request.args['date']

    # Plot the graph and generate the encoded image data for it
    timeseriesB64 = graphs.createTimeseries(decisionDB, selectedDate).decode('utf8')

    # Return image data
    return jsonify({'imgB64': timeseriesB64})
```

This view will exhibit the exact same behaviour as getScatter (from chapter 32). We will need to change the implementation of the createTimeseries function.

Firstly we have to consider the selected date we extract from the date input. We will also have to use a new x-axis which has times formatted into it. Because of this we require a method which utilises Pandas own datetime object. 'Pandas.Grouper' does exactly this: its groups record by the specified frequency if the target selection is a datetime object.

This is different to the previous implementation which went through all transactions and decided if they were in a certain hour by checking the number of elapsed seconds (chapter 31). This is incompatible with our new approach so we will have to remove this code. However we will keep the code which translates the image into b64 data.

Here's the code for our new function:

```python
def createTimeseries(decisionDB, selectedDate):

    # Get all transactions from that day
    filteredDB = decisionDB[decisionDB.Date == selectedDate]

    # Setting the date as the index since the TimeGrouper works on Index, the date column is not dropped to be able to count
    filteredDB['Datetime'] = pd.to_datetime(filteredDB['Datetime'])
    filteredDB.set_index('Datetime', drop=False, inplace=True)

    # Split by classification
    fraud = filteredDB[filteredDB.Decided_Class == 1]["Decided_Class"]
    non_fraud = filteredDB[filteredDB.Decided_Class == 0]["Decided_Class"]

    # Group transactions by hour, count them and plot it.
    fraud.groupby(pd.Grouper(freq='1H')).plot(kind='line', color="#FF8C00") # orange
    non_fraud.groupby(pd.Grouper(freq='1H')).plot(kind='line', color="#70a9e1") # light blue

    # label the axes
    plt.xlabel('')
    plt.ylabel('Frequency (per hour)')

    # Create legend
    plt.legend(["Fraud", "Non-fraud"])

    # You can load your input data you have into the BytesIO before giving it to the library.
    # After it returns, you can get any data the library wrote to the file from the BytesIO using the getvalue() method.
    figfile = BytesIO()

    # load the image data into temporary file (IO)
    plt.savefig(figfile, format='png')
    plt.clf() # clears figure

    # rewind to beginning of file
    figfile.seek(0)

    # Base64 encoding is a type of conversion of bytes into ASCII characters
    figdata_png = base64.b64encode(figfile.getvalue())
    return figdata_png
```

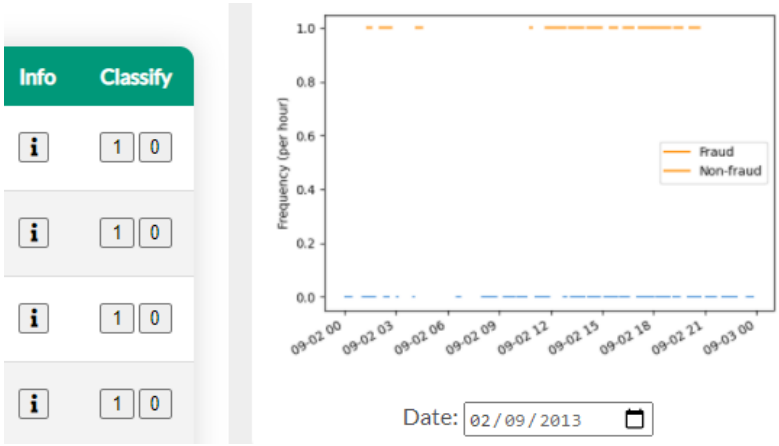
I also loaded the 2nd of September as the default time series to be loaded:

```python
# create the image data for the plots
timeseriesB64 = graphs.createTimeseries(decisionDB, '2013-09-02')
scatterB64 = graphs.createScatter( decisionDB,'V14', 'V11')

# render table and dashboard page
return render_template('index.html',  row_data = list(profilesTable.values.tolist()), table_headings = profilesColNames,
                       modal_data_table = list(detailsTable.values.tolist()), modal_data_headings = detailsColNames,
                       timeseriesImg = timeseriesB64.decode('utf8'), scatterImg = scatterB64.decode('utf8'))
```

Now everything's in place, let's test it.

Test 1

The line graphs display successfully as a frequency (per hour) over the 2nd of September since it was the initial date we loaded in. Here's a screenshot of it:



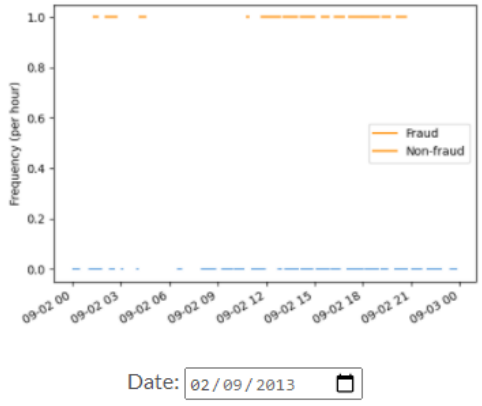
Test failed. This graph doesn't show frequency but just indicates when each type of classification occurred. Hours also didn't appear as the x-axis.

Why did it fail?

I don't believe it was the '1H' frequency which tripped it up because the documentation this page stated it was accepted. To test this hypothesis, I changed the frequency to 60Min to see if a different result would be produced.

```
# Group transactions by hour, count them and plot it.
fraud.groupby(pd.Grouper(freq='60Min')).plot(kind='line', color="#FF8C00") # orange
non_fraud.groupby(pd.Grouper(freq='60Min')).plot(kind='line', color="#70a9e1") # light blue
```

Result:



Test failed. Nothing changed.

Why did it fail?

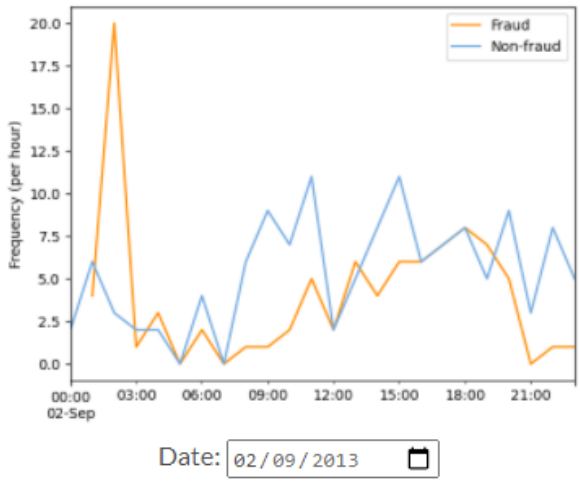
I believe the transactions may have been grouped together by time but the function failed to display how many per group there were every hour.

What am I going to change?

I'm going to chain a .count() method to the end to see if that will successfully show the frequency.

```
# Group transactions by hour, count them and plot it.
fraud.groupby(pd.Grouper(freq='1H')).count().plot(kind='line', color="#FF8C00") # orange
non_fraud.groupby(pd.Grouper(freq='1H')).count().plot(kind='line', color="#70a9e1") # light blue
```

Result:

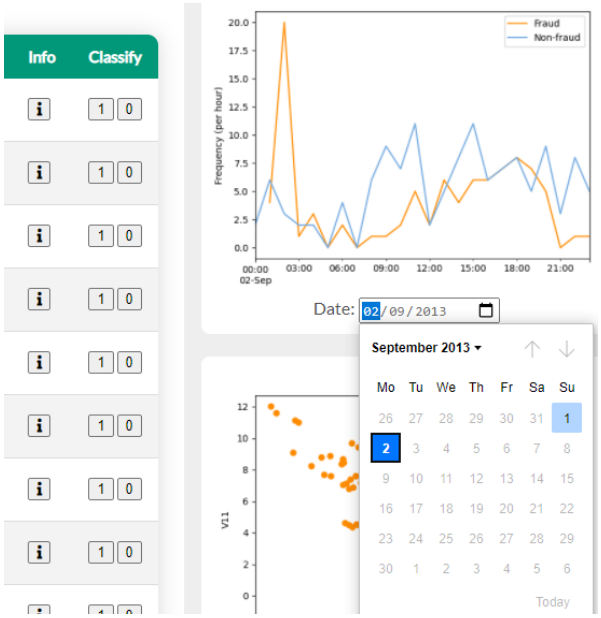


Test successful. Hourly intervals over 24 hours from 2nd of September showing.

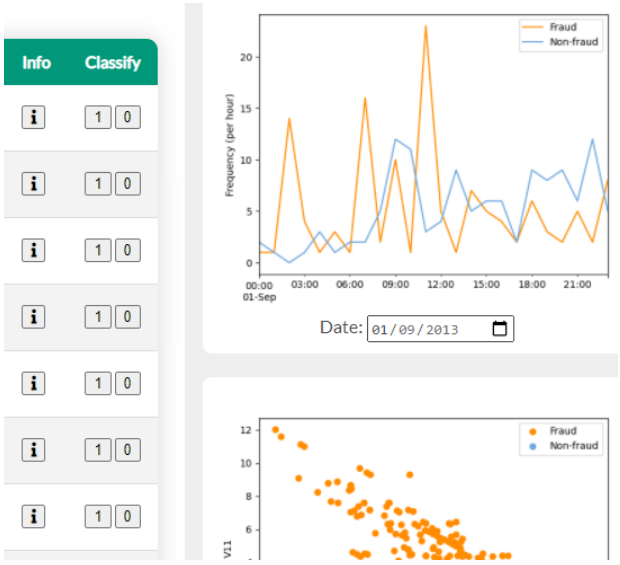
Test 2:

In this test we are going to change the date selected to the 1st of September. It should without refresh send a POST request and process the image data and send it back.

The screenshot below shows before we select the new date:



The screenshot below shows the updated image data.



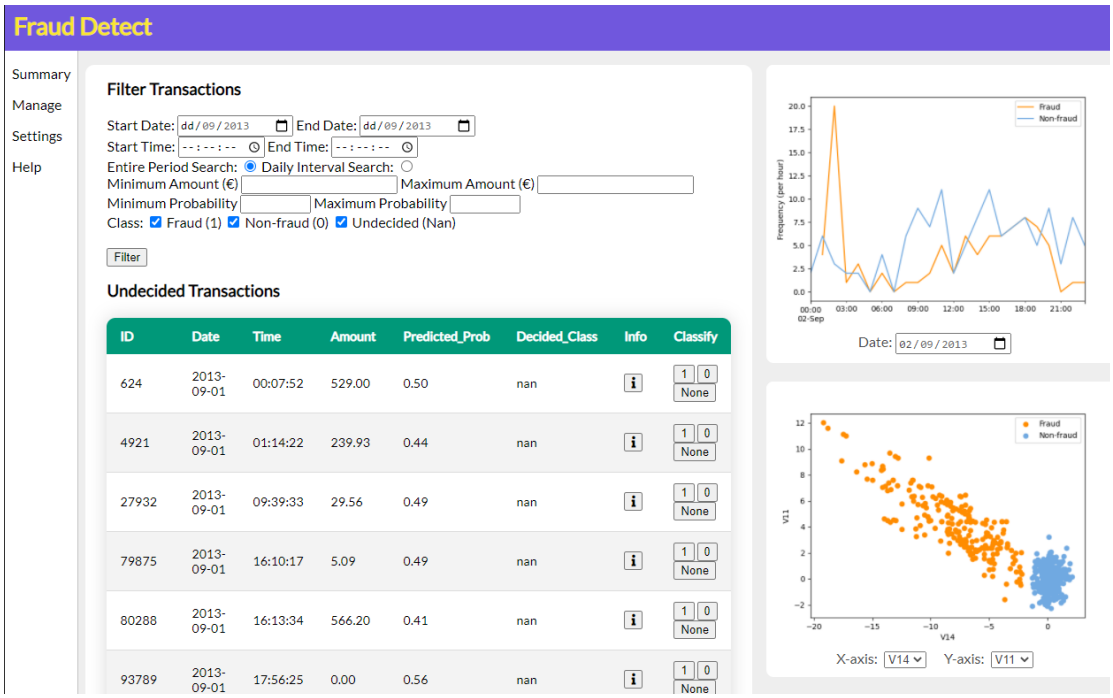
Test successful. Loaded without refresh. Hourly intervals over 24 hours from 1st of September showing.

We can now move onto combining the manage and dashboard page.

Chapter 38: Combining the manage and dashboard page. - (Objective 4*)

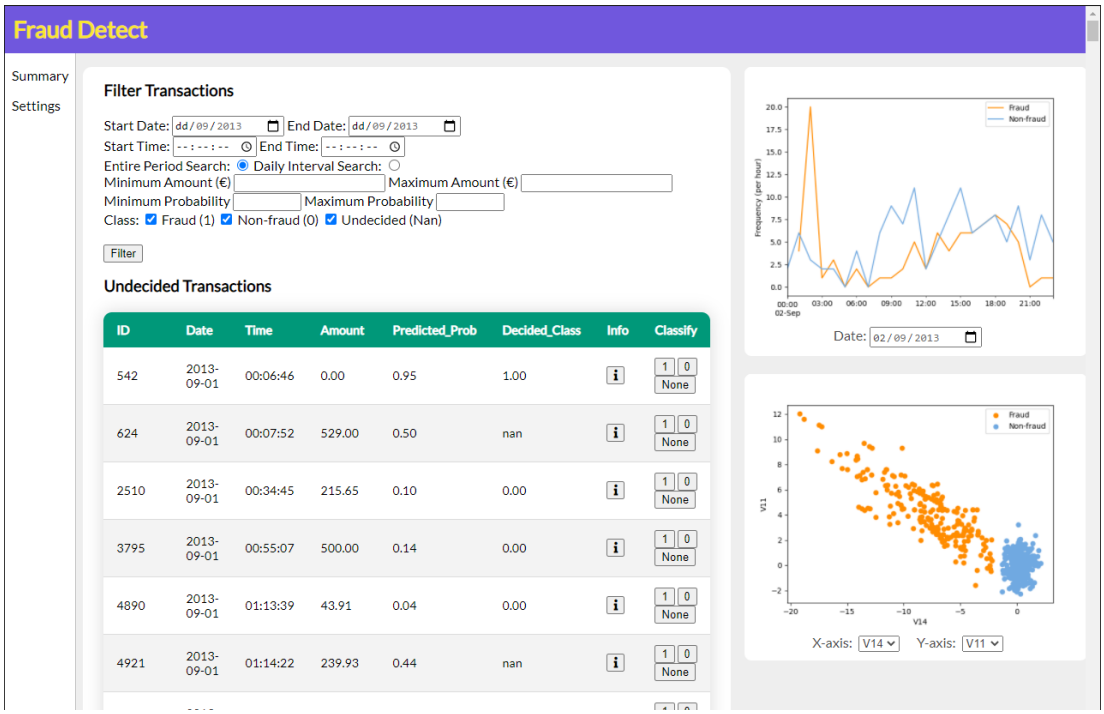
In this chapter we are going to address the 5th, 6th and 8th bullet points of the tasks outlined in the analysis of the interview with the stakeholder.

In order to merge the manage and dashboard page, we copied the filter HTML code and moved the filter javascript file to index.html. This is what the results look like:



Now we are going to remove both the help and manage pages since they are needed. In their place I am going to add the settings page to the navigation bar in preparation for the next chapter.

The dashboard page originally held only the undecided transactions. It now must hold all transactions prior to any filter operation. After making this change to app.py index view, this is what my page looked like:



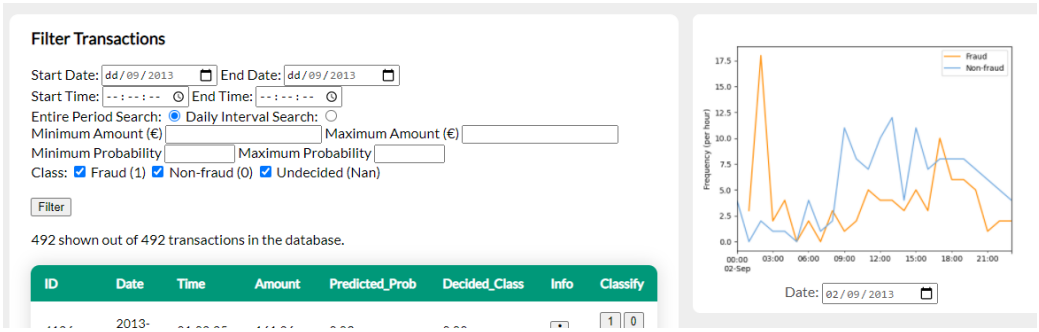
Now we must remove the ' Undecided Transactions' header and display how many transactions are being displayed out of the total number of transactions. I'm going to use the length of the filteredTable Jinja 2 variable being sent to the template and also the length of the total number of transactions 'databaseLength'.

```
# render table and dashboard page
return render_template('index.html',  row_data = list(profilesTable.values.tolist()), table_headings = profilesColNames,
                       modal_data_table = list(detailsTable.values.tolist()), modal_data_headings = detailsColNames,
                       timeseriesImg = timeseriesB64.decode('utf8'), scatterImg = scatterB64.decode('utf8'),
                       databaseLength = decisionDB.shape[0])
```

```
<!-- Number of rows displayed out of total -->
<p>{{ row_data|length }} shown out of {{ databaseLength }} transactions in the database.</p>
```

Test 1

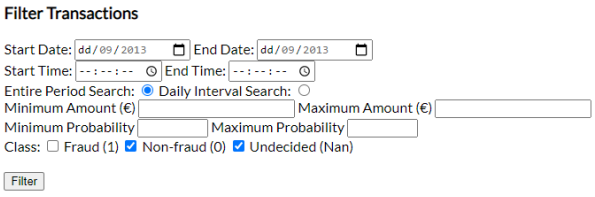
Shows full 492 out of 492 transactions when no filter applied on first launch.



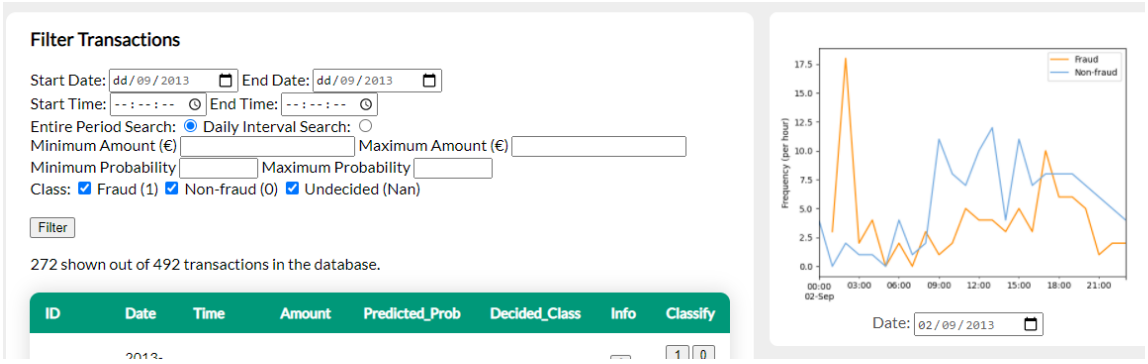
Test successful.

Test 2

Shows approximately half of the transactions when filtering out fraudulent transactions, before:



After:



Test successful. Roughly half of the transactions shown.

Chapter 39: Model reset and evaluation - (Objective 4*)

The settings page requested by the stakeholder only contained one setting - evaluate model and reset it. So I've decided I'm going to make it a button in the navigation bar. Once pressed, a

pop-up modal window should appear warning the user what it does. After that, the user is taken to a page which summarizes the success of the model. On that page there is a button which returns you back to the main page.

First I am going to evaluate the model. I will use a confusion matrix to do so.

A confusion matrix is a matrix which compares the actual target values with those predicted by the machine learning model. The name comes from the ability whether the system is confusing two classes (i.e. commonly mislabeling one as another).

We are going to use two confusion matrices in our evaluation. One to compare the predictive models performance using the “Predicted Class” attribute and the other to compare the analysts performance using the finalised “Decided Class” attribute.

I will have to make a modification to the evaluation function which returned the number of true positives, true negatives, false positives and false negatives for a given dataset. At first we needed it for our k-fold cross validation which compared the predicted class with the actual class. Now we need the option to compare to the decided class as well. I am going to add a parameter to distinguish what we are comparing for the evaluate function:

```python
def evaluate(testingSet, classType = "Predicted"):
    """
    Evaluate the amount of every correct and incroret instance of a predicted or decided classes.
    """

    # Count and class every correct and instance of a prediction
    tp = tn = fp = fn = 0 # where True Positive = tp, False Negative = fn, etc

    for index, testingExample in testingSet.iterrows():

        # Use the probability to predict a class
        predictedClass = testingSet.loc[index][f"{classType}_Class"]
        actualClass = testingSet.loc[index]["Class"]

        if actualClass == predictedClass: # Got the prediction correction
            if actualClass == 1: # and it was a positive correct... etc.
                tp += 1
            else:
                tn += 1
        else:
            if predictedClass == 1:
                fp += 1
            else:
                fn += 1

    return (tp, tn, fp, fn)
```

Now we need to create the function which creates the confusion matrix. It will use the same b64 data transmission method as the time series and scatter plot.

```
def createConfusionMatrix(dataset, testResults, classCategory):

    # create confusion matrix
    tp, tn, fp, fn = testResults
    cf_matrix = np.array([[tn, fp], [fn, fp]]) # make 2-d array (matrix)

    # add labels
    group_names = ["True Negative", "False Positive", "False Negative", "True Positive"]
    group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]

    # add percentages
    group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten() / np.sum(cf_matrix)]
    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names, group_counts, group_percentages)]

    # create heatmap
    labels = np.asarray(labels).reshape(2,2)
    sn.heatmap(cf_matrix, annot = labels, fmt="", cmap='Blues')

    # load the image data into temporary file (IO)
    figfile = BytesIO()
    plt.savefig(figfile, format='png')

    # clear figure and rewind to beginning of file
    plt.clf()
    figfile.seek(0)

    # Base64 encoding is a type of conversion of bytes into ASCII characters
    figdata_png = base64.b64encode(figfile.getvalue())
    return figdata_png
```

The above function takes in the test results in the form which is unpacked into 4 variables: true positive, false positive, true negative and false negative. These are then displayed as a matrix by the pyplot library. This is then converted to b64 image data. It also takes in a variable called classCategory which chooses whether to return the confusion matrix model's initial class prediction or the user's finalised decision classifications.

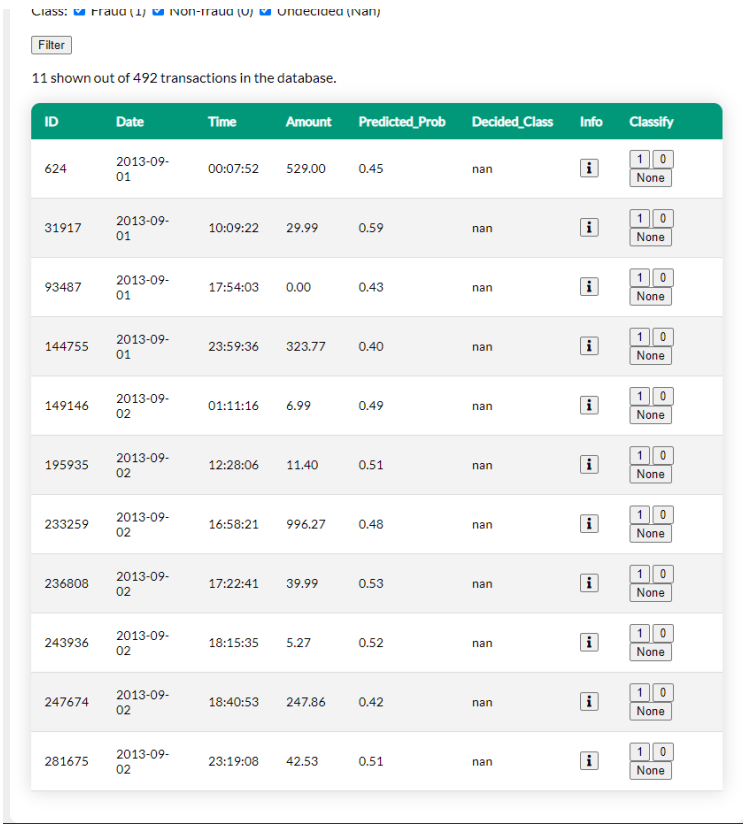
When the user decides to reset the model and see their and the predictive model's performance evaluated, these metrics will be calculated and displayed on a page. The image below is the view that handles the user's GET request for this reset page:

```python
@app.route('/reset')
def reset():

    # bring in pre-tested dataset
    decisionDB = pd.read_csv("dataset/decisionDB.csv")

    # evaluate both predicted and decided
    predictedResults = model.evaluate(decisionDB, "Predicted")
    decidedResults = model.evaluate(decisionDB, "Decided")

    # create the confusion matrices for both predicted and decided
    predConfMatrixB64 = graphs.createConfusionMatrix(decisionDB, predictedResults ,'Predicted')
    decConfMatrixB64 = graphs.createConfusionMatrix(decisionDB, decidedResults, 'Decided')

    # build evaluation for both predicted and decided
    predictedEval = {
        "recall": model.recall(predictedResults),
        "precision": model.precision(predictedResults)
    }
    decidedEval = {
        "recall": model.recall(decidedResults),
        "precision": model.precision(decidedResults)
    }

    # reset current model
    transactions = pd.read_csv("dataset/creditcard.csv") # Import the large datset locally
    transactions["V0"] = 1 # assign the 0th feature
    transactions['ID'] = list(range(1, len(transactions) + 1)) # assign unique IDs/indexes to every record

    dataset = model.balanceDataset(transactions) # balance/reduce dataset
    features = ['V0', 'V14'] # define some test model

    testedSet = model.trainTest(dataset, features)
    decisionDB = model.createDecisionDB(testedSet) # make the decision database
    decisionDB.to_csv("dataset/decisionDB.csv", index=False) # save as csv to prevent future wasted time

    # reset current model
    return render_template('evaluation.html', predConfMatrixImg = predConfMatrixB64.decode('utf8'), modelEval = predictedEval,
                                              decConfMatrixImg = decConfMatrixB64.decode('utf8'), userEval = decidedEval)
```

The HTML below describes the content of this reset/evaluation page. I'd like it to have a dark opaque black background and in the foreground a large white card with the evaluation which will be broken into two sections: first the model's initial predictions evaluation and second the users final decided classification. The metrics for each will be the precision and recall.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://kit.fontawesome.com/91c3acf03e.js" crossorigin="anonymous"></script>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/base.css') }}">
    <link rel="stylesheet" href="{{ url_for('static', filename='css/evaluation.css') }}">
    <title>Dashboard | Fraud Detection </title>
</head>
<body>
    <!-- Modal Style Evaluation Card -->
    <div class="evaluation-container">

        <h1>Performance Evaluation</h1>
        <div class="evaluation-cards">

            <!-- Confusion Matrix and Evaluation Card for the Model -->
            <section>
                <h3>Model Success</h3>
                <img src="{{ predConfMatrixImg }}" class="cf_matrix">
                <p><strong>Recall:</strong> {{ '%0.3f'| format(modelEval["recall"]|float) }} </p>
                <p><strong>Precision:</strong> {{ '%0.3f'| format(modelEval["precision"]|float) }} </p>
            </section>

            <!-- Confusion Matrix and Evaluation Card for the Analyst -->
            <section>
                <h3>User Success</h3>
                <img src="{{ decConfMatrixImg }}" class="cf_matrix">
                <p><strong>Recall:</strong> {{ '%0.3f'| format(userEval["recall"]|float) }} </p>
                <p><strong>Precision:</strong> {{ '%0.3f'| format(userEval["precision"]|float) }} </p>
            </section>

        </div>
        <a href="{{ url_for('index') }}" class="btn btn-return">Back to dashboard.</a>
    </div>
</body>
```

To see a difference in metrics, we actually must assign some classification to the undecided list of transactions. So lets filter them out:



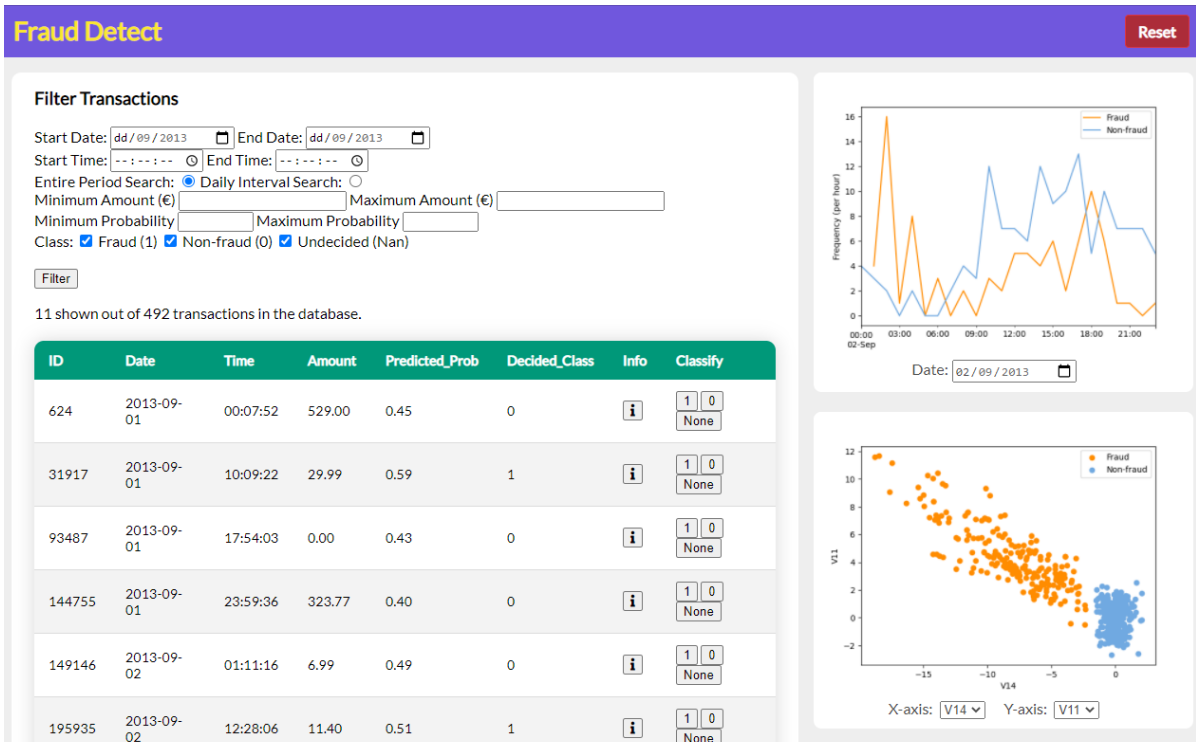
And randomly assign them to 0 and 1.

Filter

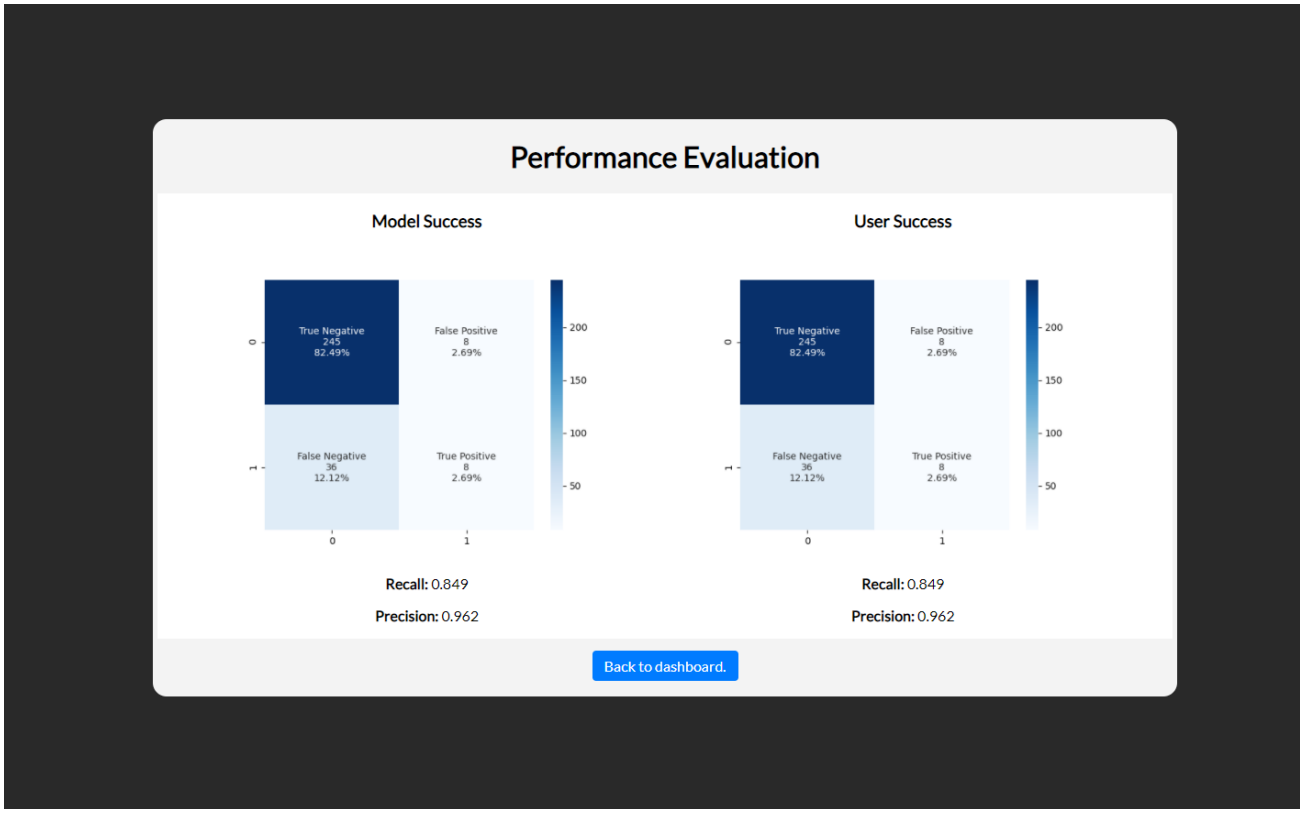
11 shown out of 492 transactions in the database.

ID	Date	Time	Amount	Predicted_Prob	Decided_Class	Info	Classify
624	2013-09-01	00:07:52	529.00	0.45	0	i	1 0 None
31917	2013-09-01	10:09:22	29.99	0.59	1	i	1 0 None
93487	2013-09-01	17:54:03	0.00	0.43	0	i	1 0 None
144755	2013-09-01	23:59:36	323.77	0.40	0	i	1 0 None
149146	2013-09-02	01:11:16	6.99	0.49	0	i	1 0 None
195935	2013-09-02	12:28:06	11.40	0.51	1	i	1 0 None
233259	2013-09-02	16:58:21	996.27	0.48	0	i	1 0 None
236808	2013-09-02	17:22:41	39.99	0.53	1	i	1 0 None
243936	2013-09-02	18:15:35	5.27	0.52	1	i	1 0 None
247674	2013-09-02	18:40:53	247.86	0.42	0	i	1 0 None
281675	2013-09-02	23:19:08	42.53	0.51	1	i	1 0 None

Now lets reset the model:



This is the final evaluation:



It has a button to go back to the dashboard where the new model is.

Tests added for post-development phase

No	Type	What is being tested?	Input / Action Taken	Expected Outcome
Reset / Evaluate the Model				
15.1	Valid	User clicks the reset button.	Mouse click on the reset button.	Goes to the evaluation page. Once the dashboard is loaded, a new model is in place.
15.2	Valid	Evaluation metrics for both user and predictive model performances are displayed.	N/A.	Shows two confusion matrices and two sets of evaluation metrics to 3.d.p.
15.3	Valid	The user can go back to the new model from the evaluation page.	Mouse click on the hyperlink.	Loads dashboard page with new model.

Chapter 40: Feature Selection Strategy (Objective 3*)

First it's important to state that we have over 30 features and it will be computationally expensive so test every feature just to select one, thus, instead of running this subroutine every time a session in the web app is started, we will test this feature and its results will form our final model.

The Forward feature selection algorithm works as follows:

1. A base model (V0) was established and evaluated to obtain a benchmark performance.
2. One by one features V1 - V28 were temporarily added to the model to gather whether the performance was better than the benchmark scores on their arrival.
3. After iterating over all features, the one with the greatest performance gain was permanently added to the model.
4. This was repeated until no performance gain was observed.

The code below describes the steps above:

```
def forwardFeatureSelction(dataset):
    selectedFeatures = ["V0"]
    candidateFeatures = [f"V{num}"for num in range(1,29)]
    running = True

    while running: # runs until no better features found

        bestCandidates = {}
        benchmarkPrecision, benchmarkRecall = cross_validation(selectedFeatures, 2, dataset)
        print("Benchmark Results: ", benchmarkPrecision, benchmarkRecall)

        # test all features
        for possibleFeature in candidateFeatures:

            print("Testing:", possibleFeature)

            trialFeatures = selectedFeatures + [possibleFeature]
            newPrecision, newRecall = cross_validation(trialFeatures, 2, dataset)

            if benchmarkPrecision <= newPrecision and benchmarkRecall <= newRecall:
                # add an average score
                bestCandidates[possibleFeature] = (newPrecision + newRecall) / 2

        # select best feature
        if len(bestCandidates) != 0:

            # get the feature with the highest average score
            bestFeature = max(bestCandidates, key = bestCandidates.get)

            # add this feature to the list
            selectedFeatures.append(bestFeature)

            # remove this feature from the candidates
            candidateFeatures.remove(bestFeature)

            print("Selected: ", selectedFeatures)

        else:
            # no candidate fratures - current model is best
            running = False

    return selectedFeatures
```

Lets test to see if this worked:

```
forwardFeatureSelction(dataset)
```

```
(216, 239, 8, 28)
(214, 240, 5, 32)
Testing: V23
(215, 240, 7, 29)
(215, 239, 6, 31)
Testing: V24
(216, 239, 8, 28)
(215, 239, 6, 31)
Testing: V25
(214, 239, 8, 30)
(215, 240, 5, 31)
Testing: V27
(216, 240, 7, 28)
(215, 240, 5, 31)
Testing: V28
(216, 240, 7, 28)
(215, 240, 5, 31)
Selected:  ['V0', 'V14', 'V20', 'V26', 'V21']
(217, 240, 7, 27)
```

Test partially failed

The algorithm didn't complete in a reasonable time and potentially could run indefinitely. After leaving it for 2 hours it only picked 4 features. I halted the runtime so I could focus on a less time-intensive approach. It only partially failed because it picked features that could have greatly improved the model.

Why did the test fail?

We will explain it in terms of time-complexity and algorithmics.

- Let k = number of folds in cross validation
- Let n = number of selected parameters / features
- Let p = number of possible features
- Let i = iteration number of outer loop, i = 0 to ->n.

Iterations = $outer\ running\ while\ loop\ \Rightarrow\ n\ *\ (\ \sum_{i=0}^{k}\ [(p\ -\ i)\ *\ cross\ validation(m,\ i)]\ )$

$= n\ *\ (\ \sum_{i=1}^{n}\ (p\ -\ i)\ *\ kmi^2\ )$ - k-fold approximation calculated in design module 7

$= n\ *\ (\ \sum_{i=1}^{n}\ (p\ -\ i)\ *\ kmi^2\ )$

$= kmn\ *\ (\ \sum_{i=1}^{n}\ (p\ -\ i)\ *\ i^2\ )$

$$= kmn * \sum_{i=1}^{n} (p\, i^2 - i\ ) = kmn\,[\,(p \sum_{i=1}^{n} i^2) - \sum_{i=1}^{n} i\ ]$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

Sum of the first n positive integers: $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$

$$= kmn * \sum_{i=1}^{n} (p\, i^2 - i\ )$$

$$= kmn\,[\,(p \sum_{i=1}^{n} i^2) - (\sum_{i=1}^{n} i)\,]$$

$$= \text{kmn}[\ p[\frac{(n+1)(2n+1)}{6}] - (\frac{n(n+1)}{2})]$$

$\approx kmn * (pn^2)$ - approximation works k-fold calculation earlier was an upper bound due to cancelled term

$= O(kmpn^3)$ - average case complexity

$= O(kmp^4)$ - worst case time complexity: n = p all possible features improve the model and are selected:

What we learn from worst case time complexity calculations.

I. Since the outer (n sized) loop cannot change, the inner summation to evaluate every possible feature before choosing the best must be removed -> since this eventually approximates to n^2. To reduce this to its average input to between 0 and n resulting in $(\frac{n}{2})^2$ we must stop always evaluating all of the features.

II. Need to prevent the worst case, n becoming p (since $time\ \alpha\ p^4$).Therefore some features could be preselected or some known to not be ever beneficial removed. This will always reduce the average case of $time\ \alpha\ n^3$ which is crucial since it was the largest factor in our time

III. Scores seemed to only ever incrementally change from a certain point. Wasted time making no change.

IV. Since k is directly proportional to time - should be reduced but risks losing accuracy. Number of transactions (m) is also directly proportional to time - could be greater loss of accuracy so would not recommend,

Remedial Action

1. Re-work the selection to make it faster and more effective:
    - During selection, once a trial feature is found to be effective, add it straight to model and start the loop again. Loop must be started again because machine learning models are sensitive to adding new features. Solves (point I)
    - Add the scaled amount and time columns missing - improves model scores but also increases the value of p and therefore time.
    - Round scores to 3 or 4 d.p so features don't negligibly "improve" the model. Also will prevent model overfitting. Solves (point III)
2. Change the cross-validation method:
    - Combine metrics into one number - called f1 score. Adds simplicity to algorithms and stopping the zero error exceptions. The previous solution wasn't in fact valid, none should have been returned instead of zero.
    - Before split into folds, shuffle order of transactions at the beginning of each k-fold to stop bias.
    - Reduce k from 5 to 3 to make the selection process 1.67 times faster. Solves (point IV)

F1 formula.

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

Combines both metrics into one and also prevents those boundary case exceptions which can cause errors.

8. Feature Selection

Selects the best list of features to predict fraud accurately in a given a dataset.

```
In [*]: # select features
        selectedFeatures2 = selectFeatures2(dataset)
```

```
Current Model:  ['V0', 'V1', 'V2', 'V3', 'V4', 'V5', 'V8', 'Amount_Scaled', 'V10', 'V14', 'V6', 'V12', 'V13', 'V16', 'Time_Sc
aled', 'V9']  gives F1 Score:  0.9428
Testing: V7
Testing: V11
Testing: V15
Testing: V17
Testing: V18
Testing: V19
Testing: V20
Current Model:  ['V0', 'V1', 'V2', 'V3', 'V4', 'V5', 'V8', 'Amount_Scaled', 'V10', 'V14', 'V6', 'V12', 'V13', 'V16', 'Time_Sc
aled', 'V9', 'V20']  gives F1 Score:  0.9439
Testing: V7
Testing: V11
Testing: V15
Testing: V17
Testing: V18
Testing: V19
Testing: V21
Testing: V22
```

The code below addresses all the remedial action.

```
: def selectFeatures2(dataset):
    selectedFeatures = ["V0"]
    candidateFeatures = [f"V{num}"for num in range(1,29)] + ["Amount_Scaled", "Time_Scaled"]
    running = True

    currF1 = round(cross_validation2(selectedFeatures, 2, dataset), 4)
    print("Current Model F1: ", currF1)

    while True: # runs until loop broken (when no better features found)
        modelImproved = False

        # test all features
        for possibleFeature in candidateFeatures:

            print("Testing:", possibleFeature)

            trialFeatures = selectedFeatures + [possibleFeature]
            newF1 = round(cross_validation2(trialFeatures, 4, dataset), 4) # round to 4 d.p - essentially converged

            # select the feature if score improved
            if newF1 > currF1:

                # remove from candidates
                selectedFeatures.append(possibleFeature)
                candidateFeatures.remove(possibleFeature)

                # update benchmark score
                currF1 = newF1
                print("Current Model: ", selectedFeatures, " gives F1 Score: ", currF1)

                # model disrupted -> loop again.
                modelImproved = True
                break

        if modelImproved == False:
            return selectedFeatures  # exits function
```

```
:
def cross_validation2(features, k, dataset):
    """
    Returns the average precision and recall values when training and testing a dataset k times.
    """

    # shuffle dataset to reduce bias
    dataset.sample(frac=1).reset_index(drop=True)

    sampleSize = len(dataset) // k
    folds = []

    # Split dataset into k folds
    for i in range(k):

        start = i * sampleSize
        end = (i + 1) * sampleSize - 1 if i != k - 1 else len(dataset) - 1 # final index if last group
        folds.append(dataset.iloc[start: end]) # the- kth fold: testing)

    # Now our folds are created we can iterate through it:

    f1Scores = []

    for i in range(k):

        folds_copy = folds.copy() # create a copy of the folds

        # Create the training/test sets
        testingSet = folds_copy.pop(i) # kth fold
        trainingSet =  pd.concat(folds_copy) # remaining k-1 folds

        # Train our model
        tunedParameters = train(trainingSet, features)

        # Test our model
        testedSet = test(testingSet, features, tunedParameters)
        testResults = evaluate(testedSet)

        # Add results for future calculations
        f1Scores.append(f1_score(testResults))

    return sum(f1Scores) / k
```

Time to test it on the dataset:

8. Feature Selection ¶

Selects the best list of features to predict fraud accurately in a given a dataset.

In [91]:
```
# select features
selectedFeatures2 = selectFeatures2(dataset)
```

```
Testing: V18
Testing: V19
Testing: V20
Current Model:  ['V0', 'V1', 'V2', 'V3', 'V4', 'V5', 'V8', 'Amount_Scaled', 'V10', 'V14', 'V6', 'V12', 'V13', 'V16', 'Time_Sc
aled', 'V9', 'V20']  gives F1 Score:  0.9439
Testing: V7
Testing: V11
Testing: V15
Testing: V17
Testing: V18
Testing: V19
Testing: V21
Testing: V22
Testing: V23
Testing: V24
Testing: V25
Testing: V26
Testing: V27
Testing: V28
```

In [92]:
```
print(selectedFeatures2)
```

```
['V0', 'V1', 'V2', 'V3', 'V4', 'V5', 'V8', 'Amount_Scaled', 'V10', 'V14', 'V6', 'V12', 'V13', 'V16', 'Time_Scaled', 'V9', 'V2
0']
```

Test successful

Large set of features selected in reasonable time.

No	Type	What is being tested?	Input / Action Taken	Expected Outcome
F1 Score				
6.5	Valid	F1 score successfully calculated.	Valid tuple of 4 positive integers.	Return the metric as a decimal between 0-1. (inclusive).
6.6	Boundary	Exception is made when a valid tuple causes a division by zero error.	Valid tuple of four integers but true positives and, false positives and negatives are equal to zero.	Print a message that no true positives and false negatives present so cannot calculate.

Section 4. Evaluation of the Solution

In this section we will evaluate how well we have completed the 4 primary objectives defined in the analysis section. We will also discuss limitations of achieving these goals and wider limitations not faced by other similar programs and why. After a final interview with the stakeholder, we will make a conclusion of our project's success.

Testing to inform evaluation

The table below shows every test taken

No	Type	What is being tested?	Input / Action Taken	Expected Outcome	Actual Outcome	Video / Screenshot Reference
Preparing the data						
1.1	Valid	Any duplicate records present in the dataset are	Dataset of 280,000 transactions searched.	No remaining duplicate records left in the dataset.	As expected.	Screenshot 1.

		removed.				
1.2	Valid	Any records with missing fields are removed.	Dataset of 280,000 transactions searched.	No records with missing values left in the dataset.	As expected.	Screenshot 1.
1.3	Valid	Amount and time columns are standardised in preparation for correlation analysis.	Amount and time elapsed columns of the 280,000 transactions.	Histogram after standardisation fattens.	As expected.	Screenshot 1.
1.4	Valid	Dataset is balanced for testing and iterative development.	Taken from the total 280,000 transactions	Equal number of each of the classifications.	As expected. 492 of each classification	Screenshot 1.
1.5	Valid	Correlation matrix is produced and the feature most correlated to the fraud class becomes the prototype model.	Reduced portion of the transactions dataset analysed.	A color mapped matrix to visualise highest levels of correlation.	As expected. 5 most correlated features were [V14, V4, V111, V12, V10].	Screenshot 1.
Sigmoid Function						
2.1	Valid	Calculating a probability for typical z value entered.	Z = Two lists whose element wise product is between -1000 and 1000.	A decimal between 0 and 1 is returned.	0.8826. As expected.	Screenshot 2.
2.2	Boundary	Calculating a probability for a large positive z value entered	Two lists whose element wise product is more than 10^3	1.0 returned	1.0. As expected.	Screenshot 2.
2.3	Boundary	Calculating a probability for a large negative z value entered	Two lists whose element wise product is less than -10^3	0.0 returned.	0.0. As expected.	Screenshot 2.
Cost Function & Partial Derivative Functions						
3.1	Valid	Cost before parameters are tuned on a	Dataset of over 500 records, small set of features and random /	High positive cost value.	4.4197. High as expected.	Screenshot 3

		preselected set of features.	null value parameters.			
3.2	Valid	The partial derivative cost function is the same value as one calculated using the first principles formula.	Comparing two values. The two cost values we are finding the difference between should have a very small change in one of the parameter values,	The two numbers are approximately the same with little to no error.	Both were 1.6659 with a 0.00004% difference. As expected.	Screenshot 3.
Gradient Descent Procedure						
4	Valid	Cost value significantly decreases after tuning the parameters.	The dataset to train the model on and list of features to train parameters for are passed into the subroutine.	A list tuned parameters is returned. Difference between two cost values is large.	Cost before: 0.693 Cost after: 0.219. Decreased by: 68.4% As expected.	Screenshot 4.
Testing & Evaluative Functions						
5.1	Valid	Predict a class for every record in a given dataset.	The dataset to be tested is passed into the subroutine.	Returns the tested dataset with no errors..	As expected.	Screenshot 5
5.2	Valid	Get a count of the number of instances of false, true positives and negatives in the tested records.	The tested dataset is passed into the subroutine.	Returns a tuple containing four values: number of true, false positive negatives.	TP: 432, TN: 482, FP: 10 and FN: 60. As expected.	Screenshot 5
Recall / Precision / F1						
6.1	Valid	The recall metric is successfully produced.	Valid tuple of 4 positive integers entered as results.	Return the metric as a decimal between 0-1. (inclusive).	As expected.	Screenshot 6.
6.2	Valid	The precision metric is successfully produced.	Valid tuple of 4 positive integers.	Return the metric as a decimal between 0-1. (inclusive).	As expected.	Screenshot 6.
6.3	Bounda	Exception is made	Valid tuple of four	Print a message	As expected.	Screenshot 6.

	ry	when a valid tuple causes a division by zero error.	integers but true positives and false negatives are equal to zero.	that no true positives and false negatives present so cannot calculate.		
6.4	Boundary	Exception is made when a valid tuple causes a division by zero error.	Valid tuple of four integers but true positives and false positives are equal to zero.	Print a message that no true positives and false positives present so cannot calculate.	As expected.	Screenshot 6
6.5	Valid	F1 score successfully calculated.	Valid tuple of 4 positive integers.	Return the metric as a decimal between 0-1. (inclusive).	As expected.	Screenshot 6
6.6	Boundary	Exception is made when a valid tuple causes a division by zero error.	Valid tuple of four integers but true positives and, false positives and negatives are equal to zero.	Print a message that no true positives and false negatives present so cannot calculate.	As expected.	Screenshot 6
K-Fold Cross-Validation						
7	Valid	A model is tested on three folds of a dataset and the evaluation metrics are reported back.	Correct parameters passed. K is set to three during cross-validation.	Two valid average precision and recall values returned.	As expected.	Screenshot 7
Forward Feature Selection						
8.1	Valid	Features are selected based on their performance when added to a predictive model.	The dataset to select features for is passed into the subroutine.	A list of the names of the selected features is returned.	4 features selected: [V14, V20, 26, 21] As expected.	Screenshot8
8.2	Valid	Compare results of prototype and selected features.	Sele			
Launching and Loading the Database						

9.1	Valid	Opening the page for the first time.	N/A	The page should show all transactions in a table.	As expected.	Video numbered 8
9.2	Valid	Page is refreshed.	Mouse click on browser's refresh button.	The web page is refreshed with all the stored data loaded back.	As expected.	Video 9.1
9.3	Valid	Page is closed and reopened.	Mouse clicks on the close tab button and then re-enters the URL of the web-app.	The web page is reloaded with all the stored data loaded back.	As expected.	Video 9.2
Classifying Records						
10.1	Valid	Cursor clicks on classify as non-fraud (0).	Mouse click on button.	Transaction should be permanently classified as non-fraud.	As expected.	Video 10.1
10.2	Valid	Cursor clicks on classify as fraud (1)	Mouse click on the classify button.	Transaction should be permanently classified as fraud.	As expected.	Video 10.2
10.3	Valid	Cursor clicks on classify as undecided (NaN)	Mouse click on button.	Transaction should be permanently classified as undecided.	As expected.	Video 10.3
Opening info button on a tab						
11	Valid	Cursor clicks the info button for the transaction.	Mouse click on button.	A window containing all the attributes V1-V28 are displayed for a transaction.	As expected.	Video 11
Choose date on time series						
12	Valid	Cursor clicks on a date for which data	Mouse click on calendar date.	New time series for that day is	As expected.	Video 12

		in the table exists.		loaded.		
Choose axis on 2-variable scatter comparing fraud and non-fraud						
13.1	Valid	Users can look for clustering of fraud / non-fraud cases between V14 and V16.	Mouse click on the menu.	Scatter plot (colour coded by orange and blue for fraud and non-fraud) is displayed for two variables.	As expected.	Video 13.1
13.2	Valid	Users can look for clustering of fraud / non-fraud cases between V4 and V10.	Mouse click on the menu.	Scatter plot (colour coded by orange and blue for fraud and non-fraud) is displayed for two variables.	As expected.	Video 13.2
13.3	Boundary	Users can find the clustering for fraud / non-fraud cases for one variable by looking at the same axes.	Mouse click on the menu.	Should display a straight line and fraud and hopefully non-fraud are grouped.	As expected.	Video 13.3
Filtering Transactions						
14.1	Valid	Search for transactions with fraudulent probability between 0.75 and 0.90.	Max prob: 0.75 and min prob: 0.90.	All transactions with fraudulent probability between 0.75 and 0.90 are displayed to the table.	As expected.	Video compilation 14
14.2	Valid	Search for all transactions with an amount more than €500.00.	Min amount: 500.	All transactions with an amount more than €500.00 are displayed to the table.	As expected.	Video compilation 14
14.3	Valid	Search for all undecided classification	Uncheck fraud and non-fraud.	All transactions that are of the undecided	As expected.	Video compilation 14

		transactions.		classification are displayed in the table.		
14.4	Valid	Perform a daily interval search to search for all transactions between 13:00 and 14:00 across the dataset.	Daily interval checked. Start time: 13:00. End time: 14:00.	All transactions between 13:00 - and 14:00 on both days are displayed in the table.	As expected.	Video compilation 14
14.5	Valid	Perform a periodic search to find all transactions between 21:00 - 01/09/2013 to 3:00 - 02/09/2013.	Start time: 21:00. Start date: 01/09/2013 End time: 03:00. End date: 02/09/2013.	All transactions between 21:00 - 01/09/2013 to 3:00 -02/09/2013. are displayed in the table.	As expected.	Video compilation 14
14.6	Invalid / Absent	No classification categories are attempted to be selected.	Mouse clicks to uncheck all classifications.	A message should pop up warning the user of this and cancel the filter request.	As expected.	Video compilation 14
14.7	Invalid	Minimum probability is greater than the maximum probability causes error.	Max prob: 0.45 and min prob: 0.10.	A message should pop up warning the user of this and cancel the filter request.	As expected.	Video compilation 14
14.8	Invalid	Minimum amount is greater than the maximum amount causes the error.	Min amount: 50 Max amount: 10.	A message should pop up warning the user of this and cancel the filter request.	As expected.	Video compilation 14
14.9	Invalid	Start date is after the end date entered causes error.	E.g start date: 02/09/2013 and end date: 01/09/2013.	A message should pop up warning the user of this and cancel the filter request.	As expected.	Video compilation 14
14.10	Invalid	Min probability is greater than the	E.g start date: 02/09/2013 and end	Multiple warning messages should	As expected.	Video compilation 14

		max probability and start date is after the end date entered. Causes multiple error messages.	date: 01/09/2013. Max prob: 0.10 and min prob: 0.40.	pop up and cancel the filter request.		
Reset / Evaluate the Model						
15.1	Valid	User clicks the reset button.	Mouse click on the reset button.	Goes to the evaluation page. Once the dashboard is loaded, a new model is in place.	As expected. However it had a long wait time.	Video 15.
15.2	Valid	Evaluation metrics for both user and predictive model performances are displayed.	N/A.	Shows two confusion matrices and two sets of evaluation metrics to 3.d.p.	As expected.	Video 15.
15.3	Valid	The user can go back to the new model from the evaluation page.	Mouse click on the hyperlink.	Loads dashboard page with new model.	As expected.	Video 15.

Testing Evidence

Screenshot 1

Testing phase

1. Data preparation and prototype model creation.

Prepare the data for analysis and testing by removing dupicates and rows with missing values.

In [33]:
```
# Import the database - read the csv
transactions = pd.read_csv("creditcard.csv")
transactions["V0"] = 1
transactions['ID'] = list(range(1, len(transactions) + 1))
```

1.1 Any duplicate records present in the dataset are removed.

In [34]:
```
duplicates = transactions[transactions.duplicated()]
print("Number of duplicate records:", len(duplicates))
```

```
Number of duplicate records: 0
```

1.2 Any records with missing fields are removed.

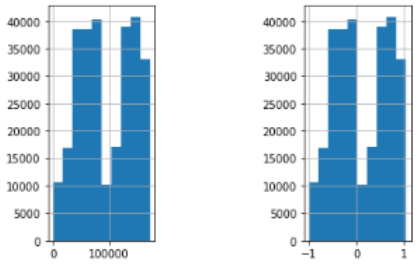
In [35]:
```
# the isnull method detects missing values for an array-like object.
print("Any missing values?", transactions.isnull().values.any())
```

```
Any missing values? False
```

1.3 Amount and time columns are standardised in preparation for correlation analysis

In [65]:
```
print("Time before and after:")
plt.subplot(1, 3, 1)
transactions['Time'].hist() # before

standardiseColumn('Time', transactions)
standardiseColumn('Amount', transactions)

plt.subplot(1, 3, 3)
transactions['Time_Scaled'].hist() # after

plt.show()
```

```
Time before and after:
```



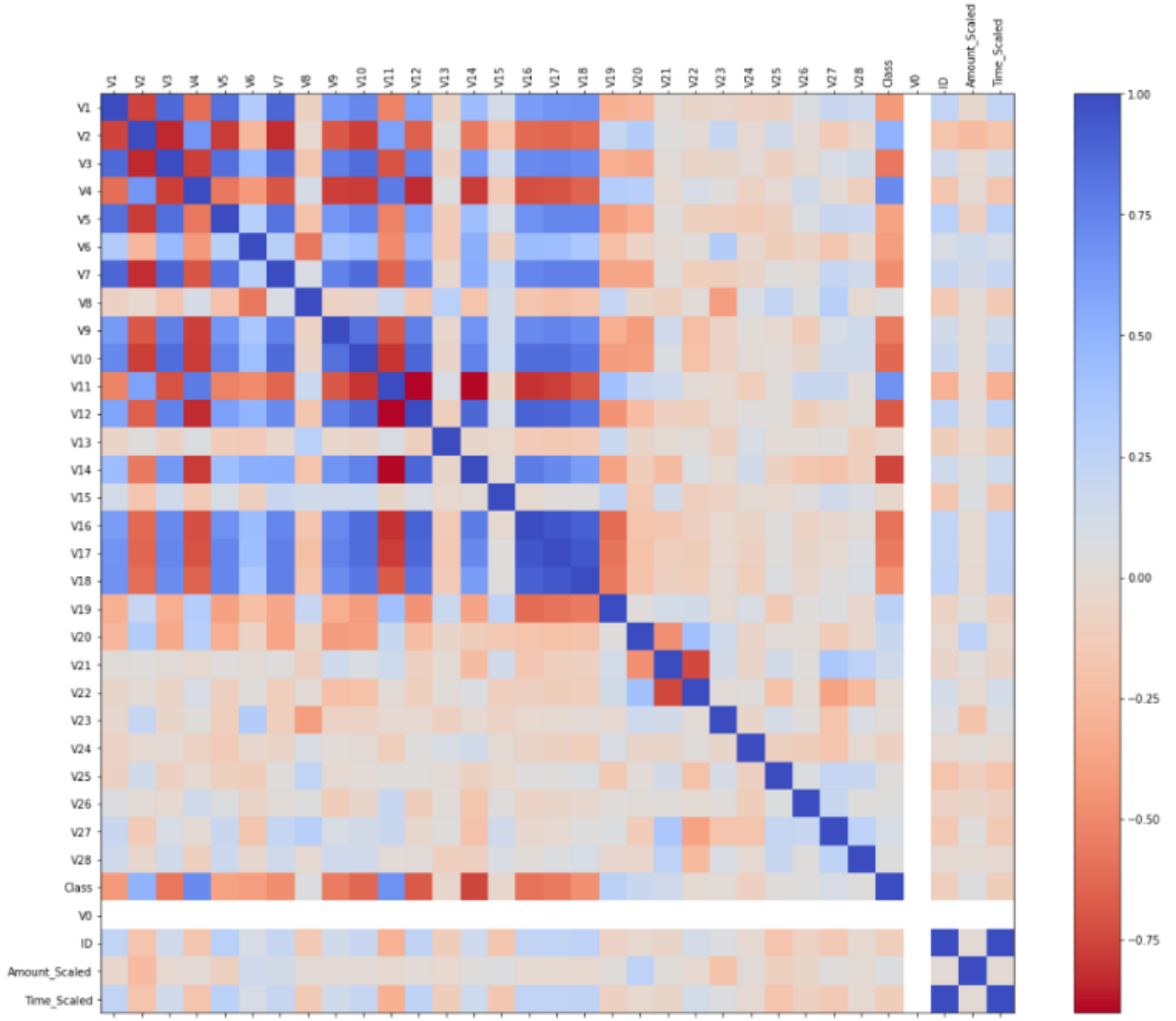
1.4 Dataset is balanced for testing and iterative development.

In [66]:
```
# Separating the portion of records that were fraulent
fraud = transactions.loc[transactions.Class == 1]

# Get the amount of fraudulent records
fraudCount = len(fraud)

# Take the random sample of non-fraudulent records
non_fraud = transactions.loc[transactions.Class == 0].sample(fraudCount)

frames = [fraud, non_fraud]
dataset = pd.concat(frames).sample(frac=1).reset_index(drop=True) # shuffle again

print("No. fraud records: ", fraudCount, "; No. authentic records: ", len(non_fraud))
```

```
No. fraud records:  492 ; No. authentic records:  492
```

1.5 Correlation matrix is produced and the feature most correlated to the fraud class becomes the prototype model.

In [69]:
```
correlationMatrix(dataset)
```



In [85]:
```
# abs returns the absolute (positive) values, unstack lays it out as a list
correlatedPairs = dataset.corr().abs().unstack()["Class"].sort_values(ascending = False)[1:6]

print("\nThe 5 most correlated features with class are:")
print(correlatedPairs)
```

```
The 5 most correlated features with class are:
V14    0.749897
V4     0.710553
V11    0.687400
V12    0.679338
V10    0.632996
dtype: float64
```

In [93]:
```
print("The most correlated feature with 'Class' was 'V14' which has a PMCC of ", correlatedPairs[0].round(4))
```

```
The most correlated feature with 'Class' was 'V14' which has a PMCC of  0.7499
```

Screenshot 2

`2.1 Typical z value entered.` --> expecting a probability between 0.01 and 0.99

In [97]:
```
parameters = [random.uniform(-1, 2) for x in range(3)]
result = sigmoid(trainingExample, features, parameters)
print("Test result of 2.1 is:", result)
```

```
Test result of 2.1 is: 0.9294440400941342
```

- `2.2 Large positive z value entered.` --> expecting a probability of 1.0

In [98]:
```
parameters = [1000, 0, 0]
result = sigmoid(trainingExample, features, parameters)
print("Test result of 2.2 is:", result)
```

```
Test result of 2.2 is: 1.0
```

- `2.3 Large negative z value entered.` --> expecting a probability of 0.0

In [99]:
```
parameters = [-1000, 0, 0]
result = sigmoid(trainingExample, features, parameters)
print("Test result of 2.3 is:", result)
```

```
Test result of 2.3 is: 0.0
```

Screenshot 3

3.1 The Cost Function

The cost function returns the average error between predicted outcomes compared with the actual outcomes.

`3.1 Randomised parameters` --> expecting a high cost value

```
randomFeatures = ['V0', 'Amount', 'V28', 'V1']
randomParameters = [random.uniform(-2, 2) for x in range(5)] # randomised parmeters
result = cost(dataset, randomFeatures, randomParameters)
print("Test result: ", result)
```

```
Test result:  3.4971273370866016
```

3.2. Partial Deriavtaive of Cost Function

With respect to a small change (h) in the value of a parameter (x) for a feature, calculate the rate of change of the cost function

`3.2 Partial Derivative is Valid` --> Derivative calculated using first principles formula and then again using the function. If they are approximately close there should be no error.

```
# preparation
testFeatures = ['V0', 'V14', 'V17']

# calculated (F(x+h) - F(x)) / h for a change in x = V17
h = 0.0000001
calculatedCostPD = (cost(dataset, testFeatures, [0, 0, 0 + h]) - cost(dataset, testFeatures, [0, 0, 0])) / h

# actual value
actualCostPD = costPartialDerivative(dataset, 'V17', testFeatures, [0, 0, 0])

# compare the two
print("Expected: ", calculatedCostPD)
print("Actual: ", actualCostPD)

error = (calculatedCostPD - actualCostPD) / actualCostPD
print("Error: ", error * 100, "%")
```

```
Expected:  1.6597514740368524
Actual:  1.6597507658560766
Error:  4.26678987211601e-05 %
```

Screenshot 4

4. Gradient Descent Algorithm

Optomise the values of a set of parmaeters to minimise the cost value.

```
# preparation
testFeatures = ['V0', 'V14', 'V20']

# Tune parameters
tunedParameters = train(dataset, testFeatures)
print("Parameters for: ", testFeatures, " are: ", tunedParameters)

# Compare the cost values before and after gradient descent
beforeDescentCost = cost(dataset, testFeatures, [0, 0, 0])
afterDescentCost = cost(dataset, testFeatures, tunedParameters)

print("Cost before: ", beforeDescentCost)
print("Cost after: ", afterDescentCost)

improvement = (beforeDescentCost - afterDescentCost) / beforeDescentCost
print("Cost value descreased (accuracy improved by): ", improvement * 100, "%")
```

```
Parameters for:  ['V0', 'V14', 'V20']  are:  [-2.1472845657133877, -1.1922781286479789, 0.3720547130394305]
Cost before:  0.6931471805599329
Cost after:  0.22473949914743854
Cost value descreased (accuracy improved by):  67.57694390881151 %
```

Screenshot 5

5. Testing / Evaluation

5.1 Predict a class for ever record. (Using the recently tuned parameters).

```
testedDataset = test(dataset, testFeatures, tunedParameters)
print(testedDataset["Predicted_Class"])
```

```
0      0
1      0
2      0
3      1
4      1
      ..
979    0
980    0
981    0
982    1
983    0
Name: Predicted_Class, Length: 984, dtype: int64
```

5.2 Get a count of the number of instances of false, true positives and negatives

```
testResults = evaluate(testedDataset)
tp, tn, fp, fn = testResults
print("True Positives: ", tp,
     "\nTrue Negatives: ", tn,
     "\nFalse Positives: ", fp,
     "\nFalse Negatives: ", fn)
```

```
True Positives:  433
True Negatives:  483
False Positives:  9
False Negatives:  59
```

Screenshot 6

6. Precision / Recall

- Two evaluation metrics

`6.1 Calculate recall using recently tested dataset:` --> decimal (0-1)

```
print("Recall: ", recall(testResults))
```

```
Recall:  0.8800813008130082
```

`6.2 Calculate precision using recently tested dataset:` --> decimal (0-1)

```
print("Precision: ", precision(testResults))
```

```
Precision:  0.9796380090497737
```

`6.3 Recall boundary case - no true positives or false negatives:` --> 0 returned (could not calculate)

```
boundaryResults = (0, 1, 1, 0)
recall(boundaryResults)
```

```
Can't calculate recall as there were no true positives or false negatives.
```

```
0
```

`` `6.4 Precision boundary case - no true positives or false positives:` --> 0 returned (could not calculate) ``

```
boundaryResults = (0, 1, 0, 1)
precision(boundaryResults)
```

```
Can't calculate precision as there were no true positives or false positives.
```

```
0
```

Screenshot 7

First Iteration:

7. K-fold Vaidation

Comparing an average of the evaluation scores for how the model performed on the training sets and on the testing sets.

`A model is tested on three folds of a dataset and the evaluation metrics are reported back.:` --> two decimals (0-1)

```
avrgPrecision, avrgRecall = cross_validation(testFeatures, 3, dataset)
print("Average Precision: ", avrgPrecision,
      "\nAverage Recall: ", avrgRecall)
```

```
Average Precision:  0.9753544494720965
Average Recall:  0.8785131490181242
```

Second iteration:

Screenshot 8

Videos 9.1 - 15.3

https://drive.google.com/drive/folders/10b-nbzTcmF0aWuDH1FqdbHMy6xbLEzm4?usp=sharing

Objectives Met

During the analysis phase, we outlined 4 basic objectives which outlined the desired features of our program.

Objective Name
No. 1: Data Preparation and Correlation Analysis. (✷)
Outcome (Test Results)
1. No duplicate records were found in test 1.1. Hence no records needed to be removed. **Success.** 2. In test 1.2, no records containing fields with missing values were found so again no records were removed. **Success.** 3. In test 1.3. the amount and time columns were standardised in preparation for correlation analysis. The before and after histograms in screenshot 1 show its success since its range was converted to between 0-1. **Success.** 4. In test 1.4, we extracted all the fraudulent transactions from the dataset, counted them, and randomly selected the same number of non-frauds from the large dataset. We shuffled the order one more time to reduce the bias during training. This formed the balanced dataset we ended up using for the rest of the duration of the project. **Success.** 5. In test 1.5 the correlation matrix displayed identified that features V14, V4, V11, V12 and V10 were most correlated to class. Using a high correlation filter, we picked the firstmost with the highest PMCC (0.75) and discarded the others to form our prototype model. **Success.**
Any limitations to the design? How could this be fixed / improved?
Since Principal Component Analysis had already been performed on the raw data to select the best features for the large dataset, there wasn't much data that needed to be prepared since they reduced thousands of features down to 30 and normalised their values in the process. After PCA, other methods could have been used to select prototype features such as the Chi-Square Test of Independence which determines whether there is an association between categorical variables. Our correlation matrix used Pearson's Moment Correlation Coefficient which later proved to be successful since even our prototype

model had high evaluation metrics. Therefore no improvements could have been made to the design to better meet the success criteria.

Did the solution meet the requirements?

By the end of this module, the dataset was free of missing values and duplicate records (criteria I), the features which formed the prototype model were chosen by analysing the correlation matrix (criteria II) and a dataset was balanced for use throughout the rest of the webapp and ML algorithms development. (Criteria III)

Objective Name

No. 2: Tune parameters to maximise model accuracy for a set of given features. **(✷)**

Outcome (Test Results)

1. The sigmoid function could calculate a fraudulency probability given a set of parameterized features for typical feature / parameter values (test 2.1) as well as extreme values (tests 2.2-2.3) - extreme values would tend to 0% or 100% without causing zero errors. **Success.**

2. Test 3.1 showed the fact that the cost function would produce a high value for a set of randomly generated parameters (untrained model). **Success.**

3. In test 3.2 we proved that the partial derivative cost function was authentic since it produced the same value as one calculated using the first principles derivative formula. The difference between the predicted and actual was less than 0.000005% ≈ 0. No error. **Success.**

4. Test 4 legitimised the fact that our implementation of Gradient Descent optimized the value of parameters to reduce the cost to a minimum, thus training the model. In the test the cost value dropped by nearly 70%. **Success.**

Any limitations to the design? How could this be fixed / improved?

Despite no major flaws and all tests passing with confidence, there were some drawbacks to our design. For example, the V0 feature being stored in the database was essentially a hack. It represented a constant value to enable the equation of the decision boundary to be optimised accurately. It doesn't need to hold the same value (1) for every single record - this is redundant. A selection statement could be added into all 3 functions that if V0 was being dealt with, then replace value with 1 instead of indexing the database. This made unnecessary trips to index the database.

A greater drawback was the amount of time it took GD to converge. This was caused by the use of a fixed learning rate value instead of one that should have increased after each step towards the minima. Newer more complex training algorithms have an adaptive learning rate. However this would have increased the programming and time complexity from its current O(n^2) to a greater power. This might have not actually improved the average time taken. Fortunately, this function was only run once for

every new model.

Did the solution meet the requirements?

The model's accuracy was not affected by the above limitations. This is confirmed by the significant decrease seen in test 3 (70%) after calculating the cost values for before and after tuning the parameters for a set of relevant features. Therefore our training algorithm's ability to tune parameters in order to obtain a high accuracy was accomplished since the lack of time/space efficiency was not a main priority and did not stunt the program's success.

Objective Name
No. 3: Feature selection by testing and evaluation. (✷)
Outcome (Test Results)

1. In test 5.1, the model predicted a fraudulency probability and class for every row in the database. Screenshot 4 shows a sample of predicted classes for the start and end of the dataset. **Success.**

2. In test 5.2, the newly predicted classes were compared to the actual classes of the transactions in order to enumerate the number of true /false - positive /negatives and thereby evaluate the performance of the model. The relative low number of false positives and false negatives in screenshot 4 showed that the model was accurate. **Success.**

3. The results obtained in test 5.2 were then used to calculate two evaluation metrics in tests 6.1-6.2 which confirmed our model's effectiveness since recall was 0.89 and precision was 0.97. These metrics were applied on different datasets throughout development, and in some cases there were no true positives or false positives/negatives causing the divisor to be zero. This would cause an undefined error but tests 6.3-6.4 showed how an exception was made for this type of error which let runtime continue. **Success.**

4. Some portions of datasets contain multiple outliers which can skew the performance results. In test 7, we demonstrated this could be handled by k-fold cross-validation which took an average of multiple folds of the data to reduce the amount of bias. Screenshot 6 showed these average precision and recall values obtained over 3 folds were approximately the same as those from the test over the entire dataset in tests 6.1-6.2. **Success.**

5. In test 8, a base model (V0) was established and evaluated to obtain a benchmark performance. One by one features V1 - V28 were temporarily added to the model to gather whether the performance increased on their arrival. After iterating over all features, the one with the greatest performance gain was permanently added to the model. This was repeated until no performance gain was observed. It was reported back that features [V14, V20, 26, 21] improved the model the most. ** Compare scores to previous unselected model -- new test needed

Any limitations to the design? How could this be fixed / improved?
Recall is the number of correct results divided by the number of results that should have been returned. Precision is the number of correct results divided by the number of all returned results. The recall was 8% lower than the precision, meaning that it missed quite a few cases of fraud instead classifying them as authentics leading to more false negatives. However out of all the predictions it made, it was 97% effective. Therefore we can deduce that the model had a high tolerance for predicting a transaction as fraudulent leading to a low number of false positives. We could view this as a limitation since out of 540 frauds in our test dataset, it missed 60 (10% of them). However the model's under or over-sensitivity might be a preference of the bank and its clients. For example if we were handling the accounts of a large stake investment or central bank, it must be over-sensitive because large amounts could be lost if let through and is more likely to be targeted by online hackers. Whereas for a commercial bank where the general public are transacting small amounts the predictive model can be under sensitive since it can be more easily refunded and constantly halting transactions misclassifying them of fraud could interfere with everyday life. Regardless, no checks were set in place to stop large amounts being lost through simple checks. That was the main limitation of the results from these tests. An easy fix to this would be a cap set at €100 with a fraudulency probability on the fringe of 0.5. ** calculate amount $£ lost over total transacted.
Did the solution meet the requirements?
Despite the limitation of the model being under-sensitive, it's near 100% precision indicated that for all transactions it predicted fraud were correct. Cross validation enabled the selection algorithm to fairly select the best performing set of features without bias. In conclusion, these newly picked features improved the overall performance of the predictive model and as result the analyst can be more reliant on the programs judgement and spend less time regulating its actions and focus more on other important matters such as client relations. This was the main priority of objective 2* defined in the analysis so is therefore completely successful.

Objective Name
No. 4: User interface and creation of the web-app. **(✷)**
Outcome (Test Results)
1. In test 9.1 the command line was used to open the web-application for the first time. The flask server provided a localhost address. Once opened in the browser, the model was trained using the fixed preselected set of features (from test 8). Then the entire dataset was tested, any predicted probability over 0.55 predicted the transaction to be fraudulent and any predicted probability under 0.45 was classified as authentic. Any remaining transactions in the range of 0.50 ± 0.05 was marked as unclassified. **Success.** 2. In tests 9.2-9.3 and, we made changes to the database by classifying random records and then

refreshed or closed / reloaded the web app to check if the changes remained. These simple tests confirmed that the database was autosaved after each change which means that the analyst won't have to waste time making manual saves and backups to ensure that their progress won't be reversed. **Success.**

3. Tests 10.1-10.3 proved that every record was able to be categorised as fraudulent (1), authentic (0) or unclassified (nan) using the buttons regardless of the transactions predicted probability. Columns displayed in the table were reduced the most important attributes, e.g. date, amount, class etc while fields (V1-V28) could be displayed by the means of opening a pop up window by clicking the info button (as demonstrated in test 11). This window saved screen real estate for graphs and improved the tables readability by hiding less important complexity. **Success.**

4. Test 12 presented how the time series compared how the frequency of each type of classification (fraudulent vs authentic) fluctuates over 24 hours. The time series had a select date (calendar) menu to see the timeseries activity of past days. **Success.**

5. Tests 13.1-13.2 showed how the scatter plot (colour coded by orange and blue for fraud and non-fraud) for two features (V1 - V28) visualised how fraudulent transactions tended to cluster for different values of each feature. Test 13.3 demonstrated how the most common value of fraud for a singular feature could be found by setting both axes to the same feature. **Success.**

6. In tests 14.1-14.5, the filter form was used to search for groups of transactions with certain characteristics. In test 14.4 a daily interval search was used to display all transactions between 13:00 and 14:00. This was useful because the time series graph this time period was observed to have a high frequency of fraud which demonstrated how the graphs and filter process worked hand in hand to improve the analysts efficiency. The filter process was the only part of our program which included input from the keyboard. This freedom of input allowed for multiple types of errors and risk sending back / displaying irrelevant data which might confuse the analyst or waste server time. Invalid tests 14.6-14.10 demonstrated how form validation blocked these invalid requests: e.g test 14.7 printed an error message to warn the user they cannot enter a minimum probability greater than a maximum probability. **Success.**

7. In tests 15.1-15.3, we evaluated the performance of and resetted the model. In 15.1 the large red reset button was clicked which successfully sent us to the evaluation page. On the server, the precision and recall metrics for both the users decisions and the models decisions were calculated. A confusion matrix visualised these performances using a color bar to show contrast between the two.

Any limitations to the design? How could this be fixed / improved?

If the dataset was upscaled: ratio of fraud to non-fraud would dwarf and the fraud line on the timeseries would have miniscule unreadable height. This could be fixed by replacing the y-axis with the proportion of the hour to the total in the day for each classification. The timeseries could also have a monthly / yearly average of how much of each classification of each type of transaction occurred within a given hour. This feature would help the user to identify general trends in fraudulent behaviour. However this might also lead them to make generalisations. In addition extreme values in frequencies on certain days

due to random fraud attacks could skew average results in the long term. However giving the user the option to choose different types of averages such as medians could reduce the effects outliers. I believe that this feature's addition of increase in programming complexity in exchange for a feature with multiple possible drawbacks was not a preferred tradeoff for a program with a simple design.

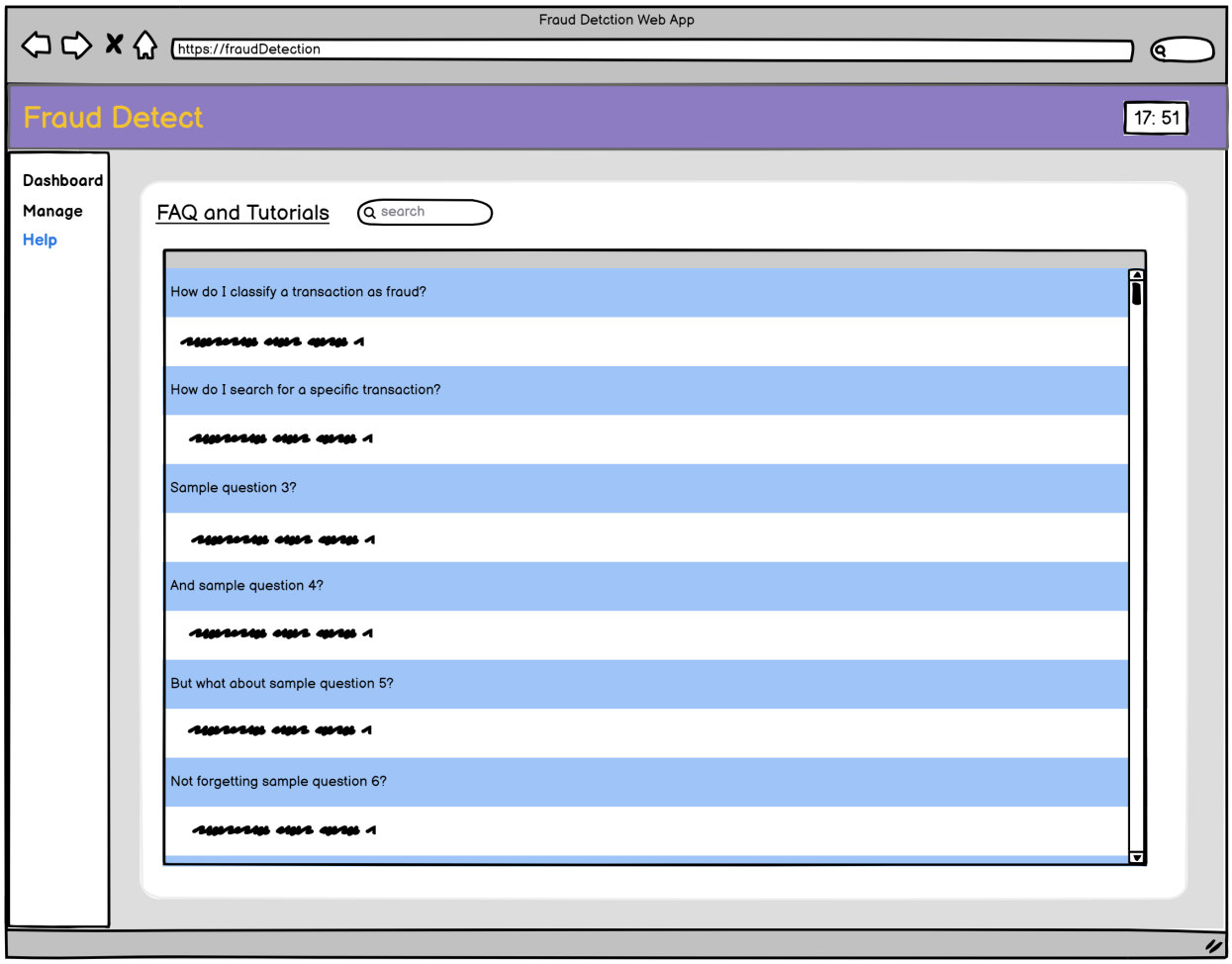
Although the filter form was the most sophisticated component of the interface, it still lacked major filter options. For example, the 2-variable plot allowed the user to identify ranges of values associated with fraud but there was no option to filter by the range of values of these V1-V28 attributes. This could be added by allowing for custom inputs to the filter form to be available to the user, each being a minimum maximum range. A general validator would be used to ensure min isn't greater than the max. Search for a transaction by ID would enable quick targeting of individual transactions which have been reported by clients. A search bar could be added added which is linked to an event handler which searches the database for the given transaction and if it can't find it passes an error message instead. Feature is not responsive and searching the server for a single transaction that might not be present could be wasteful of time.

Did the solution meet the requirements?

Not completely: criteria V.I concerning the existence of a help / documentation page was not met. This is talked about in detail below. Apart from this, all other criteria were comfortably met and talked about in detail

Unmet criteria

The main unmet success criteria was ***"The user should have a help and documentation page to guide them using the program."*** (V.I from Objective 4*). The blueprint for the help pages layout was shown in the screen design. Image repeated below:



Extent of limitations of program caused by missing feature

An analyst might potentially take longer to get used to the program and learn through experience / experimentation. However I presume this would not be a problem since there is no option to configure the models performance since we used abstraction to hide the complexities of its implementation. Also, we installed many other usability features to guide the user through design language and obviously interactive features. In the stakeholder alpha testing, Yorke reported that the program was easy to use and didn't require much direction. Therefore no complex features needed any tutorial answered by this possible FAQ help page, hence there were virtually no limitations inflicted.

Why and how this feature would be added?

As mentioned above, there weren't many hard to use or find features on our interface that needed heavy instruction. But if the possible features from the below 'Current and Future Maintenance' section and more were added, and the program was distributed to multiple banks and financial institutions then the developers would be contacted with queries more frequently.

Two solutions:

1. Manual development team driven approach. The most common questions could be answered by the ‘Fraud Detect’ development team and added to the HTML. Appropriate for a small client-base otherwise developers will be kept busy manually adding answers to the HTML.

2. Automatic community driven approach. Clients can ask for help on forums and anyone is free to answer and upvote the best answers. Developers with special accounts can still add official answers to user questions. Appropriate for a large client-base and a program with lots more features.

Broader Limitations

In a real world context, multiple fraud analysts would be working in tandem from separate accounts with different privileges. Our program failed to capture this element. In order for this to work, a login system should be implemented alongside rigorous security checks so that a high level of security is in place.

Another limitation is that the model’s performance was unable to improve during the program's runtime. This feature would be implemented in a bank's proprietary program, where the model could be periodically re-trained in real-time after a new batch of transactions are classified by the analyst or reported in by the bank's clients seeing suspicious behaviour. In our program we had to treat the database as a static copy of the bank's transactions from a moment of time. Fortunately, not having to deal with an increasing volume of transactions during runtime simplified program development, but meant our program lacks a feature similar programs have.

Also, although the current predictive model is prepared to deal with similar data sets which have the same field names and corresponding data types, introducing different databases containing new fields which identify bank accounts, new methods would have to be implemented in order to flag specific accounts containing high levels of fraudulent activity. These methods would make outliers in the spending habits of specific bank accounts easier to follow and find.

These last two limitations are not due to the design of our program, but to the lack of public databases which reflect how transactions are connected to accounts and which grow ever millisecond, like they do in the real world. This data is not open to the public in order to preserve anonymity of the bank's customers and to stop their programs from being reverse engineered so that their methods can be learnt by fraudsters and circumvented.

Current and Future Maintenance

All planned features have been implemented in the current version, which I am pleased about. Nevertheless, after extensive use, issues may arise and more features and usability improvements could have to be made.

For example, an issue pointed out by Yorke (the prime stakeholder), an issue was that the transactions listed too long and scrolling to the bottom was a tedious process. This could be improved by using pagination to let the user go to the next page.

Other possible features I conceived during discussing the limitations but deemed unessential during the analysis phase have been listed in the table below:

Possible Feature	**Justification**	**How would this be approached?**	**Potential drawbacks / limitations.**
A login / authentication process for analysts using strong encrypted passwords or biometric data.	Currently anyone on the server with the web app installed can open it without authorisation compromising banks security.	All passwords hashed and login details saved in a secure database on the server. An admin account must be in place which has the ability to issue and manage accounts for current and new employees.	Our program assumes there is only one analyst as we expect our target audience to be a small bank. Therefore added complexity doesn't cater to the target audience.
Pagination used when scrolling goes longer than the screens viewing height for long lists of records.	Save user scrolling. Improves the overall screen design - side graphs will also be in view alongside it.	Use Javascript to dynamically create a bottom menu of numbers which choose which section of transactions are displayed. Now the long table is split across multiple subpages.	None.
Real-time processing of predicting transactions which enter the simulation according to their elapsed-time.	More accurately emulate how the fraud analyst would have to deal with fraud prevention in a real world context.	Asynchronous javascript event handler sends GET fetch requests every 5 seconds to the server requesting every transaction that has taken place in that interval. Results populated in table using DOM.	Heavy processor usage for the scheduler adding transactions every second. Interrupt period might have to be extended to every minute to prevent slowing down.
Option to classify transactions due to the severity of fraud / need of reporting action. E.g. due to the amount (€)	Tiering danger of records assists the analyst in prioritizing their time to deal with the most urgent need of their	Selection buttons for choosing level of severity (e.g. 1, 2 and 3) added to each row of a fraudulent classified transaction. Option to filter by these fraud tiers should also be available.	Additional layers of complexity are not ideal for a minimal program.

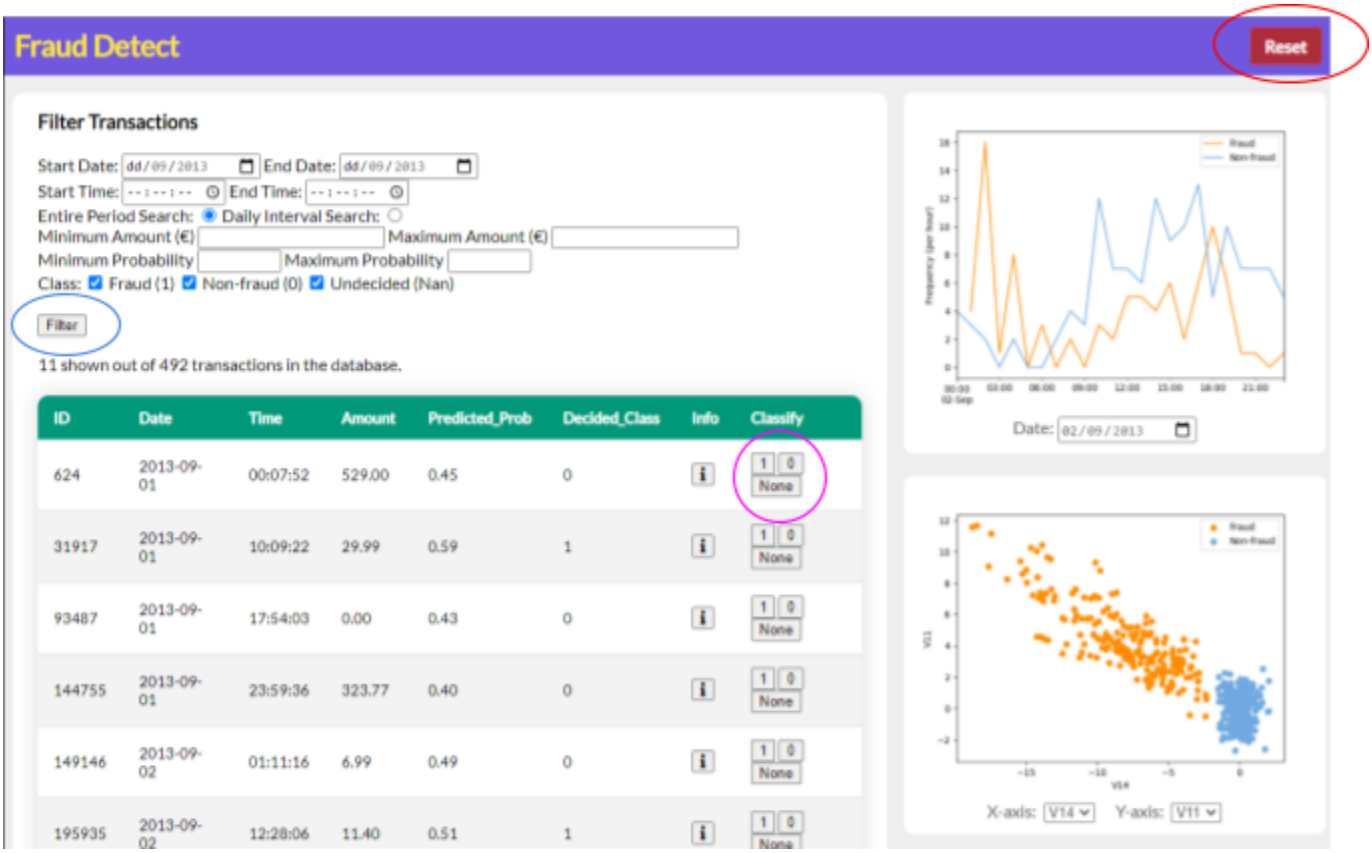
being transacted.	attention first.		

Usability Features Implemented

Buttons, inputs and menus

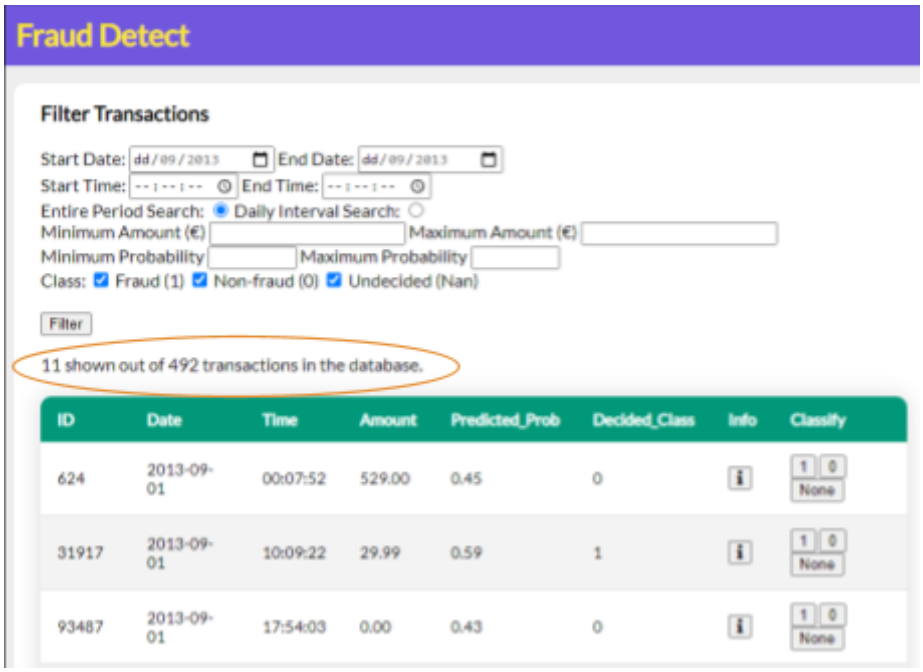
The buttons in our program are large and responsive. Buttons of greater importance such as the reset button are red to connote danger to the user so they don't accidentally reset their progress whereas buttons with neutral actions like moving navigating to the new model are coloured blue which connotes tranquility. Common use buttons (such as the classify and info buttons) are unstyled and have pale grey colours. This is because they are frequently occurring and are a repeating feature of the program which do not require any special attention from the user.

The screenshots of our web-app below highlight where buttons are used on our interface.



Labels and warning messages

A label has been added which dynamically updates after each filter operation is used to tell the user how many transactions were queried.



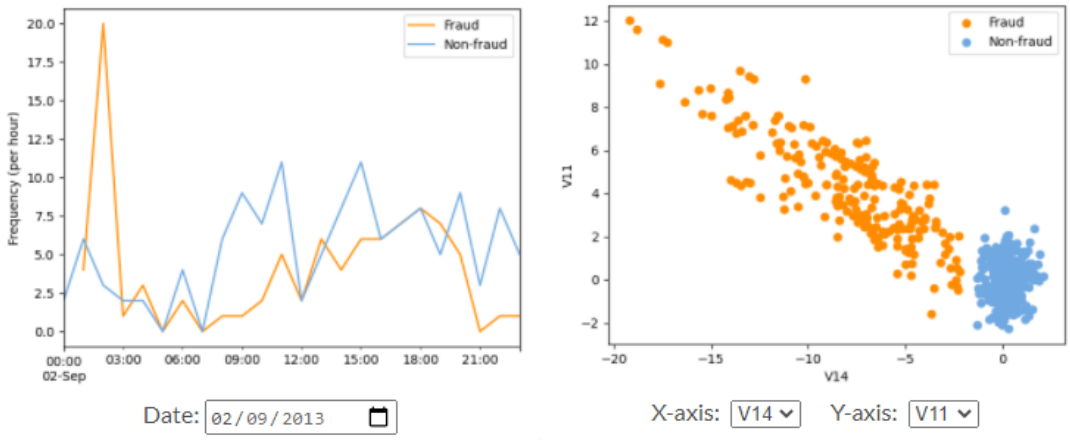
Warning messages are a method of error handling / prevention, a usability feature discussed in the discussed feature

Data visualisations

Graphs are an example of visualisation of data:

1. The time series was used to graphically indicate when fraud was most prevalent over the 24 hours of the day by the orange line which could be found by peaks of the orange line. In the left image for the 2nd of September 2013 02:00 had an unusually high level of fraud so it should be investigated.
2. The 2-variable scatter plot was the most powerful usability feature in the project because it had the ability to dynamically demonstrate how the tendency of fraudulency/ authenticity was centered around certain values for different variables as shown in video 13.

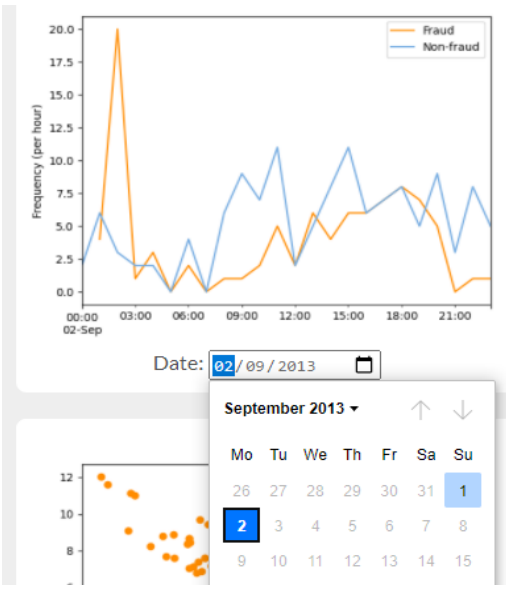
The two adjacent images below focus on how the visualizations gave the user the freedom to investigate specific trends in fraudulent activity without hard pen-to-paper analysis.

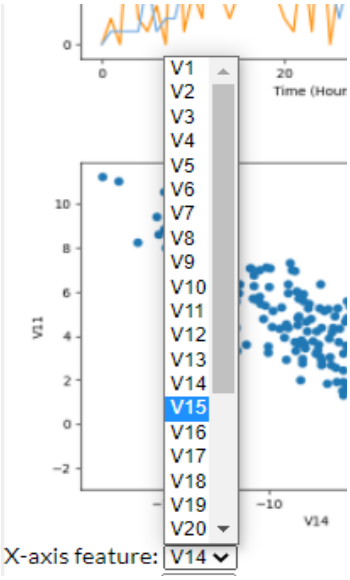


The confusion matrices (circled below) used a colour bar to show the proportions of frequencies. This could help the user quickly see if they or the model were very wrong, say if there was a dark colour anywhere but true negatives which aided in anomaly detection.

Inputs and menus

These were updated dynamically using AJAX and select inputs instead of via forms which would require full page refreshes which would make the performance and UX more clunky and slow. Thus, leading to a better usability.





Final Informal Stakeholder Interview:

I sent Yorke the final product and asked him to repeat the same set of tasks as in the Alpha testing and then asked him to reset the model. I wanted to hear his final thoughts on the web-app as a whole and if his requirements had been met

Transcript

Yorke was emailed all but the second questions prior to the interview to give him time to properly articulate his response. The interview itself was conducted on whatsapp and I have copied the responses into the docs from my phone.

Question 1

Interviewer: Now the improvements you had suggested have been made, tell me how you found using the program and what you liked most.

Yorke: I liked how the design language was simple and consistent. There was no need for 3 pages. It worked best as one It was easy to navigate and the graphs were the best feature. I had a long session classifying records and trying to break the form by putting strange queries into it - but it didn't fold. Yet again it was quick and responsive - massively beneficial in a fast paced finance environment. The reset button wasn't hard to spot and it loaded pretty quick. The colours on the evaluation page were formal and honest and the colour scale was a nice little feature for spotting where the highest concentrations of each subclassification were for me and AI. I could see that my grid had higher true positive colour so I must've picked fraud out well. I wasn't sure what recall actually meant but my score was better than the bots so I was happy.

Question 2

Interviewer: Did you have any trouble using the application? If you had the chance to change any one thing what would it be?

Yorke: Not at all this time! Still too many records and painful scrolling but the filter thing really did cut down time wasted on that. So not a massive problem. Not many places to fault it except one thing bugged me... the reload on the filter page. It slowed down the whole experience! But apart from that, great!

Question 3

Interviewer: Would you say that the usability features were successful?

Yorke: I used the graphs a lot and had no problems. I didn't run into any bugs or get stumped on trying to do difficult filters but I do believe there could be a little more direction with the difference between daily interval and period time. The buttons had a big area to press and visible which meant i

could be less precise clicking and move faster.

Question 4

Interviewer: Did you feel like the program met the expectations of it you had from the beginning? Has it met your requirements as a stakeholder for this project?

Yorke: In all honesty I thought it was going to be a bit simpler back when you started but I was impressed with how comprehensive the final thing was, more than what is needed for a small bank in my opinion. Even though I don't understand what's going on behind the scenes to make the predictions, as evident by the small amount of mis-predicted transactions seen at the final blue evaluation page, I can see that its performance is excellent. The app itself is easy to use and I'm happy with the final result.

Section 5. Project Appendices

```python
def sigmoid(trainingExample, features: list, parameters: list) -> float:
    """
    j is the index of the feature out of our selected feature set.
    n is the number of selected features
    """
    z = 0 # our input to the sigmoid (probability) function
    n = len(features)
    for j in range(n):
        featureName = features[j]
        jthFeatureVal, jthParameterVal = trainingExample[featureName], parameters[j]
        z += jthFeatureVal * jthParameterVal
    # large negative values cause a math range error
    if z < 0:
        return 1 - 1 / (1 + math.exp(z))
    else:
        return 1 / (1 + math.exp(-z))

def decision_boundary(probability):
    if probability > 0.5:
        return 1
    else:
        return 0
```

The graphs

Timeseries

```python
def createTimeseries(decisionDB, selectedDate):

    # Get all transactions from that day
    filteredDB = decisionDB[decisionDB.Date == selectedDate]

    # Setting the date as the index since the TimeGrouper works on Index, the date column is not dropped
    filteredDB['Datetime'] = pd.to_datetime(filteredDB['Datetime'])
    filteredDB.set_index('Datetime', drop=False, inplace=True)

    # Split by classification
    fraud = filteredDB[filteredDB.Decided_Class == 1]["Decided_Class"]
    non_fraud = filteredDB[filteredDB.Decided_Class == 0]["Decided_Class"]

    # Group transactions by hour, count them and plot it.
    fraud.groupby(pd.Grouper(freq='1H')).count().plot(kind='line', color="#FF8C00") # orange
    non_fraud.groupby(pd.Grouper(freq='1H')).count().plot(kind='line', color="#70a9e1") # light blue

    # label the axes
    plt.xlabel('')
    plt.ylabel('Frequency (per hour)')

    # Create legend
    plt.legend(["Fraud", "Non-fraud"])

    # load the image data into temporary file (IO)
    figfile = BytesIO()
    plt.savefig(figfile, format='png')

    # clear figure and rewind to beginning of file
    plt.clf()
    figfile.seek(0)

    # Base64 encoding is a type of conversion of bytes into ASCII characters
    figdata_png = base64.b64encode(figfile.getvalue())
    return figdata_png
```

Scatter

```
# Plotting the data - .plot(...) for line & .scatter(...) for scatter
def createScatter(dataset, feature1, feature2):

    fraud = dataset[dataset.Decided_Class == 1]
    non_fraud = dataset[dataset.Decided_Class == 0]

    # Input features to plot
    plt.scatter(fraud[feature1], fraud[feature2], color = "#FF8C00")
    plt.scatter(non_fraud[feature1], non_fraud[feature2], color = "#70a9e1")

    # label the axis
    plt.xlabel(feature1)
    plt.ylabel(feature2)

    # Create legend
    plt.legend(["Fraud", "Non-fraud"])

    # load the image data into temporary file (IO)
    figfile = BytesIO()
    plt.savefig(figfile, format='png')

    # clear figure and rewind to beginning of file
    plt.clf()
    figfile.seek(0)

    # Base64 encoding is a type of conversion of bytes into ASCII characters
    figdata_png = base64.b64encode(figfile.getvalue())
    return figdata_png
```

Confusion Matrix

```
def createConfusionMatrix(dataset, testResults, classCategory):

    # create confusion matrix
    tp, tn, fp, fn = testResults
    cf_matrix = np.array([[tn, fp], [fn, fp]]) # make 2-d array (matrix)

    # add labels
    group_names = ["True Negative", "False Positive", "False Negative", "True Positive"]
    group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]

    # add percentages
    group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten() / np.sum(cf_matrix)]
    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names, group_counts, group_percentages)]

    # create heatmap
    labels = np.asarray(labels).reshape(2,2)
    sn.heatmap(cf_matrix, annot = labels, fmt="", cmap='Blues')

    # load the image data into temporary file (IO)
    figfile = BytesIO()
    plt.savefig(figfile, format='png')

    # clear figure and rewind to beginning of file
    plt.clf()
    figfile.seek(0)

    # Base64 encoding is a type of conversion of bytes into ASCII characters
    figdata_png = base64.b64encode(figfile.getvalue())
    return figdata_png
```

Section 6. Bibliography

UK Finance (2019) *Fraud the Facts 2019.* Available at: https://www.ukfinance.org.uk/system/files/Fraud%20The%20Facts%202019%20-%20FINAL%20ONLINE.pdf (Accessed: 23 July 2020).

Career Explorer (2020) *What does a fraud analyst do?* Available at: https://www.careerexplorer.com/careers/fraud-analyst/#what-does-a-fraud-analyst-do (Accessed: 27 August 2020).

Which? (2018) *More than 96% of fraud cases go unsolved.* Available at: https://www.which.co.uk/news/2018/09/exclusive-more-than-96-of-reported-fraud-cases-go-unsolved/ (Accessed: 27 August 2020).

eInfoChips (2018) *Everything you Need to Know About Hardware Requirements for Machine Learning.* Available at: https://www.einfochips.com/blog/everything-you-need-to-know-about-hardware-requirements-for-machine-learning/ (Accessed: 18 September 2020).