COMP0198 Coursework:

Predicting Credit Card Defaults and Limit Balances

1. Task 1 - Predicting Customer Defaults

This first section of the report focuses on building multiple classification models that predicts whether a customer will default on their next payment or not. Successful predictions by the models could provide credit card issuers with information that could save them a lot of money in the long run. The models used in this task were Logistic Regression, Support Vector Machine, and a Random Forest Model.

1.1 General Analysis

This report utilizes the 2016 “default of credit card clients” dataset from the UC Irvine Machine Learning Repository. It contains 25 columns and 30,000 rows of information, the main features can be split into 4 groups + 1 extra:

X1 - A customer’s amount of given credit could have direct correlation with whether they will default their next payment or not. Higher amounts of given credit could mean the customer has a good credit history, resulting in a hig credit score, potentially leading to fewer defaults.

X2-X5 - These demographic features can provide insights into the customer’s profile, influencing their credit behaviour. While education, marital status and age could affect the possibility of defaulting, if the training data is not representative of the entire population it could create bias and discrimination.

X6-X11 - Past repayment records give a trend of the customer’s payment behaviour. A consistent delay could indicate a higher risk of default.

X12-X17 - The amount here reflects a customer's spending behaviour. A combination of a high number here along with a low payment in X18-X23 could indicate financial stress and an inability to make payments, leading to a default.

X18-X23 - The amount here shows how well the customer has been able to meet their previous payments.

While having a high number of features (23) allows the models to have more data to make predictions from, it could also become more difficult for the models to spot patterns due to the amount of noise and possibly irrelevant features. Feature selection or dimensionality reduction techniques such as PCA should be performed to avoid overfitting, techniques such as correlation analysis can also be employed to visualize and identify highly correlated features and remove redundant ones. While irrelevant for a random forest model, feature scaling should also be use, without it, certain attributes will dominate and cause imbalance.

I believe the dataset provides enough comprehensive, reliable and insightfull information for this task, the features provided should have a strong enough correlation to the possibility of a default. The challenging part of this task would be modelling the data appropriately while avoiding overfitting or underfitting.

First, conduct essential data exploration and data preprocessing required such as data cleaning, correlation analysis, and feature engineering. Then the dataset was split into training, validation, and test sets with a 70/15/15 split. Cross-validation was performed on the training set to receive the first set of results. The models were then evaluated on the validation set and optimised with hyperparameter tuning, the models did not have any interaction with the test set until the very end.

1.2 Data Exploration and Feature Engineering

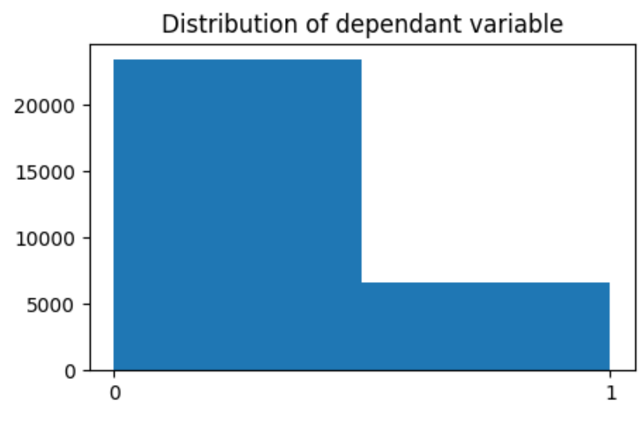
After uploading the dataset into a dataframe, some quick adjustments were made, removing the **ID** column (X0) as it has no relevance to the task, and renaming 2 columns for clarity. No missing data was present.

```
#Load data and skip the X headers
df = pd.read_excel('dataset.csv',skiprows=[0])

#Remove the ID column as it's irrelevant to the model
df.drop(columns=df.columns[0], axis=1,  inplace=True)

#Rename PAY_0 to PAY_1 for consistency and last column to Default
df = df.rename(columns={'PAY_0': 'PAY_1', 'default payment next month': 'DEFAULT'})

#Visualize the dataframe
df.head()

#Check for missing or anomalous data
df.info()
```

1.2.1 Data Distribution

A histogram was created to visualize the distribution of the dependant variable, revealing the highly imbalanced class distribution. Instances of default only accounted for 22.12% of the dataset (6636 instances). This will be addressed later with over/under sampling.

```
plt.figure(figsize=(5,3))
plt.hist(df['DEFAULT'], bins=2)
plt.title('Distribution of dependant variable')
plt.xticks([0,1])
plt.show()

class_names = {0:'No Default', 1:'Default'}
print(df.DEFAULT.value_counts().rename(index = class_names))

print("Percentage of default:  ", (df['DEFAULT'].sum() / len(df['DEFAULT'])*100), "%")
```



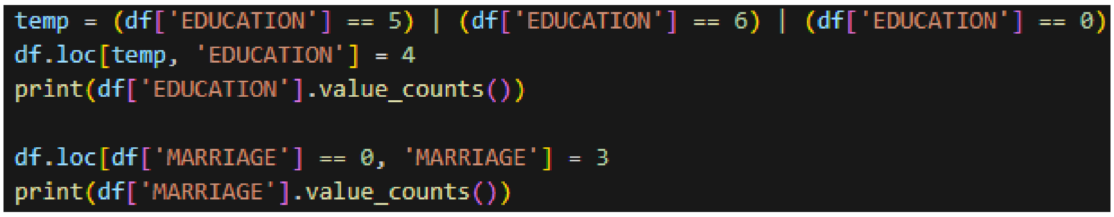
```
No Default    23364
Default        6636
Name: count, dtype: int64
Percentage of default:   22.12 %
```

1.2.2 Outliers/Anomalous Data

Upon further investigation, it seemed that the following features contained anomalous data:

- **EDUCATION** has unlabelled data category 0, 5, 6
- **MARRIAGE** has category 0 unlabelled
- **LIMIT_BAL** has a very broad range compared to other features, this will need to be dealt with later but not considered an outlier that should be removed

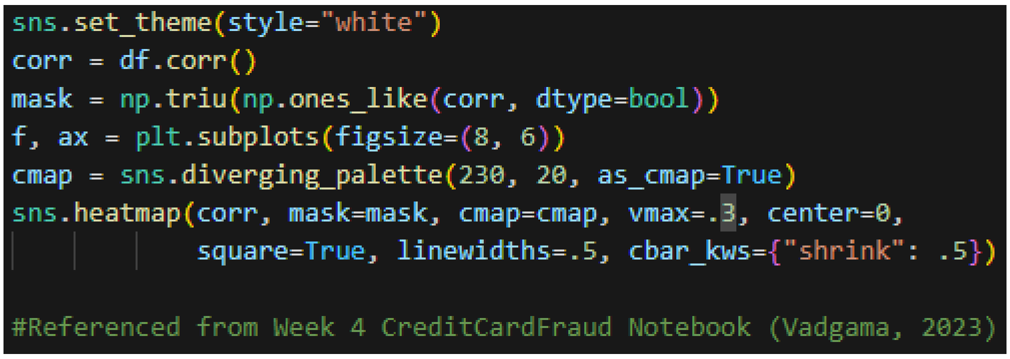
To resolve the first two points, the unlabelled data was categorized as ‘Other’ in their respective columns, 0 in **MARRIAGE** categorized as 3, while 0,5,6 in **EDUCATION** was categorized as 4.



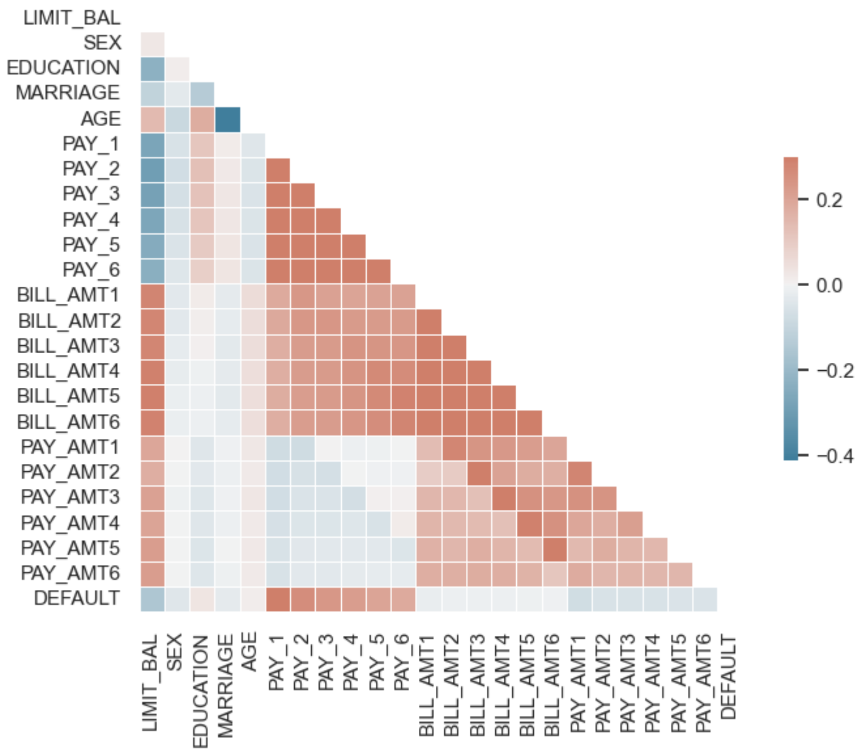
```
temp = (df['EDUCATION'] == 5) | (df['EDUCATION'] == 6) | (df['EDUCATION'] == 0)
df.loc[temp, 'EDUCATION'] = 4
print(df['EDUCATION'].value_counts())

df.loc[df['MARRIAGE'] == 0, 'MARRIAGE'] = 3
print(df['MARRIAGE'].value_counts())
```

1.2.3 Correlations

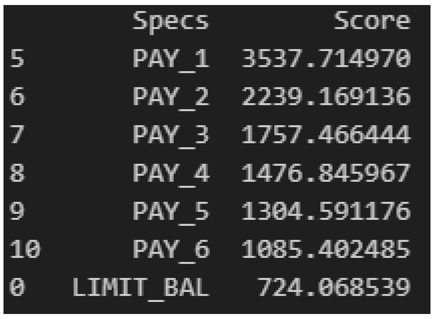
A correlation matrix was were created to visualize any possible correlations between any of the features. While the correlation isn’t strong, repayment status (**PAY_X**) is showing a positive correlation with **DEFAULT**, repayment status also displays a negative correlation with **LIMIT_BAL** which is interesting.



```
sns.set_theme(style="white")
corr = df.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
f, ax = plt.subplots(figsize=(8, 6))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})

#Referenced from Week 4 CreditCardFraud Notebook (Vadgama, 2023)
```



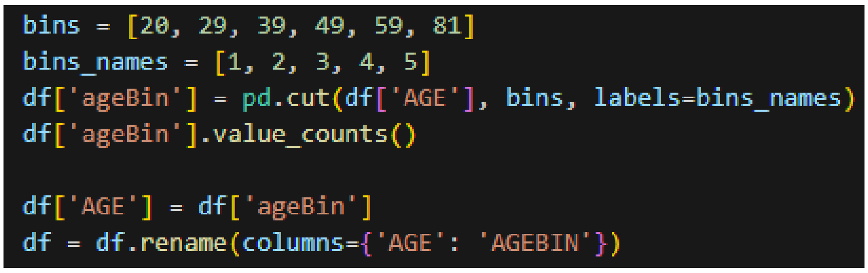
The feature selection function from scikit-learn also confirms those correlations with a score.



```
         Specs        Score
5        PAY_1  3537.714970
6        PAY_2  2239.169136
7        PAY_3  1757.466444
8        PAY_4  1476.845967
9        PAY_5  1304.591176
10       PAY_6  1085.402485
0    LIMIT_BAL   724.068539
```

1.2.4 Feature Engineering

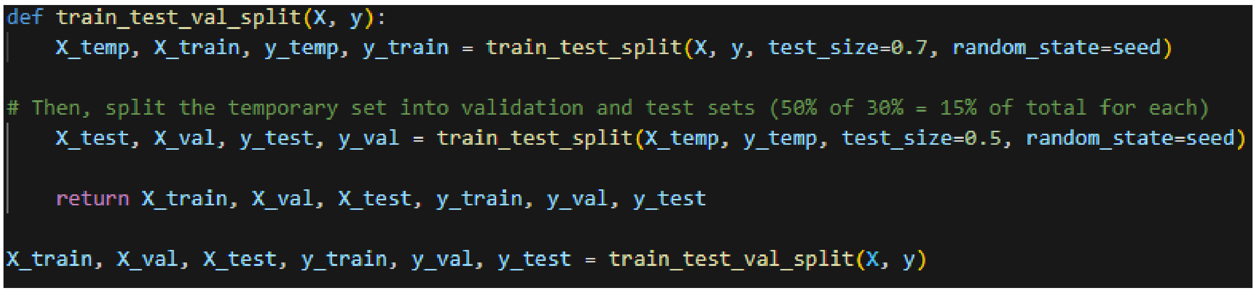
Binning was used to transform **AGE,** a continuous numerical variable into discrete “bins”. This demographic feature can be simplified into 5 categories, reducing the number of unique values and highlighting patterns within specific ranges, replacing the **AGE** column with **AGEBIN**.
The bins are 20-29, 30-39, 40-49, 50-59, 60-81



```
bins = [20, 29, 39, 49, 59, 81]
bins_names = [1, 2, 3, 4, 5]
df['ageBin'] = pd.cut(df['AGE'], bins, labels=bins_names)
df['ageBin'].value_counts()

df['AGE'] = df['ageBin']
df = df.rename(columns={'AGE': 'AGEBIN'})
```

1.2.5 Dataset Splitting

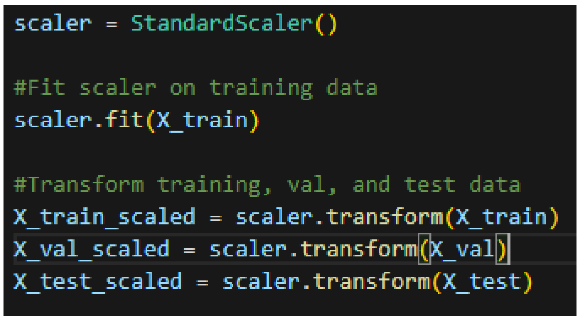
This process is conducted in this section as splitting it now before scaling ensures the models will be evaluated on unseen data. Scaling the data first could lead to information leakage, introducing information from the test set into the training set. By splitting the data into train, val, and test sets it allows the test set to be untouched until the very end, and so the models can be evaluated on how well it generalizes on completely unseen data similar to the real world.



```
def train_test_val_split(X, y):
    X_temp, X_train, y_temp, y_train = train_test_split(X, y, test_size=0.7, random_state=seed)

# Then, split the temporary set into validation and test sets (50% of 30% = 15% of total for each)
    X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=seed)

    return X_train, X_val, X_test, y_train, y_val, y_test

X_train, X_val, X_test, y_train, y_val, y_test = train_test_val_split(X, y)
```

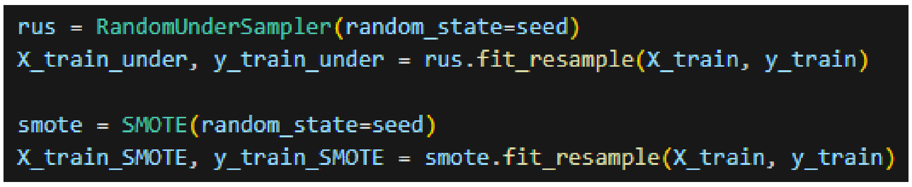
1.2.6 Data Scaling and Balancing

As mentioned in section 1.1, feature scaling is required for this task as models that rely on distance-based measurements such as SVM and Logistic Regression with regularization are being employed. Standardized scaling was chosen due to its robustness to outliers and that it preserves information about the shape of the distribution, with the negative being it does not bound ot a specific range like Normalization.



```
scaler = StandardScaler()

#Fit scaler on training data
scaler.fit(X_train)

#Transform training, val, and test data
X_train_scaled = scaler.transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

Variables for data balancing were also initialized here but will be performed after cross-validation and before hyperparameter tuning. Balancing the data before hyperparamter tuning can lead to faster training times and more stable performance metrics.



```
rus = RandomUnderSampler(random_state=seed)
X_train_under, y_train_under = rus.fit_resample(X_train, y_train)

smote = SMOTE(random_state=seed)
X_train_SMOTE, y_train_SMOTE = smote.fit_resample(X_train, y_train)
```

1.3 Model Training

1.3.1 Performance Measures

Several metrics will be used in tandem to determine the performance of the models, as each of them have their own benefits.

Confusion Matrix

A tool used to evaluate how an algorithm is doing at classifying data. By showing the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

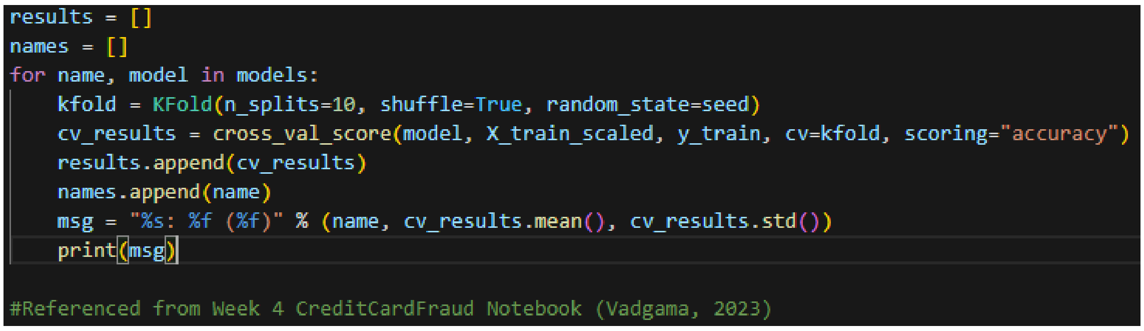
While it's ideal to minimize both FP and FN and keep a high TP and TN, we have to prioritise and optimize an element ahead of others. For this task we will focus on minimizing FN as this element represents the customers we predicted would NOT default, but DID default.

A multitude of metrics can be derived from the confusion matrix but two of them were chosen:
Accuracy - The proportion of correctly classified instances out of the total
Recall - The proportion of TP among all actual positive instances. High recall indicates a low FN rate.
Accuracy was picked due to its simple interpretability and its ability to give a general overview of how well a model is performing, recall as that directly correlates with the goal o f a low number of FN.

Another metric that will be looked at is Area Under the Receiver Operating Characteristic Curve (AUC-ROC). It is a metric that quantifies the overall performance of a model across different classification thresholds, it also performs well on imbalanced datasets.

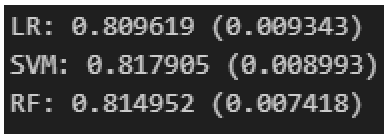
1.3.2 Cross-Validation

K-folds cross-validation is a technique for assessing the performance and generalization ability of a model, this is done purely on the training set. The set is divided into 'K' folds, trained 'K' times, each time using K-1 folds for training and the remaining fold for validation. This process is repeated 'K' times, and the performance metrics are averaged. This was done using the cross_val_score function from scikit-learn and computed 3 times, each time with a different scoring function: "accuracy", "roc_auc", and "recall".

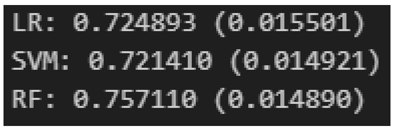


```
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
    cv_results = cross_val_score(model, X_train_scaled, y_train, cv=kfold, scoring="accuracy")
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)

#Referenced from Week 4 CreditCardFraud Notebook (Vadgama, 2023)
```

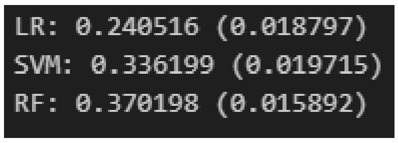
Accuracy	ROC-AUC	Recall



```
LR: 0.809619 (0.009343)
SVM: 0.817905 (0.008993)
RF: 0.814952 (0.007418)
```



```
LR: 0.724893 (0.015501)
SVM: 0.721410 (0.014921)
RF: 0.757110 (0.014890)
```

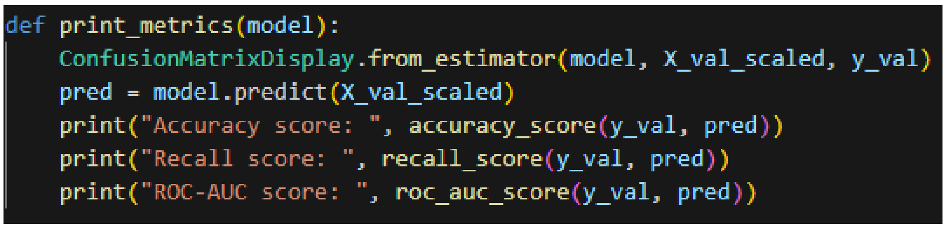


```
LR: 0.240516 (0.018797)
SVM: 0.336199 (0.019715)
RF: 0.370198 (0.015892)
```

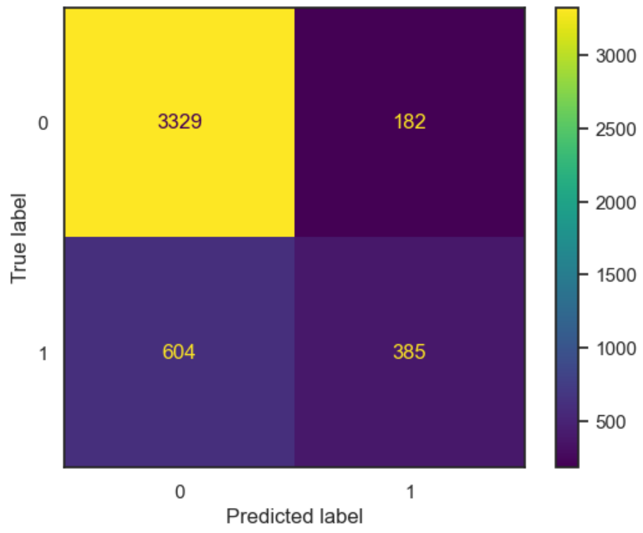
Above are the cross-validation score results, with the first number denoting accuracy and the value in brackets denoting the standard deviation. While the accuracy isn't really high, it is not bad for a base model with cross-validation and 10 folds. With logistic regression performing slightly worse than the support vector machine and random forest model. For ROC_AUC and recall, the random forest model performed the best, not only did it obtain the highest mean score for both metrics, the low variance also suggests a more stable performance across the 10 folds. While the recall score is quite low overall, I expect this to go up after the classes get balanced.

1.3.3 Validation Set Performance

The models were then fitted with the training data before being evaluated against the validation set, then the models were assessed against the performance metrics mentioned in 1.3.1.



```python
def print_metrics(model):
    ConfusionMatrixDisplay.from_estimator(model, X_val_scaled, y_val)
    pred = model.predict(X_val_scaled)
    print("Accuracy score: ", accuracy_score(y_val, pred))
    print("Recall score: ", recall_score(y_val, pred))
    print("ROC-AUC score: ", roc_auc_score(y_val, pred))
```



Above is an example confusion matrix from the Random Forest (RF) model, as mentioned earlier we want to reduce the number of FN as much as possible which would be the bottom left quadrant.
While the RF model did have the lowest number of FN out of the three, it also had a higher FP rate, which might not be desirable.

	Accuracy	ROC-AUC	Recall
LR	0.8164	0.6155	0.2568
SVM	0.8253	0.6564	0.3549
RF	0.8253	0.6687	0.3893

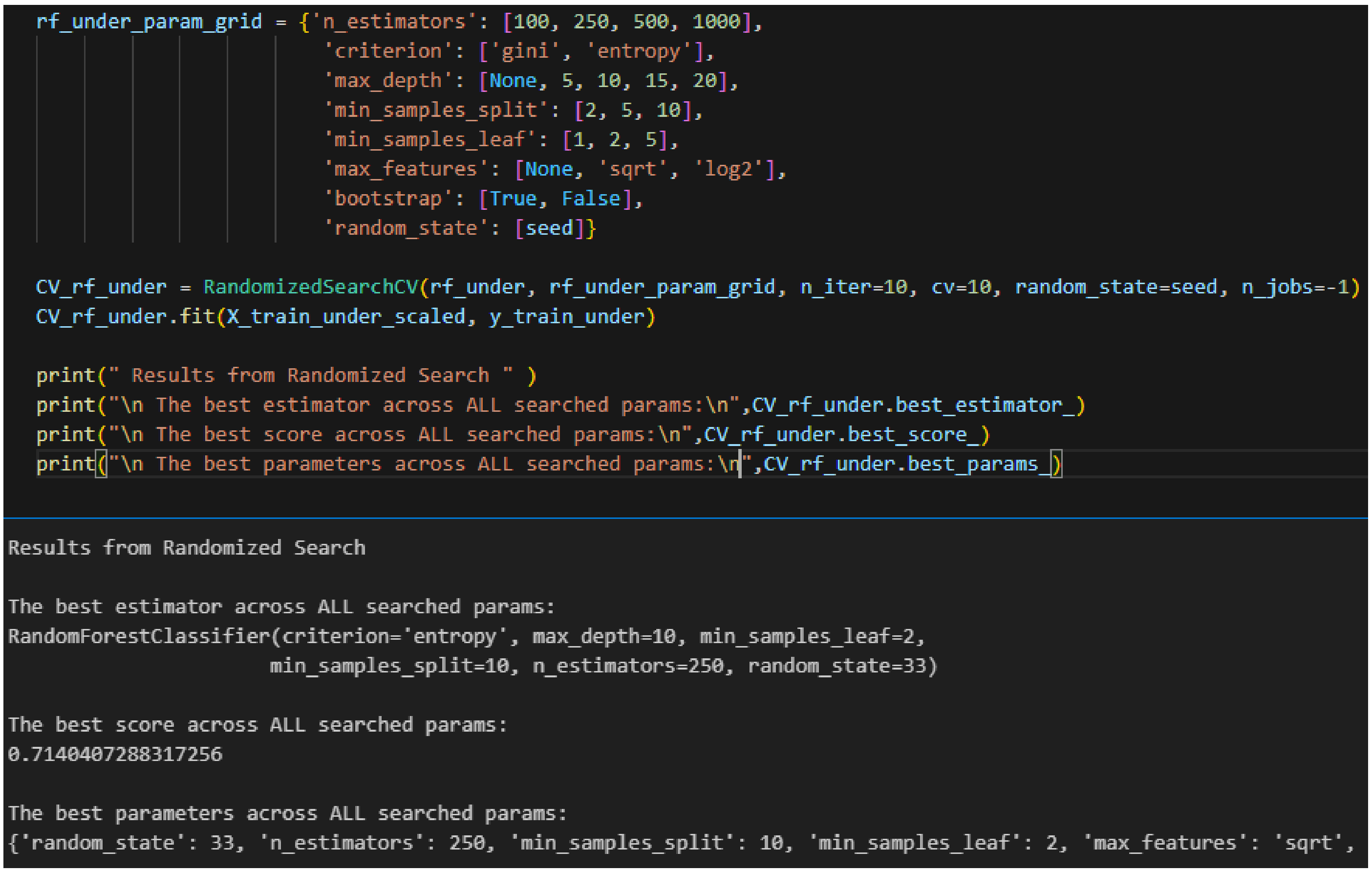
Table: Validation set performance

When compared to the cross-validation results, there's an increase in accuracy and recall across all three models which is a good sign, indicating the models are generalizing well on unseen data so far. However, there's also a sizeable drop in ROC-AUC, which could be due to a multitude of reasons such as imbalanced data or overfitting.

1.3.4 Balancing and Hyperparameter Tuning

Before tuning hyperparameters, under/over sampling was performed on the sets of data, multiple tests were conducted to find the highest which was the best performing option. The same performance criteria was used, prioritising recall score but not completely ignoring accuracy and ROC-AUC. Ultimately, logistic regression was best left as is, when trying to random undersample or SMOTE oversample, the increase in number of FPs along with the drop in accuracy and ROC-AUC was too much to ignore even though recall did increase. For RF and SVM, I decided to go with the random undersample as there was a sizeable increase in performance without sacrificing too much in other areas.

The model's performance was then refined using hyperparameter tuning. With the plethora of hyperparameters available for the models, RandomizedSearchCV from scikit-learn was picked due to its less computationally intensive nature.

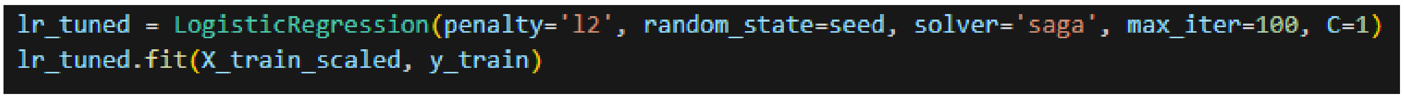


```
rf_under_param_grid = {'n_estimators': [100, 250, 500, 1000],
                       'criterion': ['gini', 'entropy'],
                       'max_depth': [None, 5, 10, 15, 20],
                       'min_samples_split': [2, 5, 10],
                       'min_samples_leaf': [1, 2, 5],
                       'max_features': [None, 'sqrt', 'log2'],
                       'bootstrap': [True, False],
                       'random_state': [seed]}

CV_rf_under = RandomizedSearchCV(rf_under, rf_under_param_grid, n_iter=10, cv=10, random_state=seed, n_jobs=-1)
CV_rf_under.fit(X_train_under_scaled, y_train_under)

print(" Results from Randomized Search " )
print("\n The best estimator across ALL searched params:\n",CV_rf_under.best_estimator_)
print("\n The best score across ALL searched params:\n",CV_rf_under.best_score_)
print("\n The best parameters across ALL searched params:\n",CV_rf_under.best_params_)
```

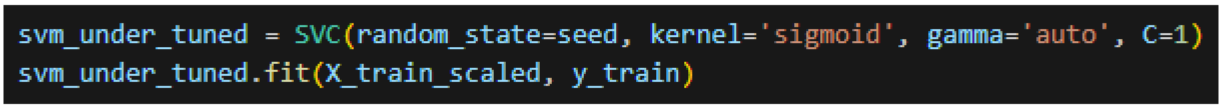
```
Results from Randomized Search

The best estimator across ALL searched params:
RandomForestClassifier(criterion='entropy', max_depth=10, min_samples_leaf=2,
                       min_samples_split=10, n_estimators=250, random_state=33)

The best score across ALL searched params:
0.7140407288317256

The best parameters across ALL searched params:
{'random_state': 33, 'n_estimators': 250, 'min_samples_split': 10, 'min_samples_leaf': 2, 'max_features': 'sqrt',
```

Example of hyperparameter tuning for the RF model

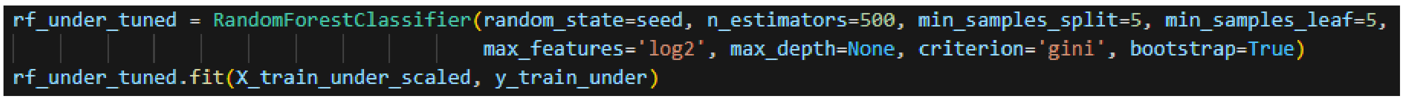
Hyperparameters Chosen:



```
lr_tuned = LogisticRegression(penalty='l2', random_state=seed, solver='saga', max_iter=100, C=1)
lr_tuned.fit(X_train_scaled, y_train)
```



```
svm_under_tuned = SVC(random_state=seed, kernel='sigmoid', gamma='auto', C=1)
svm_under_tuned.fit(X_train_scaled, y_train)
```



```
rf_under_tuned = RandomForestClassifier(random_state=seed, n_estimators=500, min_samples_split=5, min_samples_leaf=5,
                                        max_features='log2', max_depth=None, criterion='gini', bootstrap=True)
rf_under_tuned.fit(X_train_under_scaled, y_train_under)
```

	Accuracy	**ROC-AUC**	**Recall**

LR	0.8071	0.5940	0.2159
SVM	0.4889	0.5094	0.5458
RF	0.7362	0.6966	0.6263

Table: Test set performance

After tuning and fitting the models they were measured against the previously unseen test set. Due to setting the Randomized Search scoring parameter to recall, SVM and RF did return with a significantly better recall score, at the cost of a increase in the number of FPs too. ROC-AUC remained relatively similar with the RF having an increase in score, all this could possibly be attributed to the randomized nature of the search function, resulting in certain metrics performing worse than before.

2. Task 2 - Predicing Customer Limit Balance

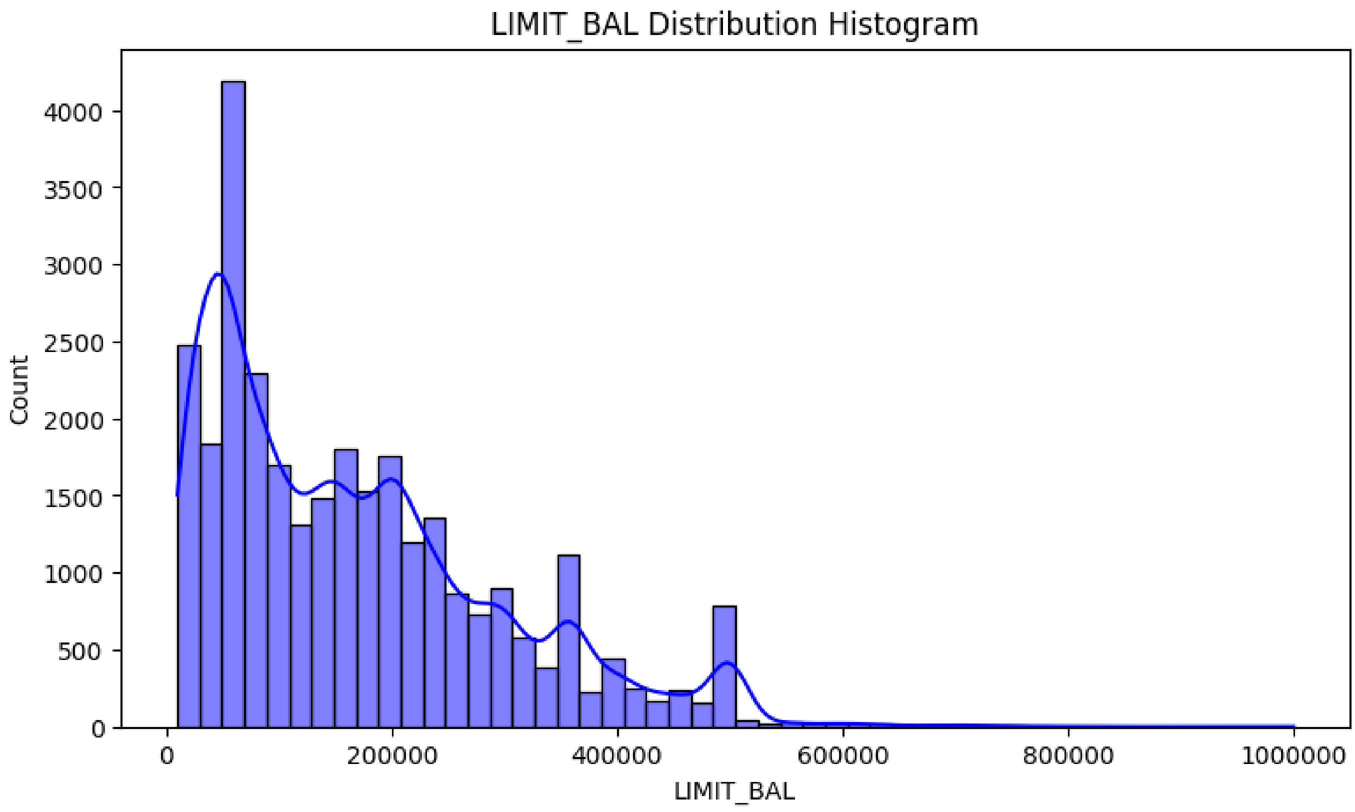
2.1 General Analysis

For this task, the only features available are the demographic features (X2 to X5), relying solely on demographic data to predict the limit balance of customers. The regression task is confronted with challenges due to having only four features (three categorical and one numerical). This limited number of features may result in a model that underfits, struggling to capture the inherent patterns in the data. Moreover, with such a small feature set, the model may encounter difficulties when met with unseen test data.

The limited amount of information could potentially lead to over-generalization of information. To overcome this limitation, augmenting the number of features, particularly by incorporating income levels of customers, can significantly improve the predictive performance of the model. The approach to this task is similar to the previous task, found is section 1.1.

2.2 Data Exploration and Feature Engineering

2.2.1 Data Distribution



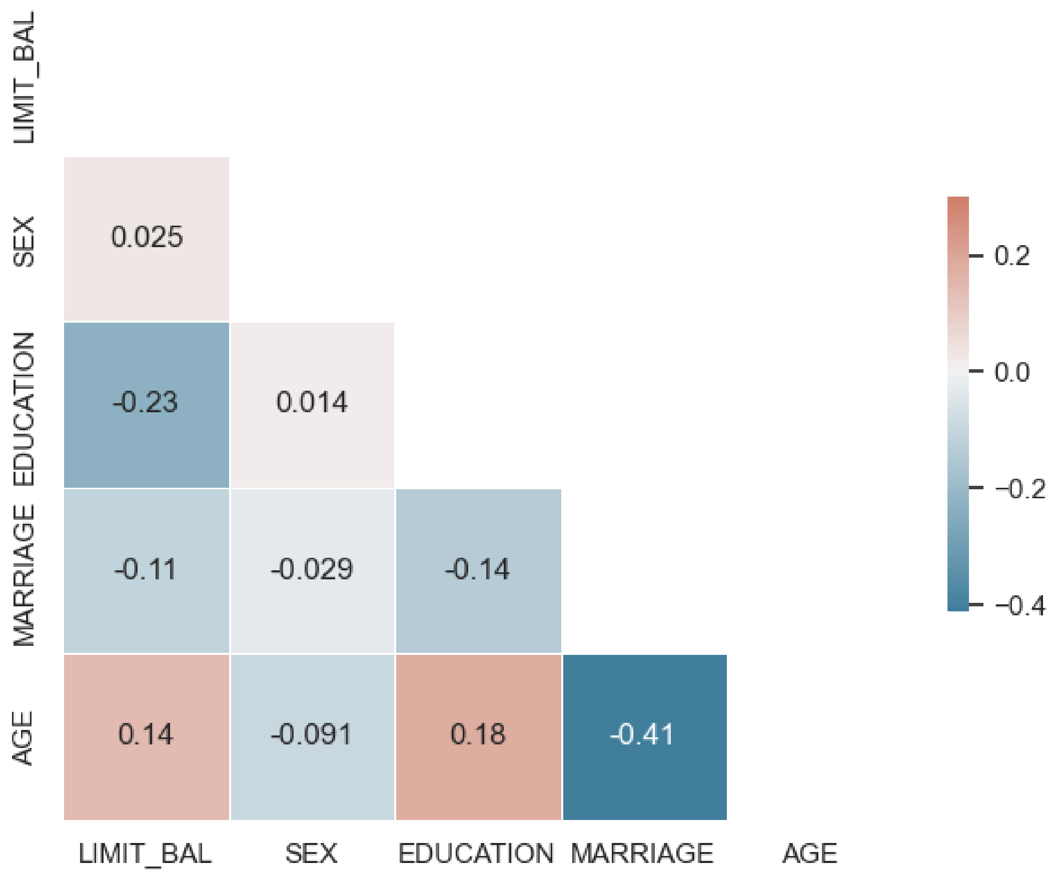
The distribution of limit balance can be visualized through this histogram. We can see that the target variable for this task showcase a strong positive skewness, meaning the average will be greater than the median as the presence of extreme values to the right pulls the average up, indicating a significant portion of data is concentrated on the lower end. Scaling and transformation should be conducted before training models.

2.2.2 Outliers/Anomalous Data

As it’s the same dataset there are no significant outliers, anomalous data has been dealt with the same way in section 1.2.2

2.2.3 Correlation

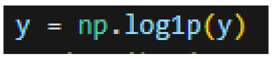
Similar to section 1.2.3, the correlation matrix for the features are displayed below.



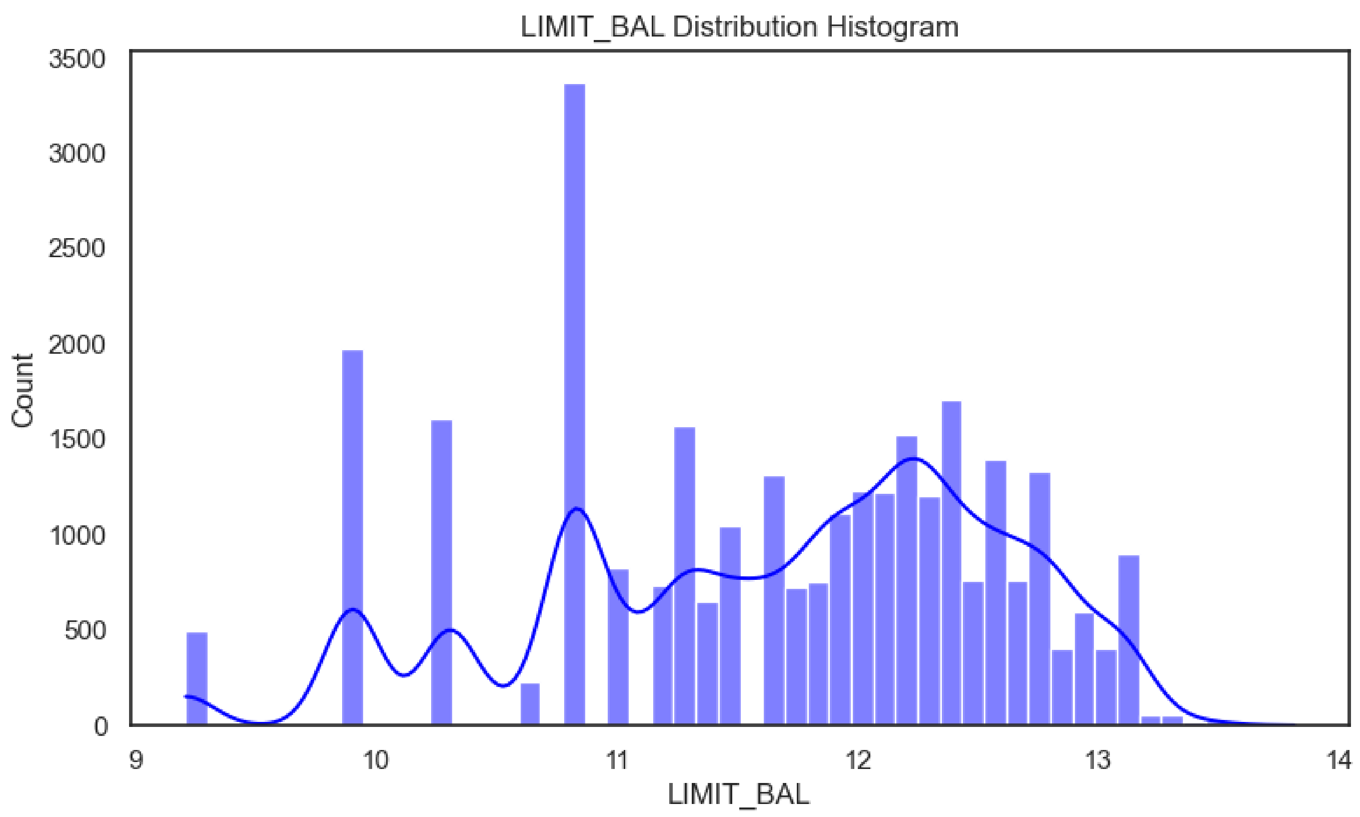
This shows that there are no significant correlations between the columns, with the slight exception of marriage and age.

2.2.4 Transformations

As mentioned in section 2.2.1, the dependent variable is positively skewed. To address the skewness, the dependent variable went through multiple approaches to make the distribution more symmetric with techniques such as square root, box-cox and reciprocal transformation. But ultimately, the best transformation was the logarithmic transformation, not only does it reduce the skewness by a ton, it also brings the level of magnitude to a much more manageable amount, overall easier to interpret.



```
y = np.log1p(y)
```



2.2.5 Dataset Splitting and Scaling

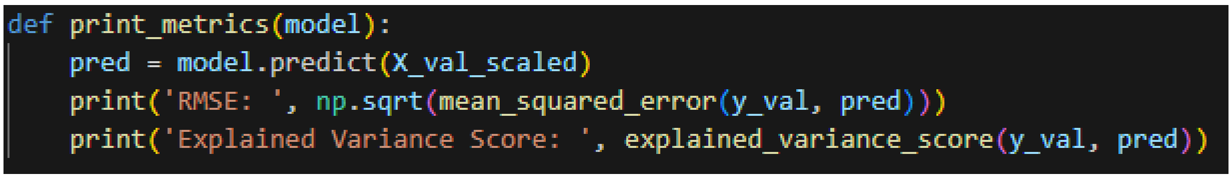
The data was split into training, validation, and test in the same in section 1.2.5, the scaling was also conducted in the same manner. Balancing is not required here as it is a regression task.

2.3 Model Training

2.3.1 Performance Measures

The two performance measures chosen are widely-used metrics Root Mean Square Error (RMSE) and Explained Variance Score (EVS), they provide valuable insight into the quality of regression models. RMSE quantifies the average magnitude of errors between predicted and actual values, offering a measure of the model's precision. On the other hand, EVS assesses the proportion of variance captured by the model, it provides an indication of how well the model captures the variability in the data.

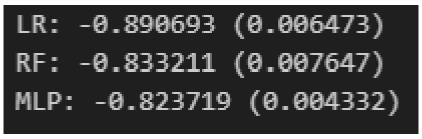
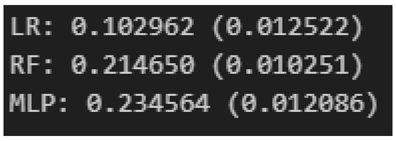
For RMSE, lower is better, while EVS higher is better.



```
def print_metrics(model):
    pred = model.predict(X_val_scaled)
    print('RMSE: ', np.sqrt(mean_squared_error(y_val, pred)))
    print('Explained Variance Score: ', explained_variance_score(y_val, pred))
```

2.3.2 Cross-Validation

The cross-validation methodology was followed closely to section 1.3.2, with the only difference being the scoring method changed to the performance measures explained above.

RMSE	EVS
 LR: -0.890693 (0.006473) RF: -0.833211 (0.007647) MLP: -0.823719 (0.004332)	 LR: 0.102962 (0.012522) RF: 0.214650 (0.010251) MLP: 0.234564 (0.012086)

The RMSE is in negative due to how the scikit-learn function works, it does not mean it’s a terrible score. LR had the highest RMSE score while MLP had ths highest EVS score.

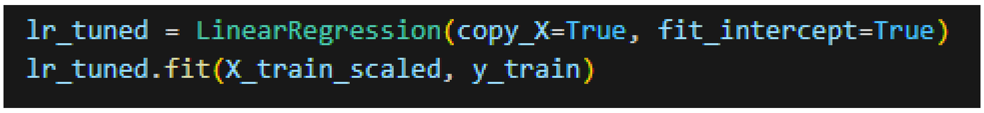
2.3.3 Validation Set Performance

Using code similar to section 1.3.3, the training data was fit onto the models with default parameters, and tested against the validation set. Very similar results comparison with the cross-validation

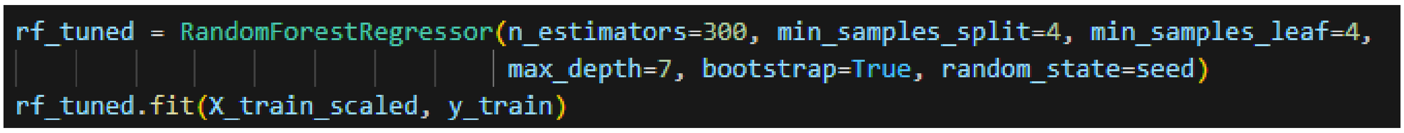
	RMSE	EVS
LR	0.8978	0.1010
RF	0.8359	0.2211
MLP	0.8211	0.2485

2.3.4 Hyperparameter Tuning

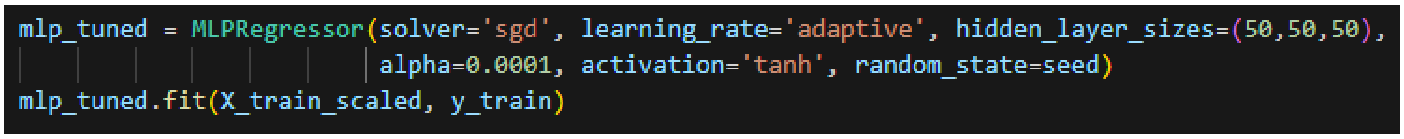
Following a similar method to section 1.3.4, the same Randomized search function was using to find the optimal hyperparameters for the models. Then evaluated against the test set, to see its performance on unseen data.



```
lr_tuned = LinearRegression(copy_X=True, fit_intercept=True)
lr_tuned.fit(X_train_scaled, y_train)
```



```
rf_tuned = RandomForestRegressor(n_estimators=300, min_samples_split=4, min_samples_leaf=4,
                                 max_depth=7, bootstrap=True, random_state=seed)
rf_tuned.fit(X_train_scaled, y_train)
```



```
mlp_tuned = MLPRegressor(solver='sgd', learning_rate='adaptive', hidden_layer_sizes=(50,50,50),
                         alpha=0.0001, activation='tanh', random_state=seed)
mlp_tuned.fit(X_train_scaled, y_train)
```

	RMSE	EVS
LR	0.8861	0.1084
RF	0.8213	0.2340
MLP	0.8213	0.2342

The RMSE scores for LR and RF had a slight increase, while MLP was the same. RF had a slight increase in EVS score while MLP performed worse. However, the changes are not significant enough to confidently say these are the optimal parameters due to reasons similar in section 1.3.4.

2.4 Conclusion

In the above table, the analysis of results reveals that both Random Forest and Multi-Layer Perceptron (MLP) exhibit similar RMSE and EVS scores. Linear Regression is superior in terms of RMSE score which is surprising due to its simple and linear nature. The lower RMSE of MLP and RF suggests more accurate average predictions compared to Linear Regressiont. In regression analysis, the balance between accuracy and interpretability is crucial, especially in scenarios like assigning limit balances to customers. The limited feature set, focusing solely on demographical factors, may benefit from the inclusion of additional features like customer income levels to enhance model performance. Furthermore, exploring the hyperparameter search space comprehensively, as discussed in section 1.3.3, could contribute to further improving the model's predictive capabilities.

Reference list

Basanisi, L. (2018). *Credit Card Default: a very pedagogical notebook*. [online] kaggle.com. Available at: https://www.kaggle.com/code/lucabasa/credit-card-default-a-very-pedagogical-notebook [Accessed 12 Dec. 2023].

Dominguez, M. (2021). *Predicting Credit Card Defaults with Machine Learning*. [online] Medium. Available at: https://medium.com/swlh/predicting-credit-card-defaults-with-machine-learning-fcc8da2fdafb [Accessed 12 Dec. 2023].

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, [online] 12(85), pp.2825–2830. Available at: https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

Yeh,I-Cheng. (2016). default of credit card clients. UCI Machine Learning Repository. https://doi.org/10.24432/C55S3H.