Dataset

The chosen dataset for this study is the Credit Approval Data Set from the UCI Machine Learning Repository. The source for this dataset is confidential and as such, we can't say for sure, but the references used are available on the site.

Explanation and preparation of datasets

The dataset, Credit Approval Data Set form the university of California repository was downloaded in a format not readable by excel so we changed it to a CSV format. The dataset set contains 16 variables and 690 observations, which for us, represents the applicants. The variables do not have variable names because of the confidentiality of the data. This will be sorted out by attributing variable names for each of them following the descriptions of the variables. The last variable which has "+" and "-" as possible outcomes represent the class variable (dependent variable). We can see from the image below that the first observation was taken as the header because of the lack of variable names.

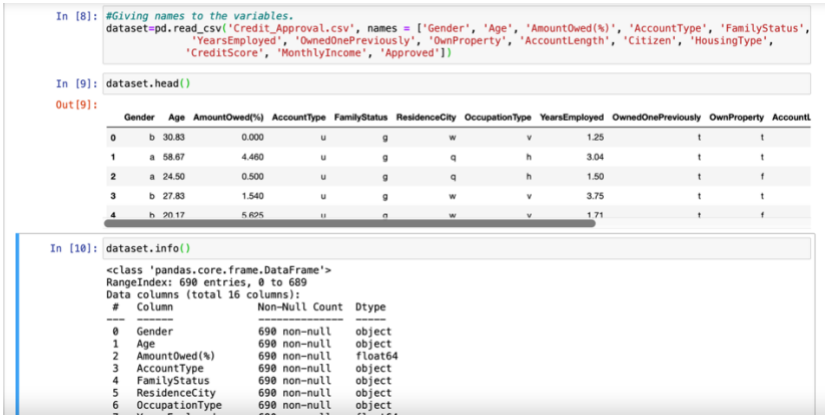
```
In [1]: #importing important libraries
        import numpy as np
        import pandas as pd
        import sklearn as sk
        import matplotlib.pyplot as plt
        import seaborn as sns

In [2]: #loading the dataset
        dataset=pd.read_csv('Credit_Approval.csv')

In [3]: dataset.head()

Out[3]:
```

	b	30.83	0	u	g	w	v	1.25	t	t.1	01	f	g.1	00202	0.1	+
0	a	58.67	4.460	u	g	q	h	3.04	t	t	6	f	g	00043	560	+
1	a	24.50	0.500	u	g	q	h	1.50	t	f	0	f	g	00280	824	+
2	b	27.83	1.540	u	g	w	v	3.75	t	t	5	t	g	00100	3	+
3	b	20.17	5.625	u	g	w	v	1.71	t	f	0	f	s	00120	0	+
4	b	32.08	4.000	u	g	m	v	2.50	t	f	0	t	g	00360	0	+

Our dataset is quite interesting as it is multivariate. The attributes are categorical, continuous, nominal with small numbers of values, and nominal with larger numbers of values. Our dataset also contains missing values. Out of our 690 observations, about 37 cases which amounts to 5%, contains missing values.

```
In [4]: dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 689 entries, 0 to 688
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   b       689 non-null    object
 1   30.83   689 non-null    object
 2   0       689 non-null    float64
 3   u       689 non-null    object
 4   g       689 non-null    object
 5   w       689 non-null    object
 6   v       689 non-null    object
 7   1.25    689 non-null    float64
 8   t       689 non-null    object
 9   t.1     689 non-null    object
 10  01      689 non-null    int64
 11  f       689 non-null    object
 12  g.1     689 non-null    object
 13  00202   689 non-null    object
 14  0.1     689 non-null    int64
 15  +       689 non-null    object
dtypes: float64(2), int64(2), object(12)
memory usage: 86.2+ KB
```

Again, we see that it has taken the first observation as the header. For this reason, we would be giving names to the variables.



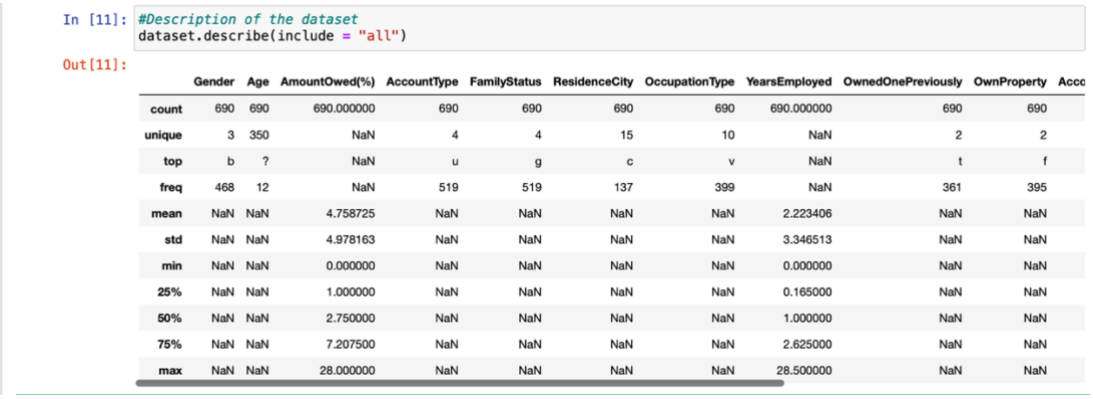
```
In [8]: #Giving names to the variables.
dataset=pd.read_csv('Credit_Approval.csv', names = ['Gender', 'Age', 'AmountOwed(%)', 'AccountType', 'FamilyStatus',
                    'YearsEmployed', 'OwnedOnePreviously', 'OwnProperty', 'AccountLength', 'Citizen', 'HousingType',
                'CreditScore', 'MonthlyIncome', 'Approved'])

In [9]: dataset.head()
```

Out[9]:

	Gender	Age	AmountOwed(%)	AccountType	FamilyStatus	ResidenceCity	OccupationType	YearsEmployed	OwnedOnePreviously	OwnProperty	AccountL
0	b	30.83	0.000	u	g	w	v	1.25	t	t	
1	a	58.67	4.460	u	g	q	h	3.04	t	t	
2	a	24.50	0.500	u	g	q	h	1.50	t	f	
3	b	27.83	1.540	u	g	w	v	3.75	t	t	
4	b	20.17	5.625	u	g	w	v	1.71	t	f	

```
In [10]: dataset.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Gender              690 non-null    object
 1   Age                 690 non-null    object
 2   AmountOwed(%)       690 non-null    float64
 3   AccountType         690 non-null    object
 4   FamilyStatus        690 non-null    object
 5   ResidenceCity       690 non-null    object
 6   OccupationType      690 non-null    object
```

The image below shows a general description of the variables. Variable 1 – Variable 15 are our independent variables and Variable 16 is our Dependent variable.



```
In [11]: #Description of the dataset
dataset.describe(include = "all")
```

Out[11]:

	Gender	Age	AmountOwed(%)	AccountType	FamilyStatus	ResidenceCity	OccupationType	YearsEmployed	OwnedOnePreviously	OwnProperty	Acco
count	690	690	690.000000	690	690	690	690	690.000000	690	690	
unique	3	350	NaN	4	4	15	10	NaN	2	2	
top	b	?	NaN	u	g	c	v	NaN	t	f	
freq	468	12	NaN	519	519	137	399	NaN	361	395	
mean	NaN	NaN	4.758725	NaN	NaN	NaN	NaN	2.223406	NaN	NaN	
std	NaN	NaN	4.978163	NaN	NaN	NaN	NaN	3.346513	NaN	NaN	
min	NaN	NaN	0.000000	NaN	NaN	NaN	NaN	0.000000	NaN	NaN	
25%	NaN	NaN	1.000000	NaN	NaN	NaN	NaN	0.165000	NaN	NaN	
50%	NaN	NaN	2.750000	NaN	NaN	NaN	NaN	1.000000	NaN	NaN	
75%	NaN	NaN	7.207500	NaN	NaN	NaN	NaN	2.625000	NaN	NaN	
max	NaN	NaN	28.000000	NaN	NaN	NaN	NaN	28.500000	NaN	NaN	

Next, we would be converting categorical variables with binary class into 1 and 0. And the ones with more outcomes, we will be giving them names. The table below summarises the attributions that we carried out.

Variable	**Old values of outcome**	**New values**
Variable 1	A, B – Male, Female	0, 1
Variable 4	U, Y, L,	Checking, Savings, Student,
Variable 5	G, P, GG	Single, Married, Divorced
Variable 6	C,D,CC,I,J,K,M,R,Q,W,X,E,AA,FF	Manchester, London, Birmingham, Derby, Chichester, Leicester, Salford, Sheffield, York, Oxford, Bristol, Exeter, Preston, Other
Variable 7	V,H,BB,J,N,Z,DD,FF,O	Data Scientist, Security Staff, Accountant, Doctor, Sale Staff, Manager, Labourer, Entrepreneur, Other

Variable 9	T, F – True, False	0, 1
Variable 10	T, F – True, False	0, 1
Variable 12	T, F – True, False	0, 1
Variable 13	G, P, S	House, Apartment, Shared Accommodation
Variable 16	+, - - Approved, Not Approved	0, 1

As we see in the image below, we have successfully converted the outputs that appeared meaningless because of the confidentiality of the data.

```
In [9]: #Changing values of the categorical variables

dataset['Gender']=dataset['Gender'].replace(['a','b'],[0,1])
dataset['AccountType']=dataset['AccountType'].replace(['u','y','l'],['Checking','Savings','Student'])
dataset['FamilyStatus']=dataset['FamilyStatus'].replace(['g','p','gg'],['Single','Married','Divorced'])
dataset['ResidenceCity']=dataset['ResidenceCity'].replace(['c','d','cc','i','j','k','m','r','q','w','x','e','aa','ff
                        ['Manchester', 'London', 'Birmingham', 'Derby', 'Chichester', 'Leicester', 'Salford',
                         'Sheffield', 'York', 'Oxford', 'Bristol', 'Exeter', 'Preston', 'Other'])
dataset['OccupationType']=dataset['OccupationType'].replace(['v','h','bb','j','n','z','dd','ff','o'],
                           ['Data Scientist', 'Security Staff', 'Accountant', 'Doctor', 'Sale Staff', 'Manager',
                            'Labourer', 'Entrepreneur', 'Other'])
dataset['OwnedOnePreviously']=dataset['OwnedOnePreviously'].replace(['t','f'],[0,1])
dataset['OwnProperty']=dataset['OwnProperty'].replace(['t','f'],[0,1])
dataset['Citizen']=dataset['Citizen'].replace(['t','f'],[0,1])
dataset['HousingType']=dataset['HousingType'].replace(['g','p','s'],['House','Apartment','Shared Accomodation'])
dataset['Approved']=dataset['Approved'].replace(['+','-'],[0,1])

In [11]: dataset.head()

Out[11]:
```

	Gender	Age	AmountOwed(%)	AccountType	FamilyStatus	ResidenceCity	OccupationType	YearsEmployed	OwnedOnePreviously	OwnProperty	AccountL
0	1	30.83	0.000	Checking	Single	Oxford	Data Scientist	1.25	0	0	
1	0	58.67	4.460	Checking	Single	York	Security Staff	3.04	0	0	
2	0	24.50	0.500	Checking	Single	York	Security Staff	1.50	0	1	
3	1	27.83	1.540	Checking	Single	Oxford	Data Scientist	3.75	0	0	
4	1	20.17	5.625	Checking	Single	Oxford	Data Scientist	1.71	0	1	

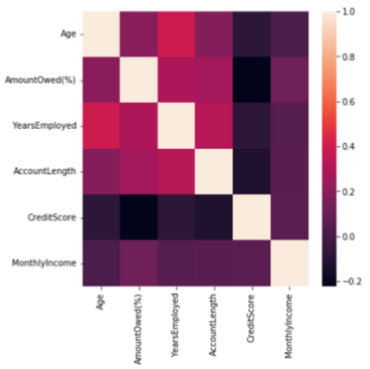
The next step would be to treat the missing values. There are a couple of ways to treat this. We can either omit the observations with any missing value and this would not affect our study seeing as the missing values are only 5% of our data. We can also replace the missing values with the mean, mode, median and for the categorical variables, we can use the most frequent value. In this study, the missing values for the numerical columns were replaced by the mean and the missing values for the categorical columns were replaced by the most frequent. We see in the figure below the columns that had missing values and the treatment of the missing values.

```
In [13]: dataset.isnull().sum()

Out[13]: Gender                 12
         Age                    12
         AmountOwed(%)           0
         AccountType             6
         FamilyStatus            6
         ResidenceCity           9
         OccupationType          9
         YearsEmployed           0
         OwnedOnePreviously      0
         OwnProperty             0
         AccountLength           0
         Citizen                 0
         HousingType             0
         CreditScore            13
         MonthlyIncome           0
         Approved                0
         dtype: int64

In [14]: #Changing the type of credit score
         dataset['CreditScore'] = dataset['CreditScore'].astype(float)

In [15]: #Treating the missing values
         from sklearn.impute import SimpleImputer
         numImputer = SimpleImputer(missing_values=np.nan, strategy='mean')
         numImputer = numImputer.fit(dataset[['Age','CreditScore']])
         dataset[['Age','CreditScore']] = numImputer.transform(dataset[['Age','CreditScore']])
         catImputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
         catImputer = catImputer.fit(dataset[['Gender','AccountType','FamilyStatus','ResidenceCity','OccupationType']])
         dataset[['Gender','AccountType','FamilyStatus','ResidenceCity','OccupationType']] = catImputer.transform(dataset[['G
```

Visualisation of dataset

A set of visualisations were created to visually understand the variables, to see the possible relationships between the variables and to detect any outliers.

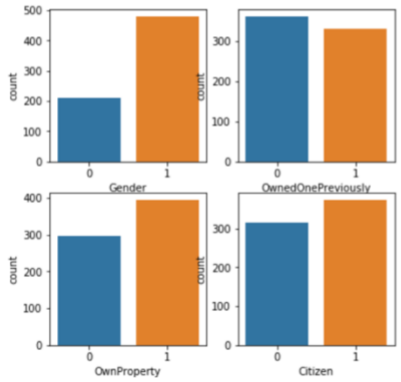
```
In [17]: #Visualisations of numerical values
         plt.figure(figsize=(6,6))
         sns.heatmap(data=dataset.corr())

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7fe89bb5c6d0>
```



We can see the correlation between the numerical variables using the figure above. We can see from the chart that some variables are negatively correlated. 1 signifies strong correlation, 0 means no correlation and -0.2 signifies weak negative correlation.

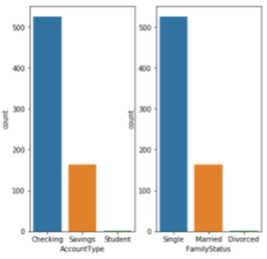
Visualisation of the Nominal Variables

```
In [26]: #Visualisation of the Nominal Variables
fig, ax = plt.subplots(2,2,figsize= (6,6))
sns.countplot(x=dataset['Gender'], ax=ax[0,0])
sns.countplot(x=dataset['OwnedOnePreviously'], ax=ax[0,1])
sns.countplot(x=dataset['OwnProperty'], ax=ax[1,0])
sns.countplot(x=dataset['Citizen'], ax=ax[1,1])

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdda98cc5d0>
```

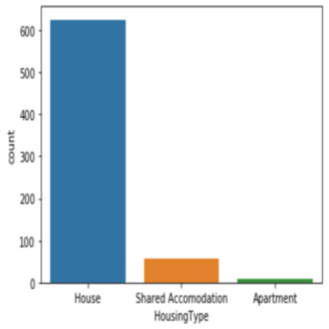


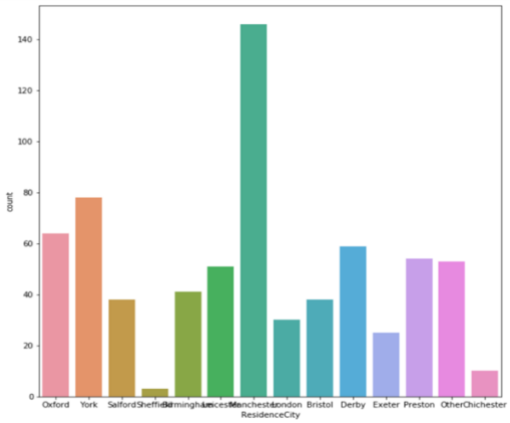
Visualisation of the categorical variables

```
In [40]: #Visualisation of the Categorical Variables
fig, ax = plt.subplots(1,2,figsize= (6,6))
sns.countplot(x=dataset['AccountType'], ax=ax[0])
sns.countplot(x=dataset['FamilyStatus'], ax=ax[1])

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdda9b896d0>
```

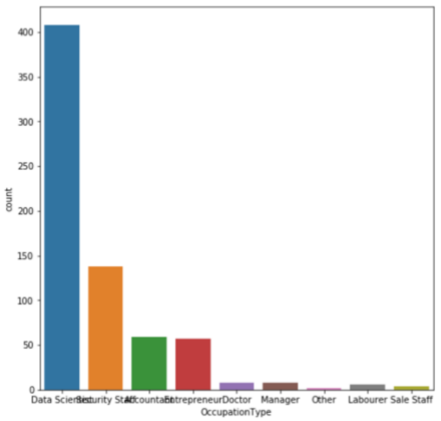



```
In [46]: sns.countplot(x=dataset['HousingType'])

Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdda9e202d0>
```



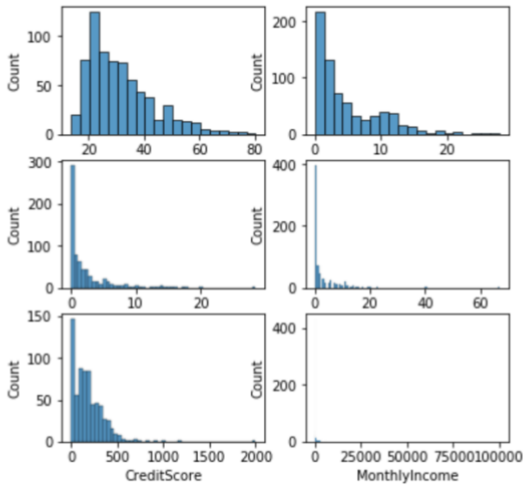


```
In [45]: plt.figure(figsize= (8,8))
         sns.countplot(x=dataset['OccupationType'])
Out[45]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdda9e27fd0>
```



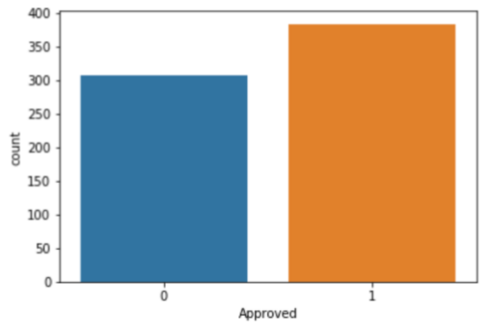
Visualisation of the Numerical Variables

```
In [53]: #Visualisation of Numerical Variables
         fig, ax = plt.subplots(3,2,figsize= (6,6))
         sns.histplot(x=dataset['Age'], ax=ax[0,0])
         sns.histplot(x=dataset['AmountOwed(%)'], ax=ax[0,1])
         sns.histplot(x=dataset['YearsEmployed'], ax=ax[1,0])
         sns.histplot(x=dataset['AccountLength'], ax=ax[1,1])
         sns.histplot(x=dataset['CreditScore'], ax=ax[2,0])
         sns.histplot(x=dataset['MonthlyIncome'], ax=ax[2,1])
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdd97ae49d0>
```



To check for class imbalance, we counted how many 1s and 0s we had for the class variable, and we see in the image below that we can go ahead with building the model as there is not a huge difference between both classes.

```
In [58]: #checking for the class imbalance
         sns.countplot(x=dataset['Approved'])
Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x7fdd9945dd10>
```



Standardization

We can evidently see from the plots that the variables are not in the same interval which can cause issues in the machine learning algorithms that we have selected. To get rid of this issue, we would be scaling the data to meet a standard normal distribution. Before we do this, we would first be splitting our dataset into two. One for training the data i.e., building the model and the other for testing the built model. We would be using the 70-30 proportion.

```
In [25]: #Label Encoding
         from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()
         for col in dataset.columns:
             if dataset[col].dtypes == object:
                 dataset[col]= le.fit_transform(dataset[col])

In [26]: #Splitting dataset
         X = dataset.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]].values
         y = dataset.iloc[:, 15].values

         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, random_state= 0)

In [27]: #standardisation
         from sklearn.preprocessing import StandardScaler
         sc=StandardScaler()
         X_train_s=sc.fit_transform(X_train)
         X_test_s=sc.fit_transform(X_test)
```

Implementation in Python / Azure Machine Learning Studio

Now that we have our dataset ready for building different machine learning classifiers, we will be using three different classification models which are Logistic Regression, Neural Network and SVM in Azure Machine Learning Designer to apply classification.

Logistic Regression

It is a classification model rather than a regression model like its name implies. It is a statistical method used to predict a binary outcome based on already existing observations of a dataset. It analyses the relationship between one or more independent variables and then predicts a dependent variable. It essentially uses a logistic function to build a model with binary outcome. Logistic regression uses a loss

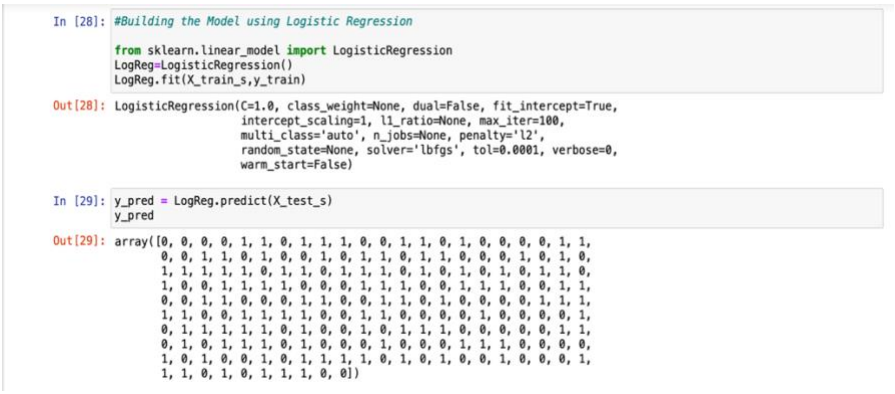
function called “maximum likelihood estimation (MLE)” which is a conditional probability and as such has its values between 0 and 1. If the probability gotten is above 0.5, then we classify it as class 0 and if it is the opposite, we classify it as class 1 (Hoss Belyadi, Alireza Haghighat, 2021).

$$Logistic\ Function = \frac{1}{1 + e^{-x}}$$

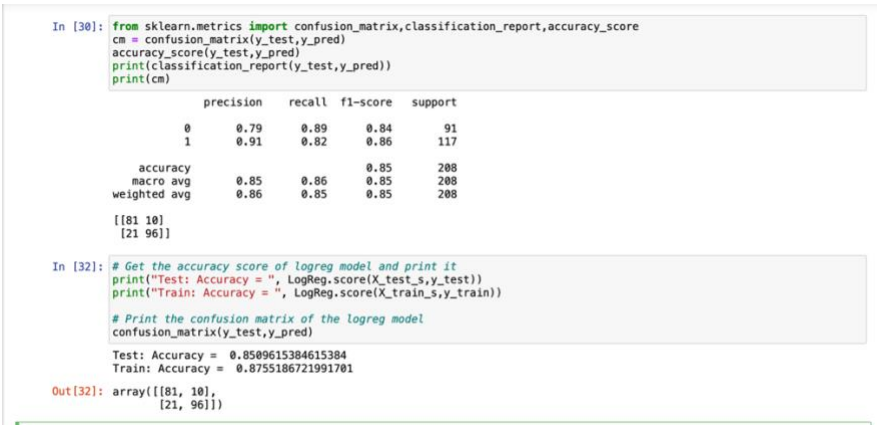
We chose to use this method because it is easy to implement and it is efficient for model training. It is also good for classification when the dependent variable is categorical which is the case for us.

Creating the Model in Python

We imported Logistic Regression from the sklearn.linear_model package and from the images below, we can see the results from building the model. For the specification of hyperparameters, we tried to use random_state = 0 and 42 but it didn’t augment the performance of our model, so we removed it and went with the default hyperparameters for the model. We also see the confusion matrix for this model. We will now use Neural Network method to perform classification and then compare both models afterwards.



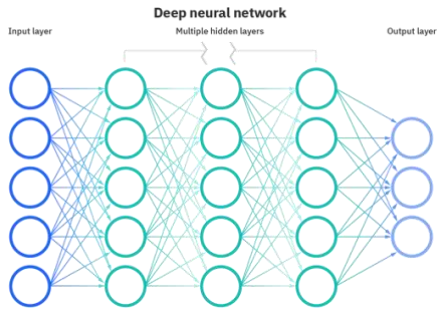
```
In [28]: #Building the Model using Logistic Regression

         from sklearn.linear_model import LogisticRegression
         LogReg=LogisticRegression()
         LogReg.fit(X_train_s,y_train)

Out[28]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='auto', n_jobs=None, penalty='l2',
                            random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                            warm_start=False)

In [29]: y_pred = LogReg.predict(X_test_s)
         y_pred

Out[29]: array([0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1,
                0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0,
                1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
                1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
                0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1,
                1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
                0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,
                1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
                1, 1, 0, 1, 0, 1, 1, 1, 0, 0])
```



```
In [30]: from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
         cm = confusion_matrix(y_test,y_pred)
         accuracy_score(y_test,y_pred)
         print(classification_report(y_test,y_pred))
         print(cm)

                       precision    recall  f1-score   support

                    0       0.79      0.89      0.84        91
                    1       0.91      0.82      0.86       117

             accuracy                           0.85       208
            macro avg       0.85      0.86      0.85       208
         weighted avg       0.86      0.85      0.85       208

         [[81 10]
          [21 96]]

In [32]: # Get the accuracy score of logreg model and print it
         print("Test: Accuracy = ", LogReg.score(X_test_s,y_test))
         print("Train: Accuracy = ", LogReg.score(X_train_s,y_train))

         # Print the confusion matrix of the logreg model
         confusion_matrix(y_test,y_pred)

         Test: Accuracy =  0.8509615384615384
         Train: Accuracy =  0.8755186721991701

Out[32]: array([[81, 10],
                [21, 96]])
```

Neural Networks

Neural networks are a series of algorithm that mimics how a brain operates to find hidden relationships between features in a dataset. They tend to resemble the connections of neurons and synapses found in the brain. They recognize patterns numerically so the dataset if not numerical, must be translated for the values to be read. They are used to classify a dataset when there is an already defined class. A simple neural network consists of three layers; input layer, hidden layer, and output layer. Which means that between the input and output layers, we could have one or more hidden layer. Most neural networks are fully connected, meaning each hidden layer and output layer is connected to every layer on both sides. And each of this connection has a weight attached to it. All inputs are multiplied by their weights, the values are then added up to get an output, which is the passed through an activation function to get an output. If this output passes a predefined threshold, it activates the node, passing it to the next layer (IBM Cloud Education, 2020).



To build the model, we will be using an optimiser and a loss function. Optimizer is how to minimise the error and the loss function is what to minimise the error. Optimizer modifies the weights and learning rate of the neural network.

Creating the Model in Python

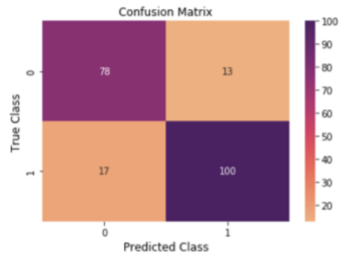
To build our model, we will be using a simple feedforward network. Our input shape will be 15 as we have 15 independent variables. In the final layer, we used the sigmoid activation function as it is good for binary class classification. The number of our neurons is 2 as it should be equal to the number of our class for our dependent variable. The loss function we used is the categorical cross entropy as it is the one mostly used for classification. The optimizer used is a common one called Adam.

```
In [30]: #Building the Model using Neural Network
         import tensorflow as tf

In [68]: model = tf.keras.models.Sequential()

         model.add(tf.keras.layers.Dense(9,activation='relu',input_shape=(15,)))
         model.add(tf.keras.layers.Dense(2,activation='sigmoid'))

In [69]: #compiling the model
         model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
                       metrics='accuracy')
         model.summary()

         Model: "sequential_4"
         _________________________________________________________________
          Layer (type)                 Output Shape              Param #
         =================================================================
          dense_8 (Dense)              (None, 9)                 144

          dense_9 (Dense)              (None, 2)                 20

         =================================================================
         Total params: 164
         Trainable params: 164
         Non-trainable params: 0
         _________________________________________________________________
```

```
In [70]: NeuNet = model.fit(X_train_s, y_train, batch_size = 8, epochs= 50, verbose=2, validation_split=0.3)

Epoch 1/50
43/43 - 1s - loss: 0.7741 - accuracy: 0.6083 - val_loss: 0.6895 - val_accuracy: 0.6276 - 862ms/epoch - 20ms/step
Epoch 2/50
43/43 - 0s - loss: 0.6729 - accuracy: 0.6706 - val_loss: 0.6164 - val_accuracy: 0.6966 - 160ms/epoch - 4ms/step
Epoch 3/50
43/43 - 0s - loss: 0.5994 - accuracy: 0.7033 - val_loss: 0.5743 - val_accuracy: 0.6966 - 297ms/epoch - 7ms/step
Epoch 4/50
43/43 - 0s - loss: 0.5463 - accuracy: 0.7359 - val_loss: 0.5423 - val_accuracy: 0.7103 - 246ms/epoch - 6ms/step
Epoch 5/50
43/43 - 0s - loss: 0.5078 - accuracy: 0.7715 - val_loss: 0.5221 - val_accuracy: 0.7103 - 263ms/epoch - 6ms/step
Epoch 6/50
43/43 - 0s - loss: 0.4781 - accuracy: 0.7864 - val_loss: 0.5022 - val_accuracy: 0.7379 - 307ms/epoch - 7ms/step
Epoch 7/50
43/43 - 0s - loss: 0.4519 - accuracy: 0.7982 - val_loss: 0.4867 - val_accuracy: 0.7655 - 332ms/epoch - 8ms/step
Epoch 8/50
43/43 - 0s - loss: 0.4294 - accuracy: 0.8190 - val_loss: 0.4726 - val_accuracy: 0.7724 - 155ms/epoch - 4ms/step
Epoch 9/50
43/43 - 0s - loss: 0.4107 - accuracy: 0.8249 - val_loss: 0.4633 - val_accuracy: 0.7655 - 308ms/epoch - 7ms/step
Epoch 10/50
43/43 - 0s - loss: 0.3933 - accuracy: 0.8309 - val_loss: 0.4537 - val_accuracy: 0.7862 - 290ms/epoch - 7ms/step
Epoch 11/50
43/43 - 0s - loss: 0.3792 - accuracy: 0.8487 - val_loss: 0.4447 - val_accuracy: 0.8069 - 178ms/epoch - 4ms/step
Epoch 12/50
43/43 - 0s - loss: 0.3663 - accuracy: 0.8605 - val_loss: 0.4373 - val_accuracy: 0.8069 - 405ms/epoch - 9ms/step
Epoch 13/50
43/43 - 0s - loss: 0.3558 - accuracy: 0.8694 - val_loss: 0.4319 - val_accuracy: 0.8000 - 263ms/epoch - 6ms/step
Epoch 14/50
43/43 - 0s - loss: 0.3450 - accuracy: 0.8754 - val_loss: 0.4276 - val_accuracy: 0.8138 - 203ms/epoch - 5ms/step
Epoch 15/50
43/43 - 0s - loss: 0.3362 - accuracy: 0.8754 - val_loss: 0.4232 - val_accuracy: 0.8276 - 279ms/epoch - 6ms/step
Epoch 16/50
43/43 - 0s - loss: 0.3302 - accuracy: 0.8694 - val_loss: 0.4197 - val_accuracy: 0.8276 - 201ms/epoch - 5ms/step
Epoch 17/50
```

We can see the model built from the images above. Now we'll look at the confusion matrix and the accuracy measurements.

```
In [74]: confusion_matrix_NeuNet = confusion_matrix(y_test,y_pred_NeuNet)
ax = sns.heatmap(confusion_matrix_NeuNet, cmap='flare',annot=True, fmt='d')
plt.xlabel("Predicted Class",fontsize=12)
plt.ylabel("True Class",fontsize=12)
plt.title("Confusion Matrix",fontsize=12)
plt.show()
```



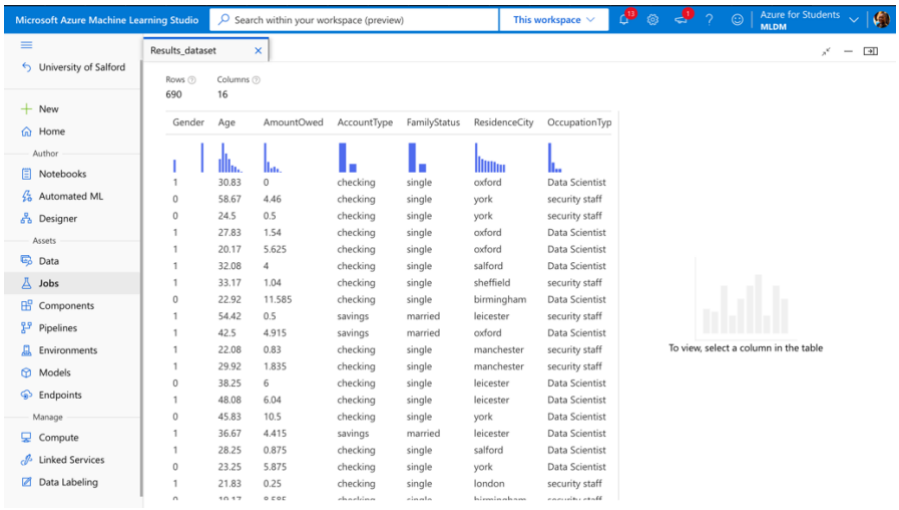

```
In [75]: print(classification_report(y_test,y_pred_NeuNet))

              precision    recall  f1-score   support

           0       0.82      0.86      0.84        91
           1       0.88      0.85      0.87       117

    accuracy                           0.86       208
   macro avg       0.85      0.86      0.85       208
weighted avg       0.86      0.86      0.86       208
```

Azure Machine Learning Designer to apply classification

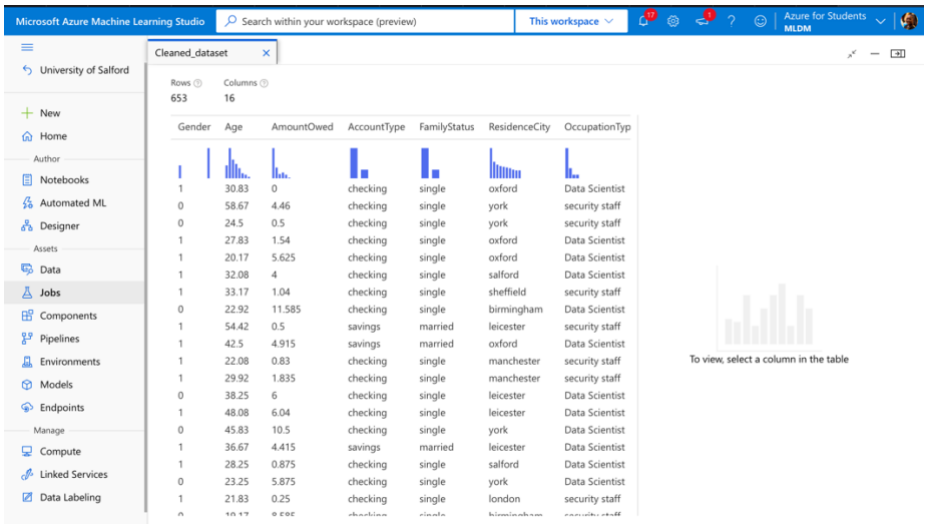
Azure machine learning is a cloud-based service used for deploying machine learning tasks. It helps simplify some of the tasks involved in building a model from data pre-processing to training a model to predictions made with the built models. It is very easy to use with its drag and drop feature and we can prepare data, select features, train models, and evaluate models using this feature. In Azure machine learning studio, there are several classification techniques available and there are different ways to create classification models. One way is to use a visual interface called designer and that is what we'll be using for our study, today. Here, we will build a project called a Pipeline. There is a blank canvas where we drag, drop, and connect the components we need for building and on the left is a panel that contains all the required tools. The pipeline begins from the dataset that we want to train on (Mike Cornell, 2015).

Creating the Model in Azure Machine Learning Studio

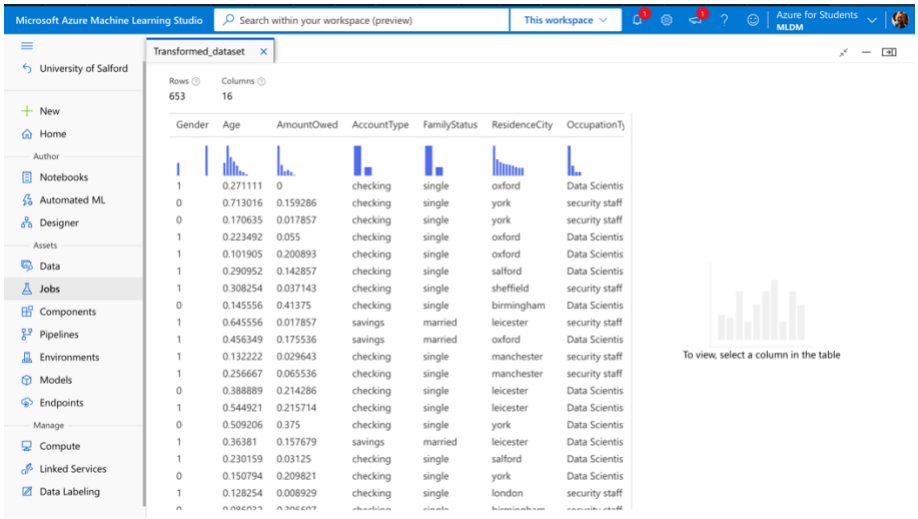
We first created and set the parameters for a workshop, then we went ahead to create a pipeline. We then uploaded the data using the Data feature under assets. The data was loaded into the workshop in a no-header format. Then we used the Edit Meta data feature to add names to the columns. We used the same names we used for the Logistic Regression and Neural Networks modelling. The image below shows the output after we submitted and ran the pipeline.



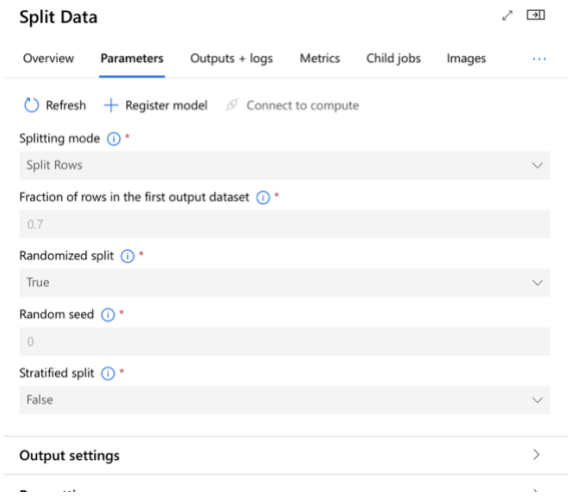
For treating the missing values, we used the Clean Missing data feature. For this we eliminated the observations (rows) that contained missing values. We did this because the missing values were only 5% of the data and we knew it wouldn't affect our model results and accuracy. After this we were left with 653 observations from 690 observations.



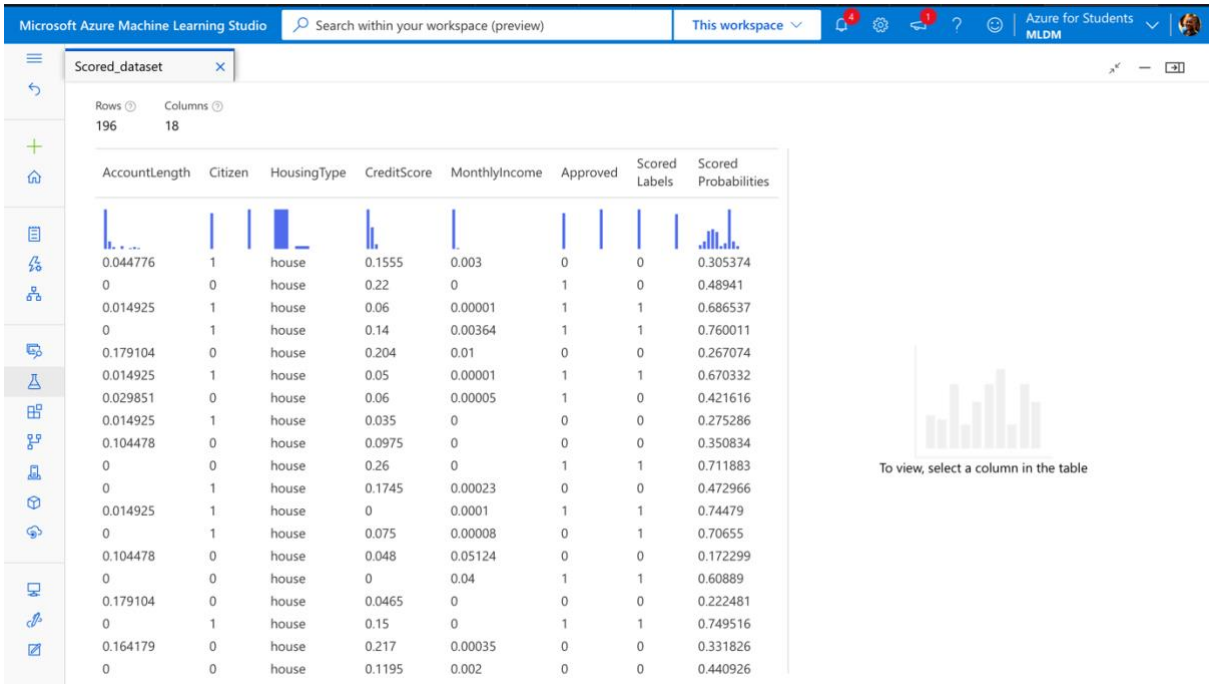
We then normalised the data using the minmax transformation method. We did this because as earlier started, if the numerical values aren't on the same interval, it could cause issues in building our model.



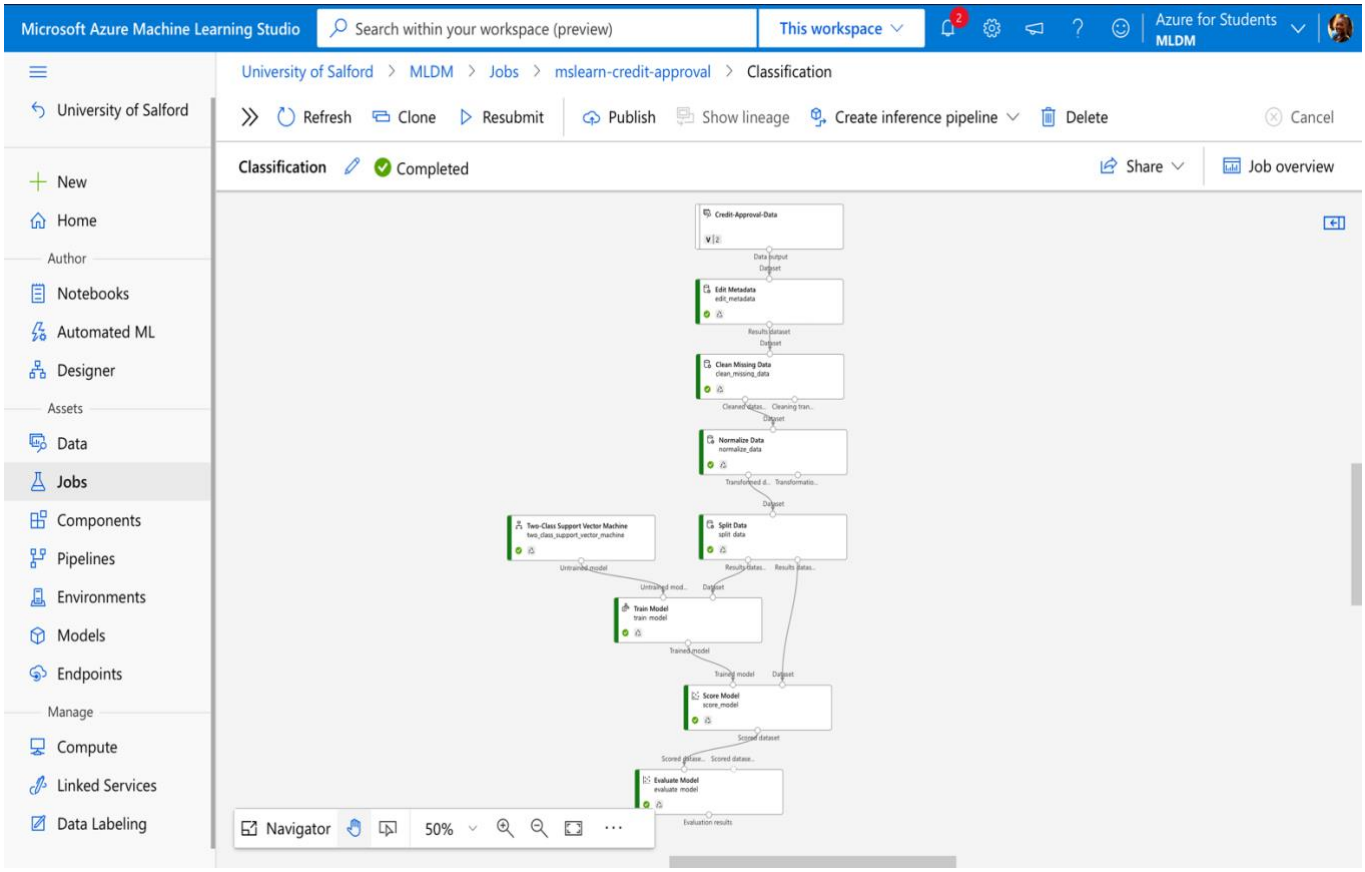
Next, we split our data into two sets. The training set and the test set. We used the 70% - 30% proportion. Find the parameters used for this in the image below.



For our classification, we used the Two Class Support Vector Machine (SVM). SVM is a supervised machine learning algorithm used in classification problems. It works by finding a hyperplane that divides the data set linearly into two classes (Chirag Goyal, 2021). We chose this because of its high-performance metrics and because it is a good linear model for a dataset with less than 100 variables which is our case. From the image below, we see the scored labels column (the model using the test set) and the scored probability column with values between 0 and 1. If the probability is above 0.5, the predicted label is 0 which means approved in our case. The opposite is the case when it is <0.5.



Find below, the pipeline built for the classification of credit card applications.



Results analysis and discussion

We used the confusion matrix, accuracy, and precision score to measure the performance of the models. We used these three instead of just using accuracy which is a popular accuracy metric because using accuracy when we have an unequal number of observations in each class can be a bit misleading. With a confusion matrix, we can see the types of errors that our model is making and where we can find these errors. With precision, we know when our models predict positive correctly. We also used the AUC score for the SVM model.

Confusion matrix is a tabular visualisation of the actual class versus the predicted class. The rows represent the actual class, and the columns represent the predicted class.

	PREDICTED CLASS	
TRUE CLASS	TP	FN
	FP	TN

True Positive (TP) – Correctly classified as positive class.

True Negative (TN) – Correctly classified as negative class.

False Positive (FP) – Wrongly classified as positive class.

False Negative (FN) – Wrongly classified as negative class.

Accuracy is the number of correct predictions divided by the total number prediction.

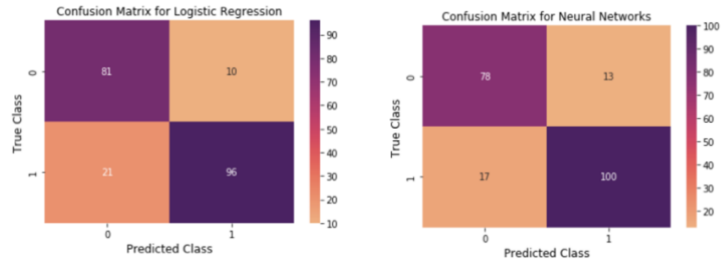
$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

Precision gives the value of correct positives divided by all the predicted positives.

$$Precision = \frac{TP}{TP + FP}$$

AUC - Area under the ROC Curve tells the capability of the model to distinguish between both class.

<u>Logistic Regression and Neural Networks</u>



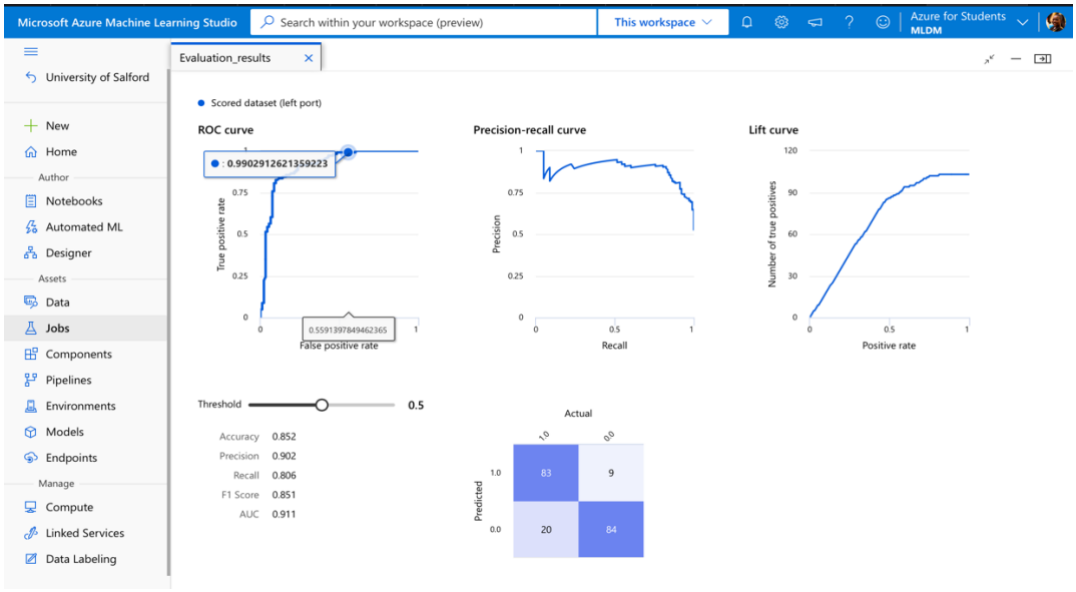
We drew a table containing the values for both models to see and compare both results.

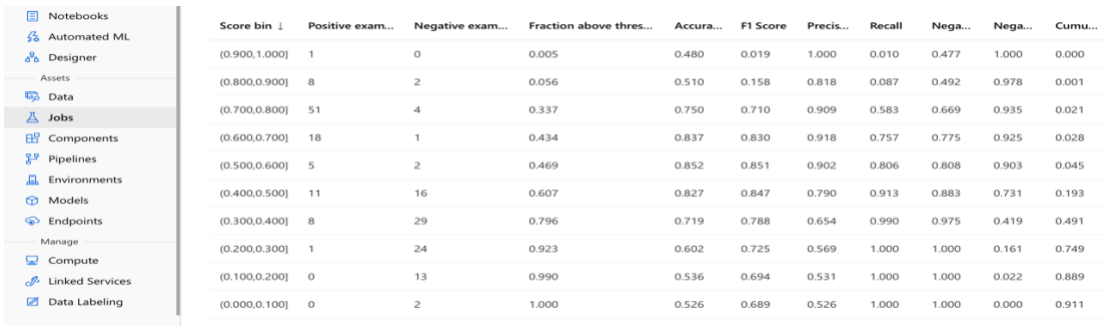
MACHINE LEARNING MODEL	METRIC	SCORE
Logistic Regression	Accuracy	85%
	Precision	79%
Neural Network	Accuracy	86%
	Precision	82%

From the table drawn and from the confusion matrices, both models derived for the methods are good but Neural Network trained the model better than Logistic Regression. The difference is minute, but it exists.

<u>Support Vector Machine in Azure Machine Learning Designer</u>

We can evaluate the model based on the metrics seen in the image below. With an accuracy score of 85%, precision score of 90%, the confusion matrix with a good number of true values and an AUC score of 91%, we can say that our model is a good model for the classification of credit card approvals. It is to be noted that 0, in the confusion matrix speaks to an application being approved.

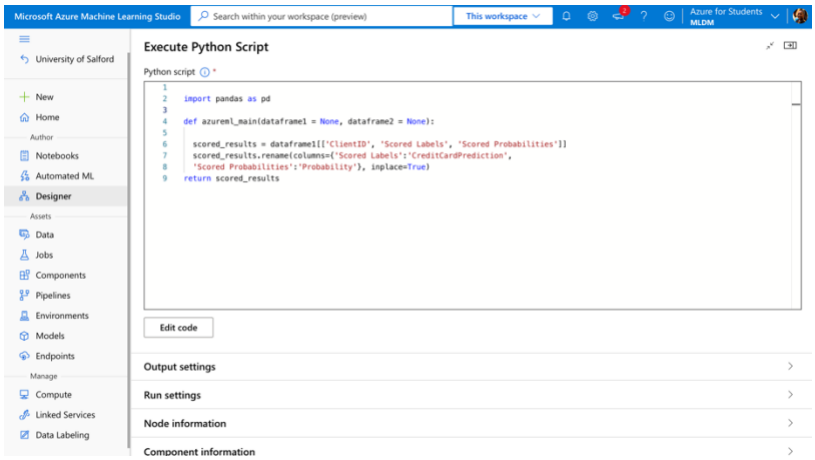


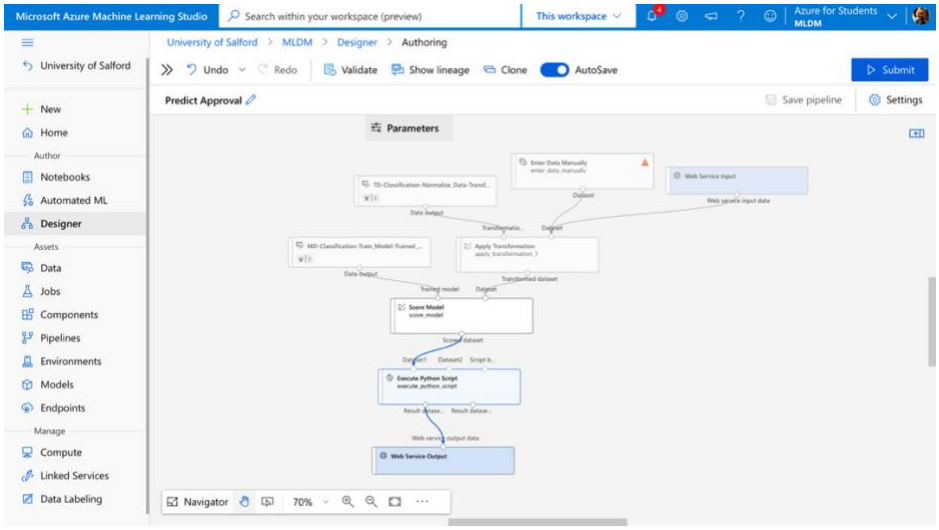


Score bin ↓	Positive exam...	Negative exam...	Fraction above thres...	Accura...	F1 Score	Precis...	Recall	Nega...	Nega...	Cumu...
(0.900,1.000]	1	0	0.005	0.480	0.019	1.000	0.010	0.477	1.000	0.000
(0.800,0.900]	8	2	0.056	0.510	0.158	0.818	0.087	0.492	0.978	0.001
(0.700,0.800]	51	4	0.337	0.750	0.710	0.909	0.583	0.669	0.935	0.021
(0.600,0.700]	18	1	0.434	0.837	0.830	0.918	0.757	0.775	0.925	0.028
(0.500,0.600]	5	2	0.469	0.852	0.851	0.902	0.806	0.808	0.903	0.045
(0.400,0.500]	11	16	0.607	0.827	0.847	0.790	0.913	0.883	0.731	0.193
(0.300,0.400]	8	29	0.796	0.719	0.788	0.654	0.990	0.975	0.419	0.491
(0.200,0.300]	1	24	0.923	0.602	0.725	0.569	1.000	1.000	0.161	0.749
(0.100,0.200]	0	13	0.990	0.536	0.694	0.531	1.000	1.000	0.022	0.889
(0.000,0.100]	0	2	1.000	0.526	0.689	0.526	1.000	1.000	0.000	0.911

We also created a Real-Time Inference pipeline to be published as a service to be used for card applications. We removed the data we used for modelling and inserted a 'Enter

data manually' feature so the dataset for future applicants be entered. We also created the output to have the ClientID, Scored Labels (0 - approved or 1 – not approved) and the Scored Probabilities. After this, we deployed the real-time endpoint so it can be used. The images showing this pipeline is attached below.





General Note

In this modelling process, we ensured that the confidentiality of the clients and data was maintained. Fair and concise judgement was ensured to create unbiased models. The models and metrics were clearly explained to present a transparent analysis.

Conclusions

From the above study, we can conclude that machine learning algorithms are very efficient in the classification of credit card approvals. Without jeopardising the confidentiality of an applicant, it is very easy to predict if the application would be successful or not. Also, we see how the three models performed well with very close accuracy scores.