DEFAULT PAYMENT CLASSIFICATION

Done by: Goh Rui Zhuo (2222329)

OBJECTIVES

Problem Statement

1. Predict which customer will have default payment next month
2. 1 is default while 0 is not default

Dataset includes:

1. 1600 Rows x 13 Columns
2. 8 numerical columns, 4 categorical columns and 1 target variable

General Info

1. No anomaly dtype in the dataset
2. However, anomaly was observed where bill amount less than $0
3. Dataset is clean with no missing values and therefore additional exploratory data analysis can be done

EXPLORATORY DATA ANALYSIS (PART I)

Categorical Data

1. Slightly higher proportion of female as compared to male at 41.2% as compared to 58.7%.
   a. In addition, the countplot shows that female has over 800 rows while male contains more than 600 rows
2. Second plot: Higher proportion of university datas, at 37.7%
   a. From the countplot, University data has over 700 rows in total
3. Third graph: Higher proportion of people are single at 56.2% and married at 43.8%.
   a. From the countplot, we can see that single contains over 800 rows
4. Fourth graph: Imbalance of class where the output 0 occurs more than 78.8% of the time
   a. From the countplot, this amounts to more than 1200 of class 0

Numerical Data

1. Female has a higher percentage of 59.4% with male at 40.6%
   a. Lower amount of total amount of people with default payment 1
2. University students has the highest percentage of people having classify as default payment 1 but this could be because university has the highest respondents in the dataset
3. Single has a higher percentage of people having default payment 1 similarly for default payment 0 but this could be due to higher respondents

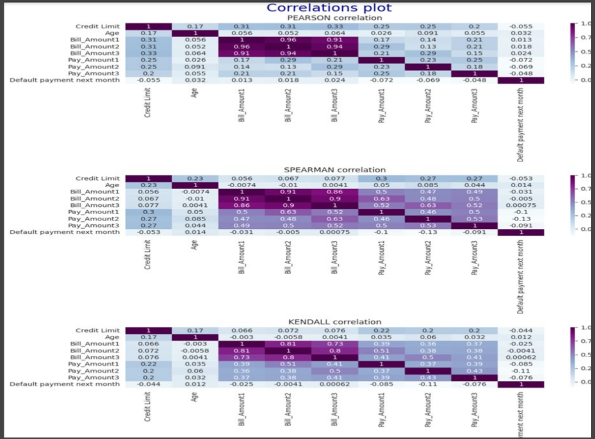
EXPLORATORY DATA ANALYSIS (PART II)

Analysis on Numerical Data by Target

1. Credit Limit: Those have default payment of 0 have a larger range and interquartile range as compared to those who have a default payment of 1
   a. From the histogram, it is positively skewed, with majority between the 10000 to 20000 range
2. Age,: Those who have default payment of 0 have a smaller range and interquartile range as compared to those who have a default payment of 1
   a. From the histogram, Age is positively skewed with majority less than the age of 40
3. Bill Amounts: Those who have default payment of 0 have a wider range and interquartile range as compared to those who have a default payment of 1. Both contains numerous outliers
   a. From the histograms, Bill Amount 1 to 3 is positively skewed with majority less than 10000
   b. Also, there are negative values in bill amounts
4. Pay Amounts: Those who have default payment of 0 have a wider range and interquartile range as compared to those who have a default payment of 1. Both contains numerous outliers
   a. From the histograms, Pay Amount 1 to 3 is positively skewed with majority less than around 3000

Correlation Analysis

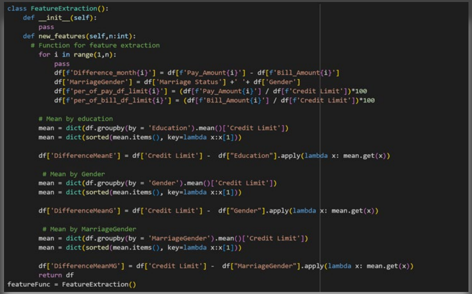
1. From the different types correlation plot, we observed that there is a strong relationship between bill amounts and pay amounts for different months
   a. However, as bill amount 1 2 and 3 are variables that are dependent with each other, multicollinearity is not found



FEATURE ENGINEERING

Feature Extraction

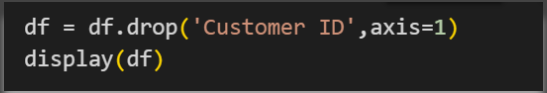
1. Difference between payment and bill
2. Percentage of payment with respect to credit limit
3. Percentage of bill with respect to credit limit
4. Marriage + Gender
5. Difference between bill amount with respect to mean by gender
6. Difference between bill amount with respect to mean by Age



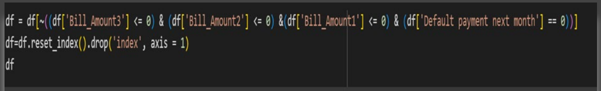
```python
class FeatureExtraction():
    def __init__(self):
        pass
    def new_features(self,n:int):
      # Function for feature extraction
        for i in range(1,n):
            pass
            df[f'Difference_month{i}'] = df[f'Pay_Amount{i}'] - df[f'Bill_Amount{i}']
            df['MarriageGender'] = df['Marriage Status'] +' '+ df['Gender']
            df[f'per_of_pay_df_limit{i}'] = (df[f'Pay_Amount{i}'] / df[f'Credit Limit'])*100
            df[f'per_of_bill_df_limit{i}'] = (df[f'Bill_Amount{i}'] / df[f'Credit Limit'])*100

        # Mean by education
        mean = dict(df.groupby(by = 'Education').mean()['Credit Limit'])
        mean = dict(sorted(mean.items(), key=lambda x:x[1]))

        df['DifferenceMeanE'] = df['Credit Limit'] -  df["Education"].apply(lambda x: mean.get(x))

         # Mean by Gender
        mean = dict(df.groupby(by = 'Gender').mean()['Credit Limit'])
        mean = dict(sorted(mean.items(), key=lambda x:x[1]))

        df['DifferenceMeanG'] = df['Credit Limit'] -  df["Gender"].apply(lambda x: mean.get(x))

         # Mean by MarriageGender
        mean = dict(df.groupby(by = 'MarriageGender').mean()['Credit Limit'])
        mean = dict(sorted(mean.items(), key=lambda x:x[1]))

        df['DifferenceMeanMG'] = df['Credit Limit'] -  df["MarriageGender"].apply(lambda x: mean.get(x))
        return df
featureFunc = FeatureExtraction()
```

Feature and Row Selection

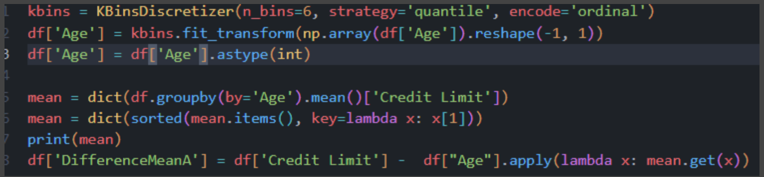
1. Drop customer ID due to no meaning of column
2. Drop columns that have bill amounts 1,2,3 less than 0 and default payment 0 due to it having no exact definition
3. For advanced model, Discretization was done using KBinsDeiscrezation



```python
df = df.drop('Customer ID',axis=1)
display(df)
```



```python
df = df[~((df['Bill_Amount3'] <= 0) & (df['Bill_Amount2'] <= 0) &(df['Bill_Amount1'] <= 0) & (df['Default payment next month'] == 0))]
df=df.reset_index().drop('index', axis = 1)
df
```

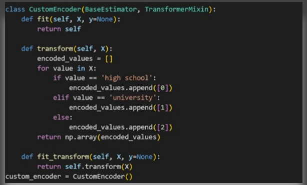


```python
kbins = KBinsDiscretizer(n_bins=6, strategy='quantile', encode='ordinal')
df['Age'] = kbins.fit_transform(np.array(df['Age']).reshape(-1, 1))
df['Age'] = df['Age'].astype(int)

mean = dict(df.groupby(by='Age').mean()['Credit Limit'])
mean = dict(sorted(mean.items(), key=lambda x: x[1]))
print(mean)
df['DifferenceMeanA'] = df['Credit Limit'] -  df["Age"].apply(lambda x: mean.get(x))
```

DATA PREPROCESSING

Preparing and Encoding

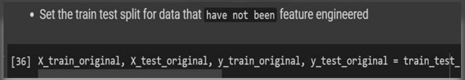
1. Original df was prepared for comparison in later parts
2. Data that are categorical was encoded
3. Marriage status , Gender was done with ordinal encoding
4. Marriage Gender was done with one hot encoding
5. Education was done with custom encoding as on average different status have different state of incomes resulting in different influence



```python
class CustomEncoder(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        encoded_values = []
        for value in X:
            if value == 'high school':
                encoded_values.append([0])
            elif value == 'university':
                encoded_values.append([1])
            else:
                encoded_values.append([2])
        return np.array(encoded_values)

    def fit_transform(self, X, y=None):
        return self.transform(X)
custom_encoder = CustomEncoder()
```

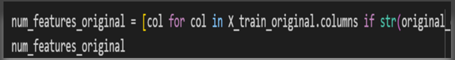
Train test Split

1. Train test split was done with a 20% test set
2. Set the numerical features of the original and feature engineered dataset



- Set the train test split for data that `have not been` feature engineered

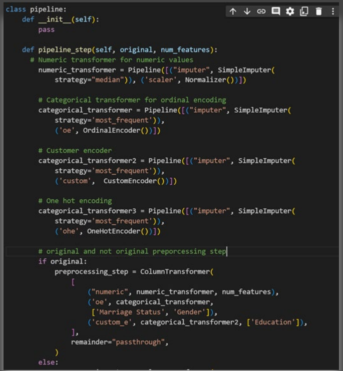
```
[36] X_train_original, X_test_original, y_train_original, y_test_original = train_test_
```



```python
num_features_original = [col for col in X_train_original.columns if str(original_
num_features_original
```

Scaling

1. Normaliser was done to scale the dataset
2. Setting the values to between 0 and 1



```python
class pipeline:
    def __init__(self):
        pass

    def pipeline_step(self, original, num_features):
      # Numeric transformer for numeric values
        numeric_transformer = Pipeline([("imputer", SimpleImputer(
            strategy="median")), ('scaler', Normalizer())])

        # Categorical transformer for ordinal encoding
        categorical_transformer = Pipeline([("imputer", SimpleImputer(
            strategy='most_frequent')),
            ('oe', OrdinalEncoder())])

        # Customer encoder
        categorical_transformer2 = Pipeline([("imputer", SimpleImputer(
            strategy='most_frequent')),
            ('custom',  CustomEncoder())])

        # One hot encoding
        categorical_transformer3 = Pipeline([("imputer", SimpleImputer(
            strategy='most_frequent')),
            ('ohe', OneHotEncoder())])

        # original and not original preporcessing step
        if original:
            preprocessing_step = ColumnTransformer(
                [
                    ("numeric", numeric_transformer, num_features),
                    ('oe', categorical_transformer,
                     ['Marriage Status', 'Gender']),
                    ('custom_e', categorical_transformer2, ['Education']),
                ],
                remainder="passthrough",
            )
        else:
```

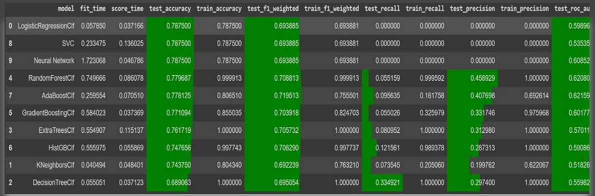
BASELINE MODEL

Model Selected

1. LogisticRegressionClf
2. KNeighborsClf
3. DecisionTreeClf
4. ExtraTreesClf
5. RandomForestClf
6. GradientBoostingClf
7. HistGBClf
8. AdaBoostClf
9. SVC
10. Neural Network

Original Dataframe

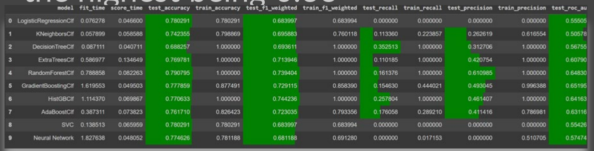
1. All the models did not experience any overfitting
2. Logistic regression, SVC, Neural Network has the tied highest test accuracy at 0.7875
3. Test recall for all models are less than 0.5 with the highest being 0.33



	model	fit_time	score_time	test_accuracy	train_accuracy	test_f1_weighted	train_f1_weighted	test_recall	train_recall	test_precision	train_precision	test_roc_au
0	LogisticRegressionClf	0.057850	0.037166	0.787500	0.787500	0.693885	0.693881	0.000000	0.000000	0.000000	0.000000	0.59896
8	SVC	0.233475	0.136025	0.787500	0.787500	0.693885	0.693881	0.000000	0.000000	0.000000	0.000000	0.53535
9	Neural Network	1.723068	0.046786	0.787500	0.787500	0.693885	0.693881	0.000000	0.000000	0.000000	0.000000	0.60852
4	RandomForestClf	0.749666	0.086078	0.779687	0.999913	0.708813	0.999913	0.055159	0.999592	0.458929	1.000000	0.62080
7	AdaBoostClf	0.259554	0.070510	0.778125	0.806510	0.719513	0.755501	0.095635	0.161758	0.407698	0.692614	0.62159
5	GradientBoostingClf	0.584023	0.037369	0.771094	0.855035	0.703918	0.824703	0.055026	0.325979	0.331746	0.975968	0.60177
3	ExtraTreesClf	0.554907	0.115137	0.761719	1.000000	0.705732	1.000000	0.080952	1.000000	0.312980	1.000000	0.57011
6	HistGBClf	0.555975	0.055869	0.747656	0.997743	0.706290	0.997737	0.121561	0.989378	0.287313	1.000000	0.59086
1	KNeighborsClf	0.040494	0.048401	0.743750	0.804340	0.692239	0.763210	0.073545	0.205060	0.199762	0.622067	0.51826
2	DecisionTreeClf	0.055051	0.037123	0.689063	1.000000	0.695054	1.000000	0.334921	1.000000	0.297400	1.000000	0.55982

Feature Engineered Dataframe

1. All the models did not experience any overfitting
2. Logistic regression, SVC, Neural Network has the tied highest test accuracy at 0.7875
3. Test recall for all models are less than 0.5 with the highest being 0.33



	model	fit_time	score_time	test_accuracy	train_accuracy	test_f1_weighted	train_f1_weighted	test_recall	train_recall	test_precision	train_precision	test_roc_au
0	LogisticRegressionClf	0.076278	0.046600	0.780291	0.780291	0.683997	0.683994	0.000000	0.000000	0.000000	0.000000	0.55505
1	KNeighborsClf	0.057899	0.058588	0.742355	0.798869	0.695883	0.760118	0.113360	0.223857	0.262619	0.616554	0.50578
2	DecisionTreeClf	0.087111	0.040711	0.688257	1.000000	0.693611	1.000000	0.352513	1.000000	0.312706	1.000000	0.56755
3	ExtraTreesClf	0.586977	0.134649	0.769781	1.000000	0.713946	1.000000	0.110185	1.000000	0.420754	1.000000	0.60790
4	RandomForestClf	0.788858	0.082263	0.790795	1.000000	0.739404	1.000000	0.161376	1.000000	0.610985	1.000000	0.64830
5	GradientBoostingClf	1.619553	0.049503	0.777859	0.877491	0.729115	0.858390	0.154630	0.444021	0.493045	0.996388	0.65195
6	HistGBClf	1.114370	0.069867	0.770633	1.000000	0.744236	1.000000	0.257804	1.000000	0.461407	1.000000	0.64163
7	AdaBoostClf	0.387311	0.073823	0.761710	0.826423	0.723035	0.793356	0.176058	0.289210	0.411416	0.786981	0.63116
8	SVC	0.138513	0.065959	0.780291	0.780291	0.683997	0.683994	0.000000	0.000000	0.000000	0.000000	0.55426
9	Neural Network	1.827638	0.048052	0.774626	0.781188	0.681188	0.691280	0.000000	0.017153	0.000000	0.510705	0.57474

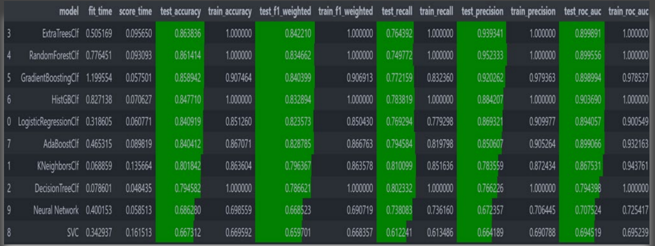
Observations

1. Test Accuracy: About half of the model increase while the rest remain the same
2. Test recall increase for majority of the model
3. From the EDA portion, Age., Credit limit are example of column that can be used for discretization
4. Test size was also reduced to 0.15
5. Kbins discretization was also selected
6. Additional feature selection was done with selectkbest
7. In addition, class imbalance was also observed (SMOTEN was used)

ADVANCED MODEL AND HYPERPARAMETER TUNING

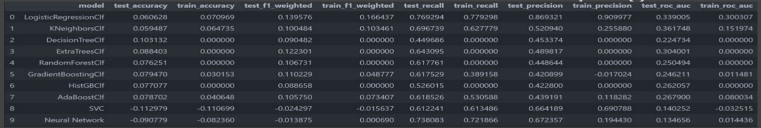
Advanced Model

1. All model did not show signs of overfitting
2. Extra Tree has the highest test accuracy at 0.863
3. All models scored above 0.8 except for neural network and SVC
4. Test Recall also improved where all models are above 0.7 except for SVC
5. Across the board, all models improved



	model	fit_time	score_time	test_accuracy	train_accuracy	test_f1_weighted	train_f1_weighted	test_recall	train_recall	test_precision	train_precision	test_roc_auc	train_roc_auc
3	ExtraTreesClf	0.505169	0.095650	0.863836	1.000000	0.842210	1.000000	0.764392	1.000000	0.939341	1.000000	0.899891	1.000000
4	RandomForestClf	0.776451	0.093093	0.861414	1.000000	0.834662	1.000000	0.749772	1.000000	0.952333	1.000000	0.899556	1.000000
5	GradientBoostingClf	1.199554	0.057501	0.858942	0.907464	0.840399	0.906913	0.772159	0.832360	0.920262	0.979363	0.898994	0.978537
6	HistGBClf	0.827138	0.070627	0.847710	1.000000	0.832894	1.000000	0.783819	1.000000	0.884207	1.000000	0.903690	1.000000
0	LogisticRegressionClf	0.318605	0.060771	0.840919	0.851260	0.823573	0.850430	0.769294	0.779298	0.869321	0.909977	0.894057	0.900549
7	AdaBoostClf	0.465315	0.089819	0.840412	0.867071	0.828785	0.866763	0.794584	0.819798	0.850607	0.905264	0.899066	0.932163
1	KNeighborsClf	0.068859	0.135664	0.801842	0.863604	0.796367	0.863578	0.810099	0.851636	0.783559	0.872434	0.867531	0.943761
2	DecisionTreeClf	0.078601	0.048435	0.794582	1.000000	0.786621	1.000000	0.802332	1.000000	0.766226	1.000000	0.794398	1.000000
9	Neural Network	0.400153	0.058513	0.686280	0.698559	0.668523	0.690719	0.738083	0.736160	0.672357	0.706445	0.707524	0.725417
8	SVC	0.342937	0.161513	0.667312	0.669592	0.659701	0.668357	0.612241	0.613486	0.664189	0.690788	0.694519	0.695239

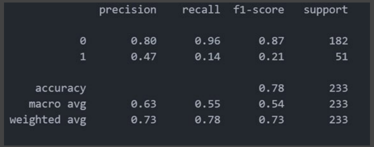
1. All test accuracy experience an increase after resampling and bins discretization
2. This is also seen in recall
3. Across other evaluation metrics, all scores improved which shows the usefulness of resampling



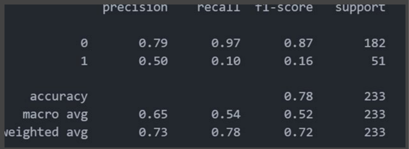
	model	test_accuracy	train_accuracy	test_f1_weighted	train_f1_weighted	test_recall	train_recall	test_precision	train_precision	test_roc_auc	train_roc_auc
0	LogisticRegressionClf	0.060628	0.070969	0.139576	0.166437	0.769294	0.779298	0.869321	0.909977	0.339005	0.300307
1	KNeighborsClf	0.059487	0.064735	0.100484	0.103461	0.696739	0.627779	0.520940	0.255880	0.361748	0.151974
2	DecisionTreeClf	0.103132	0.000000	0.090482	0.000000	0.449686	0.000000	0.453374	0.000000	0.224734	0.000000
3	ExtraTreesClf	0.088403	0.000000	0.122301	0.000000	0.643095	0.000000	0.489817	0.000000	0.304001	0.000000
4	RandomForestClf	0.076251	0.000000	0.106731	0.000000	0.617761	0.000000	0.448644	0.000000	0.250494	0.000000
5	GradientBoostingClf	0.079470	0.030153	0.110229	0.048777	0.617529	0.389158	0.420899	-0.017024	0.246211	0.011481
6	HistGBClf	0.077077	0.000000	0.088658	0.000000	0.526015	0.000000	0.422800	0.000000	0.262057	0.000000
7	AdaBoostClf	0.078702	0.040648	0.105750	0.073407	0.618526	0.530588	0.439191	0.118282	0.267900	0.080034
8	SVC	-0.112979	-0.110699	-0.024297	-0.015637	0.612241	0.613486	0.664189	0.690788	0.140252	-0.032515
9	Neural Network	-0.090779	-0.082360	-0.013875	0.000690	0.738083	0.721866	0.672357	0.194430	0.134656	0.014436

Hyperparameter tuning

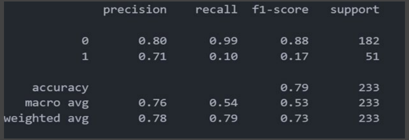
1. Top 3 models of Extra tree, Random Forest and Gradient Boosting was chosen to hyperparameter tune
2. Random Forest, Extra Tree and Gradient Boosting was then hyperparameter tune with Randomised Search CV
3. **Observations**
   a. No overfitting seen
   b. Accuracy score improved
   c. Compare to a dummy model, we can see that All models improved in all areas
   d. Classification report accuracy scores improved



```
              precision    recall  f1-score   support

           0       0.80      0.96      0.87       182
           1       0.47      0.14      0.21        51

    accuracy                           0.78       233
   macro avg       0.63      0.55      0.54       233
weighted avg       0.73      0.78      0.73       233
```



```
              precision    recall  f1-score   support

           0       0.79      0.97      0.87       182
           1       0.50      0.10      0.16        51

    accuracy                           0.78       233
   macro avg       0.65      0.54      0.52       233
eighted avg       0.73      0.78      0.72       233
```

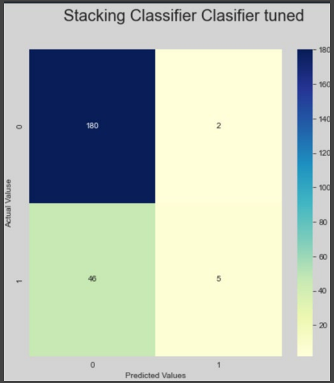


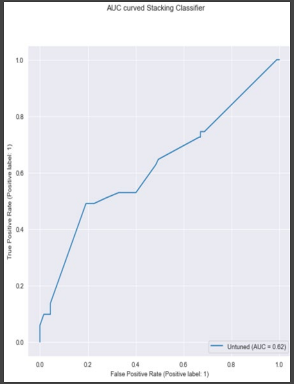
```
              precision    recall  f1-score   support

           0       0.80      0.99      0.88       182
           1       0.71      0.10      0.17        51

    accuracy                           0.79       233
   macro avg       0.76      0.54      0.53       233
weighted avg       0.78      0.79      0.73       233
```

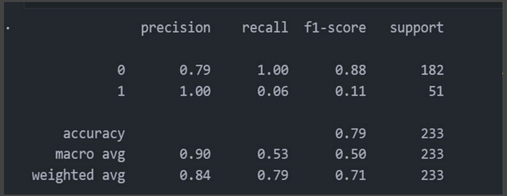
STACKING CLASSIFIER AND FINAL MODEL

Stacking Classifier

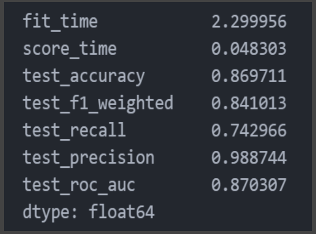
1. Stacking Classifier indeed improve the scores of the model
2. Majority of the scores increased slightly
3. However auc is lower compared to extra tree and Random Forest alone with a huge decrease







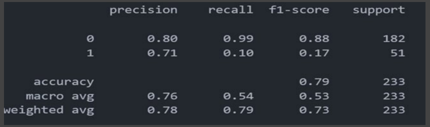
```
              precision    recall  f1-score   support

           0       0.79      1.00      0.88       182
           1       1.00      0.06      0.11        51

    accuracy                           0.79       233
   macro avg       0.90      0.53      0.50       233
weighted avg       0.84      0.79      0.71       233
```



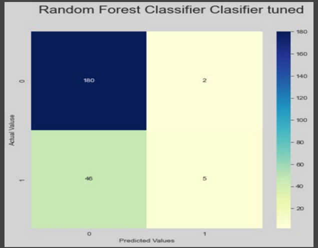
```
fit_time            2.299956
score_time          0.048303
test_accuracy       0.869711
test_f1_weighted    0.841013
test_recall         0.742966
test_precision      0.988744
test_roc_auc        0.870307
dtype: float64
```

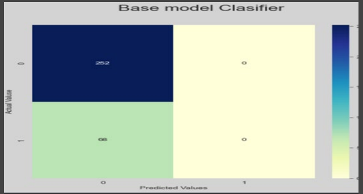
Final Model

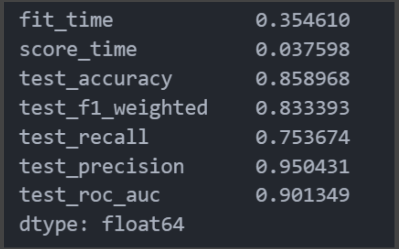
1. Random Forest Classifier was chosen
2. Test accuracy of 0.86 with test recall of 0.76
3. Classification report of precision is 0.78 and recall is 0.79s
4. False negative was reduced by some margin and test accuracy increase with other evaluation metrics also increased through the process



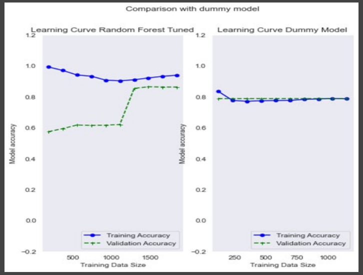
```
              precision    recall  f1-score   support

           0       0.80      0.99      0.88       182
           1       0.71      0.10      0.17        51

    accuracy                           0.79       233
   macro avg       0.76      0.54      0.53       233
weighted avg       0.78      0.79      0.73       233
```







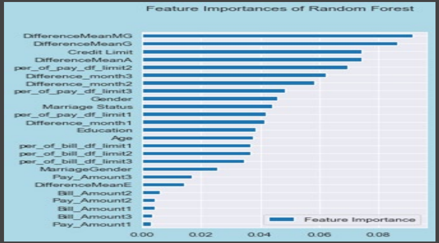
```
fit_time            0.354610
score_time          0.037598
test_accuracy       0.858968
test_f1_weighted    0.833393
test_recall         0.753674
test_precision      0.950431
test_roc_auc        0.901349
dtype: float64
```

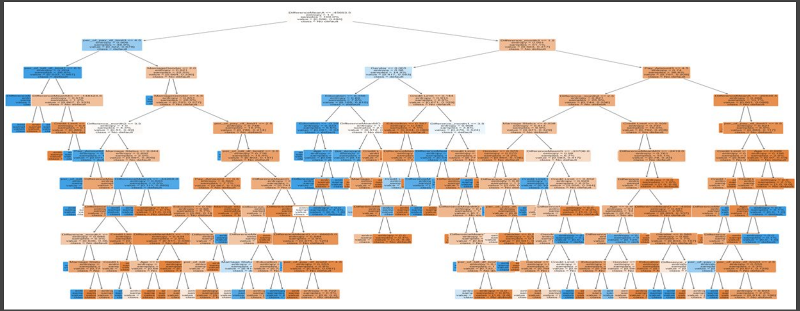


FEATURE IMPORTANCE AND TREE BASED INTERPRETATION

Feature Importance

1. From the above feature importance, we can see that the best feature was actually difference between mean between Marriategender, which could possibly means that if there is a huge difference between the average, then there could be a higher risk of defaulting and that the marriage status and gender is extremely important





Final Conclusion

Overall the model improved as compared to a baseline model, while stacking classifier was great the score produced for auc is not ideal, hence I reverted to Random Forest for a more balanced score across the board