RMIT International University Vietnam
Assignment Cover Page

Subject Name:	Machine Learning
Subject Code:	COSC2753
Title of Assignment:	Individual Assessment 1
Student Name:	Vo Minh Thien An
Student ID:	s3916570
Lecturer Name:	Bao Nguyen
Word Count:	1346

I. Exploratory Data Analysis (EDA)

1. Briefly Describe

After importing the training dataset to the project, I named it data_train then started the cleaning and analysing process.
By printing out the dataset we can see how many rows and columns it has. I used the .info() and .isna() function to fully see the type of data and if it has a missing value or not. Luckily we don't have a missing value on this dataset.
After this step we know that the dataset's value is integer, made of 25 columns and 202943 rows.

2. Find Outlier

We can know that if the value is in integer it will have limitations, we can find it in the code_book. Most of the data is boolean and ordinal, which means that we can easily visualise it by using bar charts, this type of data is also easy to find outliers because the min max limit is obvious. By using min and max function we can have more insight of the data and figure out outlier (if any)
After this step we see that ExtraMedTest and ExtraAlcoholTest attribute break the limit, these two have values that go from -199 to 199 but the limit is only -100 to 100. One more attribute that might have outliers is the BMI column, the max value is up to 98 which is really high, since I'm not the doctor but in any research said that BMI usually lower than 70 I will consider it as outlier. I will double check it by using boxplot we can see that the BMI value higher than 40 is somewhat considered as outlier, but since we know these value can reach 70, I will set the limit at 70

3. Solve Outlier

All three attributes have the same problem which is value higher or lower than the limit, so I will use the .loc[] function to set it back. Another way I can think of is drop it all, but since the amount of outlier is too big (around 60.000) I think it would be better if we just change it, not to remove
By using .loc[] we can make sure that the data is well changed and have no problem.
Instead of putting the value to mean for ExtraMedTest and ExtraAlcoholTest, I decided to divide it into two groups: higher (more than 100) and lower (less

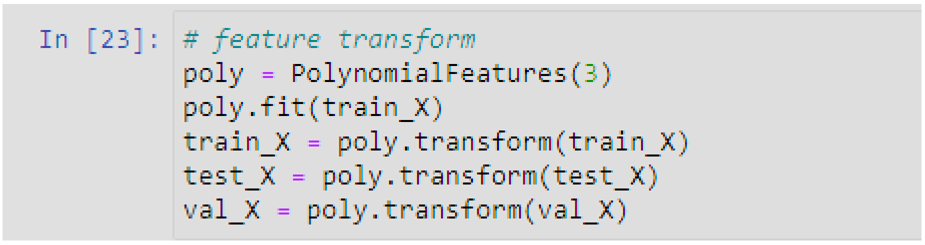
than -100). I will give -100 to the lower group and 100 to the higher group. Due to two reasons. Maximise the model's and practical experience. To optimise the model, I strongly believe that either higher or lower data has a greater possibility of impacting the overall learning step. If I simply assign them back to mean, I would miss out on a lot of data that should be in the proper position. Another reason is that I have tested out two ways of assign the data and this way prints out a better result for both testing and validation.

4. **Setting up Evaluation Framework**

   First we need to figure out which attributes to use which not, we can see in this situation the only column that we will not use is "Id" so we will use drop function to drop it. Another attribute we must drop is "Status" because it is a dependent attribute.

   Use train_test_split to split the dataset into 4 parts, with a test ratio of 20% and set shuffle to "True" so that it can be different each run. The train_test_split is called one more time to split the "train" dataset into 4 parts, with a validation ratio of 25%. We need a validation dataset so that we can check the accuracy score, learning model will familiar with the train and test set and might be overfitting, so that it will reduce accuracy with unfamiliar set.

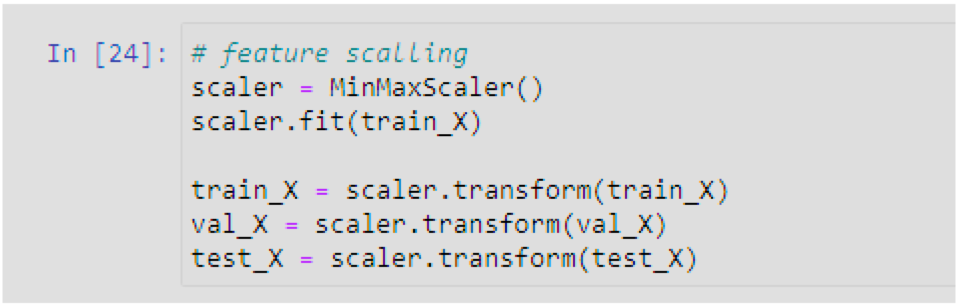
   We then use polynomial feature transformation to create more data in a non-linear way so that the model can capture more complex relationships in the data.



```
In [23]: # feature transform
         poly = PolynomialFeatures(3)
         poly.fit(train_X)
         train_X = poly.transform(train_X)
         test_X = poly.transform(test_X)
         val_X = poly.transform(val_X)
```

<Apply PolynomialFeatures>

After that we do feature scaling by using minmaxscaler, which ensures that all features have the same range of value, which can improve the performance and stability of many machine learning models.



```
In [24]: # feature scalling
         scaler = MinMaxScaler()
         scaler.fit(train_X)

         train_X = scaler.transform(train_X)
         val_X = scaler.transform(val_X)
         test_X = scaler.transform(test_X)
```

<Apply MinMaxScaler>

II. Select and Implement Model

1. Selecting model

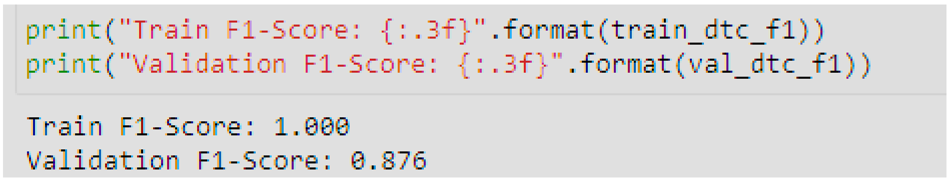
First requirement is to use the model we learned from week 2 to week 5.Second is the model should fit with the problem requirement which is classification.

So I came up with three models: Decision Tree, Random Forest, Logistic Regression.

2. Decision tree

The reason for choosing the Decision Tree model is that even though it is not specialised for this type of problem, it can perform classification work well.

First we run the model by its default to have a better understanding about it, then we calculate the F1-score for both train set and validate set.



```
print("Train F1-Score: {:.3f}".format(train_dtc_f1))
print("Validation F1-Score: {:.3f}".format(val_dtc_f1))

Train F1-Score: 1.000
Validation F1-Score: 0.876
```

<F1-Score>

We can see that the F1-Score for the train set is 100%, but the validation score is only 87%. This shows the problem of overfitting to the set. This problem is one of the disadvantages of Decision Tree, and we need to do the hyper tuning work to solve it.

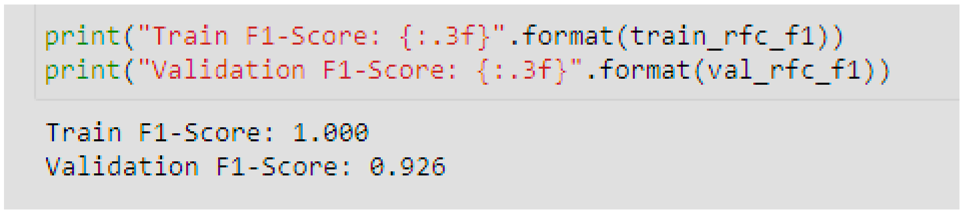
In this project I'm using GridSearch to do the hyper tuning. After getting the best parameters we use it to re-run the model and do the final analysing.

We use F1-Score, Accuracy score and heatmap to calculate the accuracy score for this model, both show that the accuracy is around 94% which is really good.

3. **Random Forest**

Random Forest is an "updated" version of Decision Tree, so it has all the benefits of Decision Tree and even more, which is the reason I chose it.

The other step is similar to the Decision Tree. First we run the model by its default to have a better understanding about it, then we calculate the F1-Score for both train set and validate set.



```
print("Train F1-Score: {:.3f}".format(train_rfc_f1))
print("Validation F1-Score: {:.3f}".format(val_rfc_f1))

Train F1-Score: 1.000
Validation F1-Score: 0.926
```

<F1-Score>

We can see that the F1-Score for the train set this time is still 100%, but the validation score is up to 92%. Which is not bad already, but it is still overfitting and since we can use the hyper tuning to improve it, we will move to that process once again.

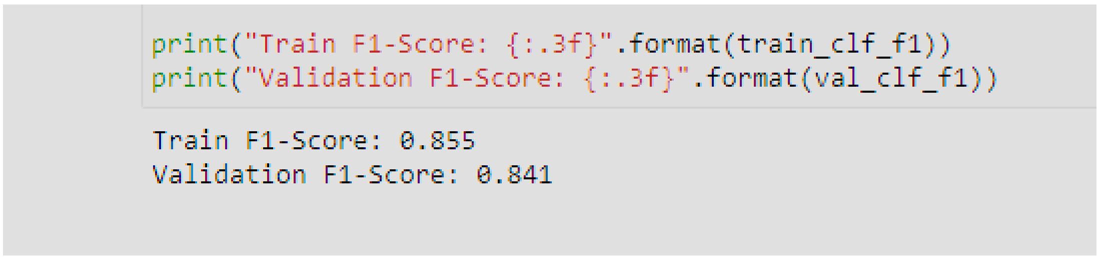
I'm using GridSearch to do the hyper tuning. After getting the best parameters we use it to re-run the model and do the final analysing.

In the final step, we use F1-Score and heatmap to evaluate the model. This time both the F1-Score and heatmap are 96% accurate, which is very good.

4. **Logistic Regression**

Logistic Regression's main purpose is to do the classification work, so it is "in theory" the best fit model for this problem.

First step is to run the model for the first time and check those F1 training and validation score



```
print("Train F1-Score: {:.3f}".format(train_clf_f1))
print("Validation F1-Score: {:.3f}".format(val_clf_f1))

Train F1-Score: 0.855
Validation F1-Score: 0.841
```

<F1-Score>

The result looks not bad, but the validation score and training score are really close to each other. I personally think we can’t do much for this model so I will skip the hyper tuning work and go directly to the final analysing.

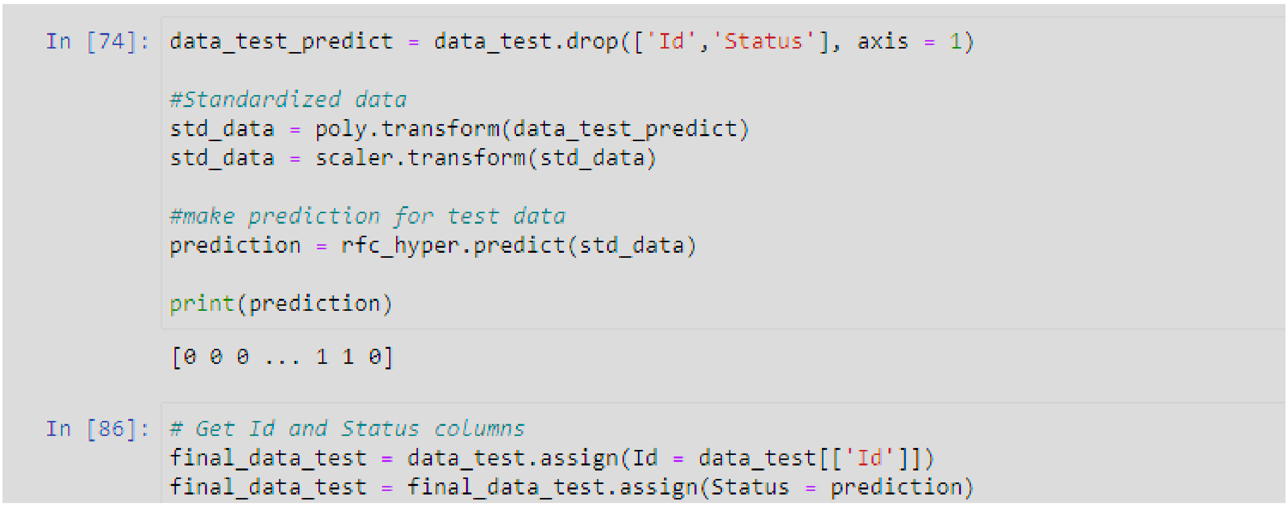
To my surprise both F1-Score and heatmap calculation is really good, but it not close to each other since F1-Score is 90% and heatmap is 96.4%

III. Final Judgement

After running three models we can see that Random Forest currently has the highest performance, both in training, testing and validation. I decided to use it for the final prediction.

Assume that the test data is splitted from the same source so that it will have the same problems as train data which has 3 outliers and no missing value. I will treat it the same way I did with train data.

Finally do the standardise data and put it to prediction



```
In [74]: data_test_predict = data_test.drop(['Id','Status'], axis = 1)

         #Standardized data
         std_data = poly.transform(data_test_predict)
         std_data = scaler.transform(std_data)

         #make prediction for test data
         prediction = rfc_hyper.predict(std_data)

         print(prediction)

         [0 0 0 ... 1 1 0]

In [86]: # Get Id and Status columns
         final_data_test = data_test.assign(Id = data_test[['Id']])
         final_data_test = final_data_test.assign(Status = prediction)
```

<Do the prediction>

Then remove all unnecessary columns and write it to csv file