CMP-7023B Data Mining Coursework-2
Data Mining the Lending Club Loan Dataset

Chandupraveen Gudi, Reg. 100329132

Abstract

Credit risk prediction is more important for the financial institutions while approving the loans. Status of the loan can be affected by various reasons. This paper aims to analyse the credit risk prediction with both supervised (KNeighborsClassifier, DecisionTreeClassifier, RandomForestClassifier) and unsupervised (Kmeans clustering) algorithms using a lending club loans dataset with more than 70000 observations and 108 features. The findings shows that RandomForest Classifier outperforms KNN and DecisionTree classifiers. Clustering results are not satisfactory due to imbalances in data. Supervised learning algorithms were implemented by grouped data to mitigate the data imbalances. Various evaluation metrics like accuracy, precisiom, recall, f1-score were used to validate the models. Using hyperparameter tuning, the performance of the model have seen an improvement especially with f1-score.

1 Introduction

Lending Club is an american peer-to-peer loaning company that created a platform to connect borrowers and investors. Borrowers can get loan amounts ranging from \$1,000 and \$40,000. Data regarding the loans provided will be disclosed by the firm at specified time periods (Yash, 2020). The dataset used for analysis contains the loans, that are issued from the first quarter of 2013 till the last quarter of 2017. This paper discusses the various supervised and unsupervised machine learning algorithms for analyzing the lending club dataset.

Section 2 discusses about the broad summary statistics of the features available, Data preprocessing techniques were implemented in Section3. Section4 explains various supervised models and their evaluations, Section6 has the details of unsupervised clustering and finally the conclusion in Section6.

2 Summary of Features

Summarizing the available features is an important task that will help in cleaning the dataset and for finding the relationships between the variables using various visualization techniques.

2.1 Data types

Given dataset contains a total of 108 variables with 20 categorical and 88 are numerical as shown in Figure 1. Out-of 20 categorical variables, 5 are datetime and 15 are strings; The string variables contain 4 ordinal variables and 11 nominal variables. Among 88 numerical variables, 49 are continuous i.e. float and rest 39 are discrete i.e. int.

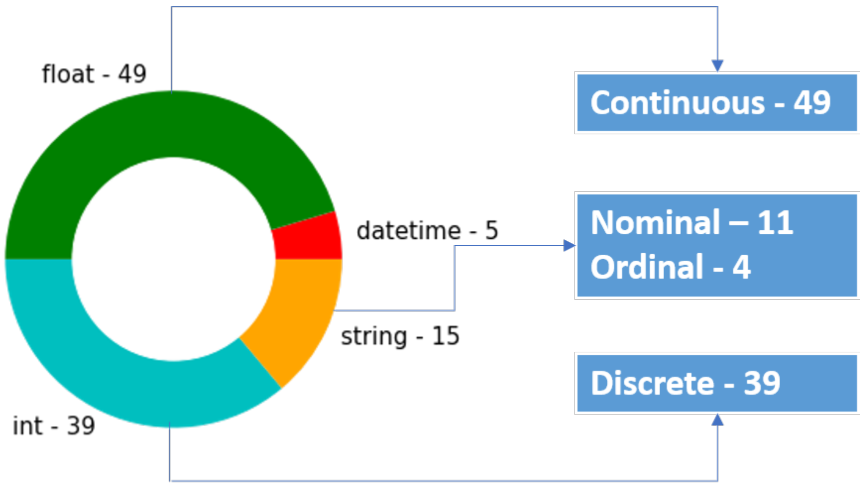


Figure 1: Datatypes present

2.2 Skewness and Kurtosis

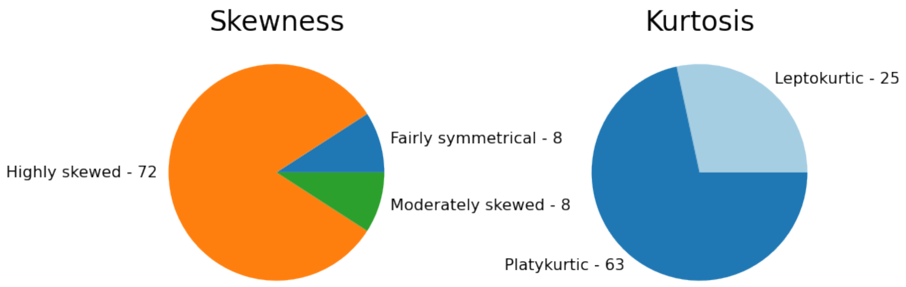


Figure 2: Skewness and Kurtosis on numerical columns

Skewness and Kurtosis is performed on numerical features. Skewness is a degree of symmetry (Dugar, 2020). Among the 88 numerical features available, most of them are highly skewed i.e. 72 features. Moderately skewed features are 8 and the rest 8 are fairly symmetrical. Kurtosis determines whether data is heavy-tailed or light-tailed. 25 features have Leptokurtic distribution, where the data is heavily tailed i.e. these columns have more outliers. Other 63 features have Platykurtic distribution, where the presence of outliers is less.

2.3 Missing data

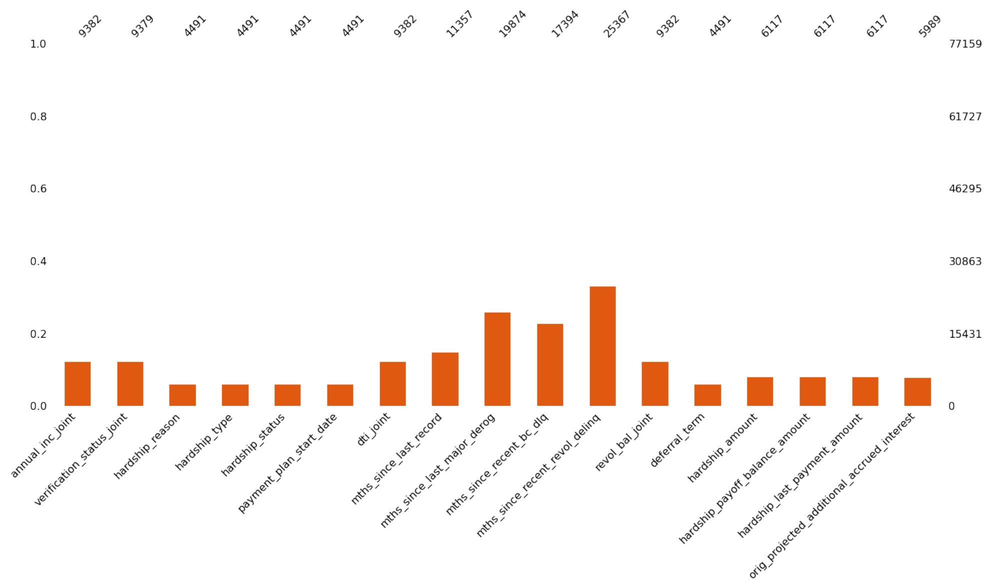


Figure 3: Columns with morethan 60% null values

Missing information will have significant impact on predictive analysis. Out of 108 columns, 17 columns contain more than 60% null values, whose pattern of missing is shown in Figure 3. The number at the top of each bar indicates the number of values present. As these columns have more nulls than values, these columns can be ignored before applying any algorithm.

As shown in Figure 4 More than 80 columns have null values less than 10% and a very few columns have null values between 10% - 20% and 50% - 60%.

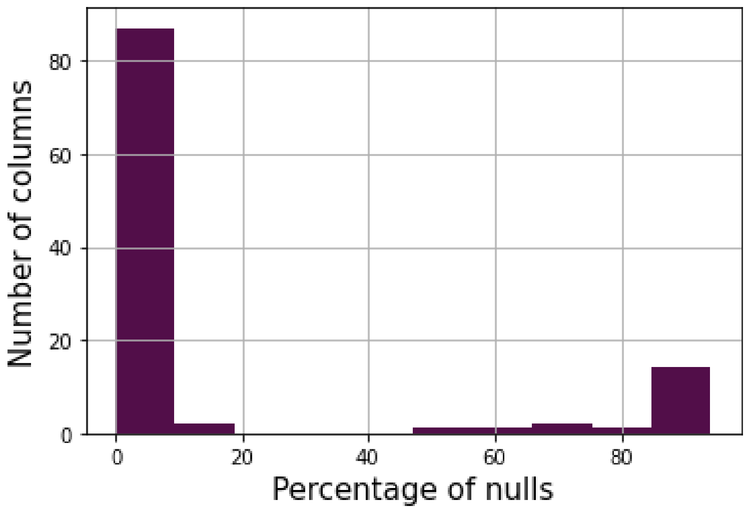
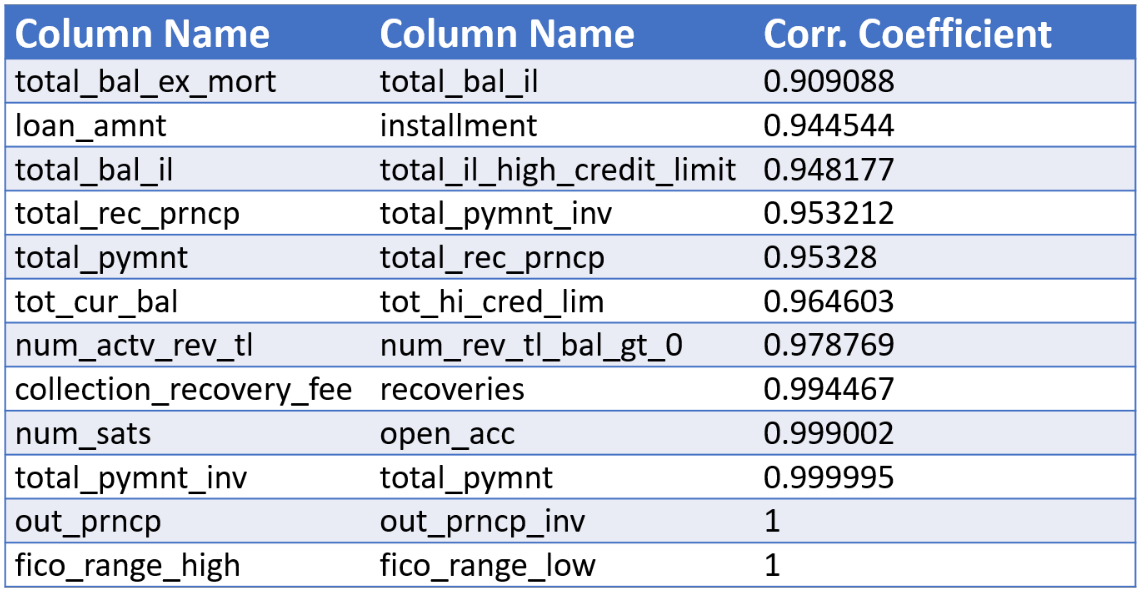


Figure 4: Null values precentage for all columns

2.4 Multicollinearity features

Multicollinearity is a potential problem, that has to be identified and sorted. Heatmap is generally used for identifying collinearity, but for 88 numerical features, finding the collinearity is difficult using heatmap. Predefined functions in pandas library helps in finding the most correlated features as shown in Figure 5.



Column Name	Column Name	Corr. Coefficient
total_bal_ex_mort	total_bal_il	0.909088
loan_amnt	installment	0.944544
total_bal_il	total_il_high_credit_limit	0.948177
total_rec_prncp	total_pymnt_inv	0.953212
total_pymnt	total_rec_prncp	0.95328
tot_cur_bal	tot_hi_cred_lim	0.964603
num_actv_rev_tl	num_rev_tl_bal_gt_0	0.978769
collection_recovery_fee	recoveries	0.994467
num_sats	open_acc	0.999002
total_pymnt_inv	total_pymnt	0.999995
out_prncp	out_prncp_inv	1
fico_range_high	fico_range_low	1

Figure 5: Most correlated features

2.5 Some important features

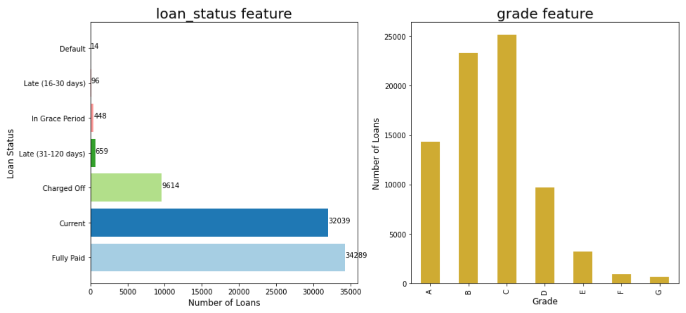


Figure 6: Loan Status and grade distribution

loan_status is a dependent and final target variable which describes the current status of the loan. From Figure 6, it is clearly evident that majority of the loans are fully paid and quite a large number of loans are in current state. The distribution of data if highly imbalanced, this has to be handled using grouping techniques.

Grade is another categorical feature, which is ordinal. The order of the grade should be A>B>C>D>E>F>G, this helps in prioritising the loan. Majority of the loans fall under Grades A, B, C and a very few loans fall under E, F, G.

home_ownership represents the ownership status of the borrower during the loan registration. From Figure 7 it is clearly seen that, most of the loans fall under mortgage and rent category. A very few handful of loans fall under Any or None category. application_type is a categorical feature with only 2 values, individual or joint application. As per the distribution of loans shown in 7, most of the loans are individual and very less loans fall under joint application.

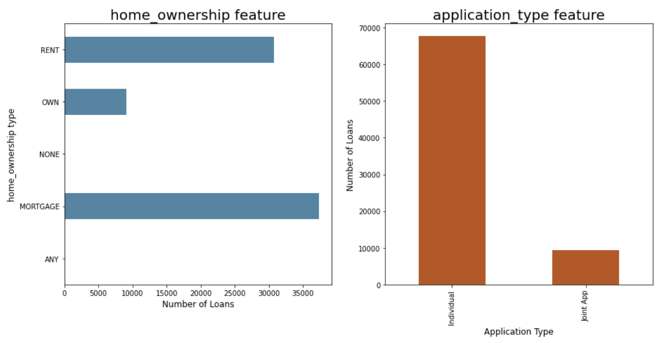


Figure 7: Home Ownership Application Type

3 Data pre-processing Stages

3.1 Split the columns based on datatype

Creating 3 separate dataframes based on datatype will be very useful while working on data preprocessing.

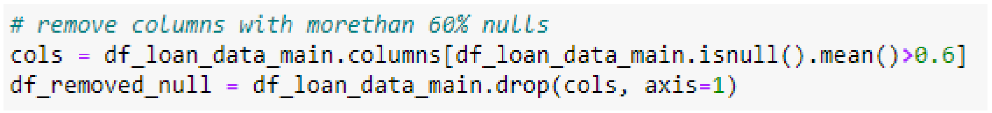
- **df_numerical** → columns with int and float datatypes
- **df_string** → columns with object datatype
- **df_datetime** → columns with datetime datatype

3.2 Handling missing values

In any real-world dataset, missing values is a very common phenomena, but the dataset must be comlpete to build a machine learning model (GeeksforGeeks, 2019). Missing values must be dealt carefully during the data analysis as removing observations might lead to skewed perspective of the data.

3.2.1 Dropping columns with morethan 60% nulls

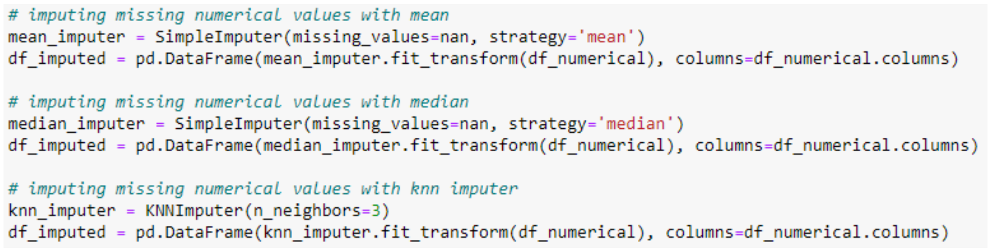
Few columns will have more null values than the actual values, these columns will decrease the performance of the model. A basic and effective procedure for handling such columns is to remove them with some threshold. In this case, columns with morethan 60% null values will be removed.



```
# remove columns with morethan 60% nulls
cols = df_loan_data_main.columns[df_loan_data_main.isnull().mean()>0.6]
df_removed_null = df_loan_data_main.drop(cols, axis=1)
```

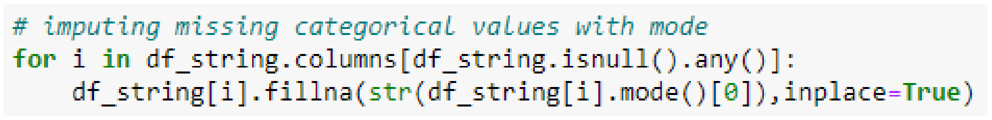
3.2.2 Imputing columns

- Imputing numerical features can be done with the measures of central tendency, by using the **mean** or **median** which works well when the values are randomly missing. Using mean or median could have some bias during the estimation of variances and covariance. Another imputation method based on distance is **KNN based imputation**. Even though it gives best results, it is a time-consuming process (Akinfaderin, 2018). All the above mentioned three strategies were experimented and found KNN to be the best among three.



```
# imputing missing numerical values with mean
mean_imputer = SimpleImputer(missing_values=nan, strategy='mean')
df_imputed = pd.DataFrame(mean_imputer.fit_transform(df_numerical), columns=df_numerical.columns)

# imputing missing numerical values with median
median_imputer = SimpleImputer(missing_values=nan, strategy='median')
df_imputed = pd.DataFrame(median_imputer.fit_transform(df_numerical), columns=df_numerical.columns)

# imputing missing numerical values with knn imputer
knn_imputer = KNNImputer(n_neighbors=3)
df_imputed = pd.DataFrame(knn_imputer.fit_transform(df_numerical), columns=df_numerical.columns)
```

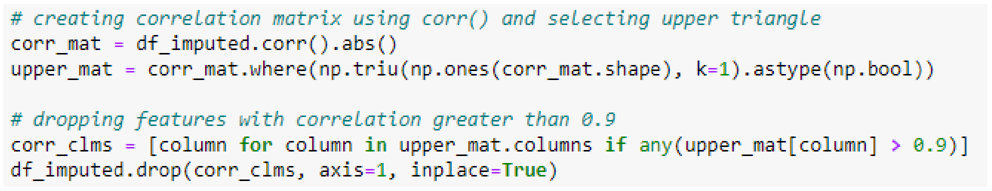
- For categorical features mode can be used as a measure of central tendency.



```
# imputing missing categorical values with mode
for i in df_string.columns[df_string.isnull().any()]:
    df_string[i].fillna(str(df_string[i].mode()[0]),inplace=True)
```

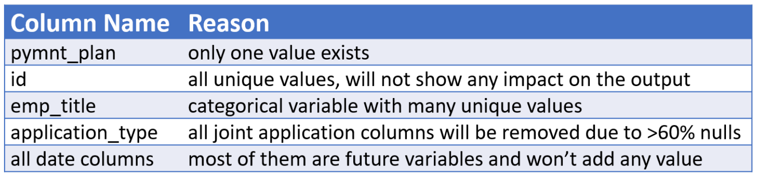
3.3 Removing attributes

- To avoid multi collinearity, repeated variables must be removed (Nilsson and Shan, 2018). As mentioned in the features summary, attributes with more than 0.9 correlation coefficient will be removed. By removing the highly correlated features, runtime will be improved and bias will be decreased.



```
# creating correlation matrix using corr() and selecting upper triangle
corr_mat = df_imputed.corr().abs()
upper_mat = corr_mat.where(np.triu(np.ones(corr_mat.shape), k=1).astype(np.bool))

# dropping features with correlation greater than 0.9
corr_clms = [column for column in upper_mat.columns if any(upper_mat[column] > 0.9)]
df_imputed.drop(corr_clms, axis=1, inplace=True)
```

- Below mentioned columns has to be removed.



Column Name	Reason
pymnt_plan	only one value exists
id	all unique values, will not show any impact on the output
emp_title	categorical variable with many unique values
application_type	all joint application columns will be removed due to >60% nulls
all date columns	most of them are future variables and won't add any value

3.4 Encoding categorical features

All the categorical variables whether ordinal or nominal must be converted to numbers as Machine Learning algorithms cannot process strings (is a Data Science enthusiast, 2020).

3.4.1 Ordinal encoding

Ordinal encoding must have ranked ordering between the values, Outof 4 ordinal categorical features, verification_status_joint feature is removed. Other 3 columns were ranked as shown below.

(1) grade $\rightarrow$ G < F < E < D < C < B < A
(2) term $\rightarrow$ 36 months < 60 months
(3) verification_status $\rightarrow$ Not Verified < Source Verified < Verified

```
# ordinal encoding on grade, term and verification_status attributes
grade_mapper = {"A":7, "B":6, "C":5, "D":4, "E":3, "F":2, "G":1}
term_mapper = {"36 months":1, "60 months":2}
verification_status_mapper = {"Verified":3, "Not Verified":1, "Source Verified":2}

df_string["grade"] = df_string["grade"].replace(grade_mapper)
df_string["term"] = df_string["term"].replace(term_mapper)
df_string["verification_status"] = df_string["verification_status"].replace(verification_status_mapper)
```

3.4.2 One Hot encoding

One Hot encoding is generally used for the nominal features. For each unique value in the feature, onehot encoder will create a new column. These features also known as dummy variables, each category is classified either as 0 or 1. Dummy encoding is applied for home_ownership, purpose, hardship_flag, initial_list_status features.

```
# using dummy encoder for encoding categorical values
nominal_columns = ['home_ownership', 'purpose', 'hardship_flag', 'initial_list_status']
dummy_df = pd.get_dummies(df_string[nominal_columns])
df_string = pd.concat([df_string, dummy_df], axis=1)
df_string = df_string.drop(nominal_columns, axis=1)
```

3.5 Handling outliers

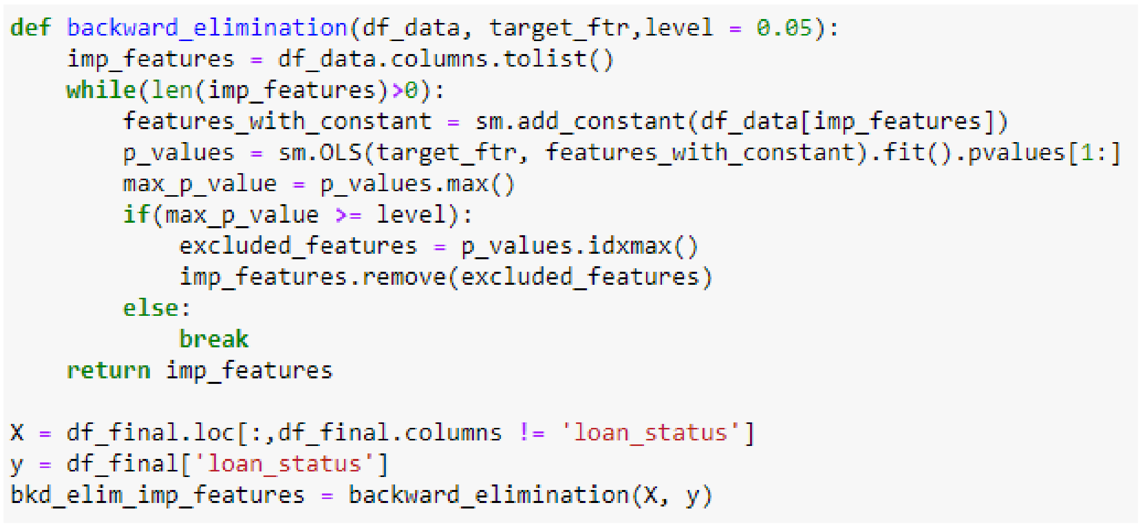
Outlier is a data point which stands out from the actual distribution of data which may be present due to the variability of data. Outliers can be find and eliminated using Interquartile Range method and Z-Score method (Yemulwar, 2019). Both methods are experimented and found Z-Score is more feasible as it covers more data points.

```
# filtering outliers using IQR method
Q1 = df_final[df_imputed.columns].quantile(0.25)
Q3 = df_final[df_imputed.columns].quantile(0.75)
IQR = Q3 - Q1
df_final = df_final[~((df_final[df_imputed.columns] < (Q1 - 1.5 * IQR)) |
                      (df_final[df_imputed.columns] > (Q3 + 1.5 * IQR))).any(axis=1)]

# filtering outliers using zscore method
from scipy import stats
zscores = stats.zscore(df_final[df_imputed.columns])
abs_zscores = np.abs(zscores)
filtered_entries = (abs_zscores < 3).all(axis=1)
df_final = df_final[filtered_entries]
```

3.6 Feature Selection

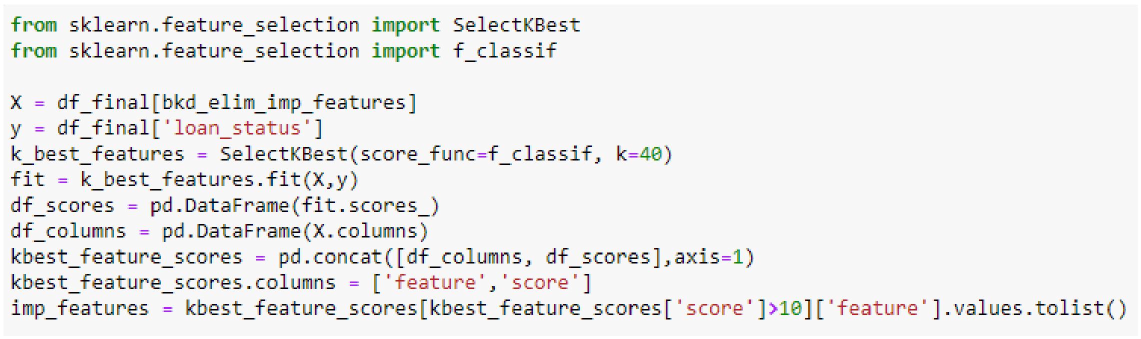
The final dataframe contains 90 independent features and one target feature. Training a machine learning model on 90 features will be time consuming, also it will contain a lot of features which cannot influence the target variable. Feature Selection referes to the selection of most relevant features from a dataset, a wide range of techniques are available to perform feature selection.

- **Backward Elimination** for classification In backward elimination process, a model will be developed with all independent features, then expel the least significant value with greater p-value. This process continues until a significant set of features is available. In this approach significant value is taken as 0.05, with this approach the count of independent features has reduced from 90 to 40.



```
def backward_elimination(df_data, target_ftr,level = 0.05):
    imp_features = df_data.columns.tolist()
    while(len(imp_features)>0):
        features_with_constant = sm.add_constant(df_data[imp_features])
        p_values = sm.OLS(target_ftr, features_with_constant).fit().pvalues[1:]
        max_p_value = p_values.max()
        if(max_p_value >= level):
            excluded_features = p_values.idxmax()
            imp_features.remove(excluded_features)
        else:
            break
    return imp_features

X = df_final.loc[:,df_final.columns != 'loan_status']
y = df_final['loan_status']
bkd_elim_imp_features = backward_elimination(X, y)
```

- **SelectKBest** is a module from sklearn, used for selecting the k best features. Internally the algorithm uses oneway ANOVA F-test to interpret the results. A total of 27 features were selected whose scores were greaterthan 10.

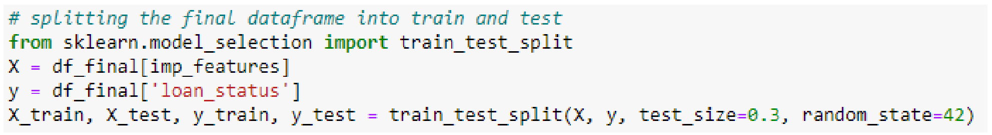


```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

X = df_final[bkd_elim_imp_features]
y = df_final['loan_status']
k_best_features = SelectKBest(score_func=f_classif, k=40)
fit = k_best_features.fit(X,y)
df_scores = pd.DataFrame(fit.scores_)
df_columns = pd.DataFrame(X.columns)
kbest_feature_scores = pd.concat([df_columns, df_scores],axis=1)
kbest_feature_scores.columns = ['feature','score']
imp_features = kbest_feature_scores[kbest_feature_scores['score']>10]['feature'].values.tolist()
```

4 Supervised Model Training and Evaluation

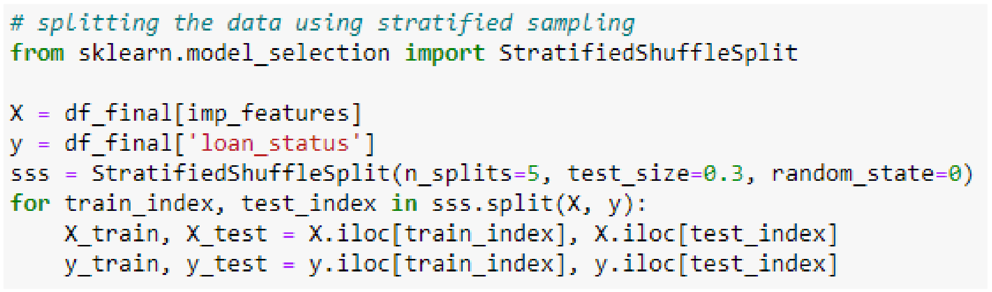
4.1 Splitting the data

- Normal train test split
  Final dataset has to be split into training and testing datasets to train and evaluate the model. Generally machine learning models will be trained on large portion of data and tested on small amounts of data. In this case, data is split in random using a module from sklearn with 70% of the data being used for training and remaining 30% for testing.



```
# splitting the final dataframe into train and test
from sklearn.model_selection import train_test_split
X = df_final[imp_features]
y = df_final['loan_status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

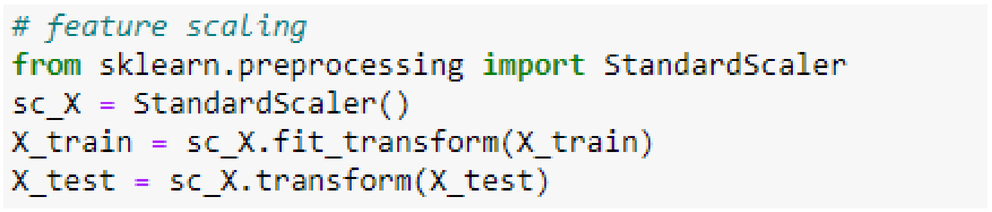
- Stratified sampling
  Simple train test split may not split the data evenly when the dataset is fully biased, stratified sampling can be used to partition the data with even distribution of data. Stratified sampling samples the data by keeping the desired feature in equal proportions. Given data is stratified into 5 subsets with both training and testing in them.



```
# splitting the data using stratified sampling
from sklearn.model_selection import StratifiedShuffleSplit

X = df_final[imp_features]
y = df_final['loan_status']
sss = StratifiedShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
for train_index, test_index in sss.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

4.2 Feature scaling

As the machine learning algorithms are profoundly delicate the features in the dataset, the values have to be in the similar magnitudes. Standardization generally follows guassian distribution, the data will be centered around the mean. Since the final dataset has values in different scale, standardization is applied using StandardScaler to transform the data with mean as 0 and standard deviation as 1 (Bhandari, 2020).



```
# feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

4.3 Classification

In supervised machine learning, the model is trained on labelled dataset. loan_status is the target variable with 7 values. Below supervised algorithms are experimented.

4.3.1 KNeighborsClassifier

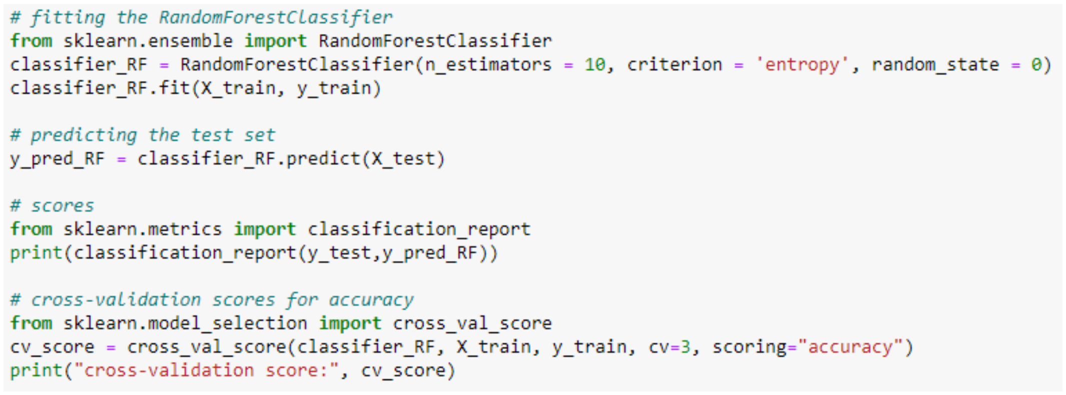
KNeighborsClassifier is based on k nearest neighbors in the feature space. KNN classifier will not make any assumptions on the independent features, it performs selection based on the proximity of other data points. It is classified by a plurality vote of its neighbors. KNN is a lazy learning algorithm, so it has very little training phase.

4.3.2 DecisionTreeClassifier

DecisionTreeClassifier is a flowchart that represents decisions in the leaf nodes. To split the data until it belongs to some class, entropy is used as a measure and forms a set of it-then-else rules as a tree. The complexity of the tree is proportional to the depth of the tree. Decision trees are capable of handling both categorical and numerical data.

4.3.3 RandomForestClassifier

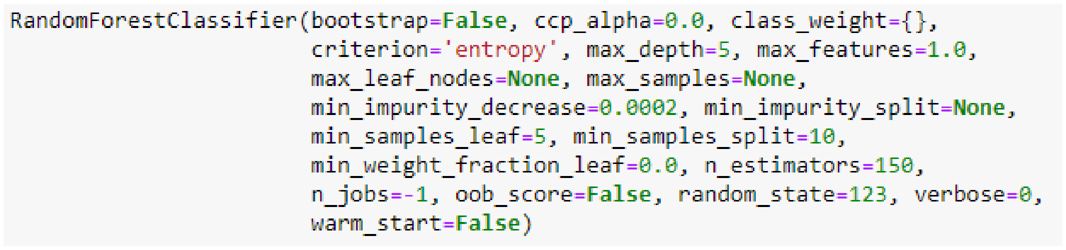
RandomForestClassifier is an ensemble learning algorithm which creates decision trees from a randomly selected subset data of the training dataset. The final class will be decided by aggregating the votes from all the subset decision trees.



```python
# fitting the RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
classifier_RF = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier_RF.fit(X_train, y_train)

# predicting the test set
y_pred_RF = classifier_RF.predict(X_test)

# scores
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred_RF))

# cross-validation scores for accuracy
from sklearn.model_selection import cross_val_score
cv_score = cross_val_score(classifier_RF, X_train, y_train, cv=3, scoring="accuracy")
print("cross-validation score:", cv_score)
```

4.3.4 Hyperparameter tuning and feature importance

Out of the three machine learning algorithms experimented, RandomForestClassifier has the highest accuracy 0.96. To improve the performance of the classifier, hyperparameters are applied as shown below. After finetuning, the accuracy of the model has improved to 0.9604. Even-though the accuracy has not shown significant improvement, f1-score has improved from 0.78 to 0.9558.



```
RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight={},
                       criterion='entropy', max_depth=5, max_features=1.0,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0002, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=150,
                       n_jobs=-1, oob_score=False, random_state=123, verbose=0,
                       warm_start=False)
```

Figure 8 shows the features that contributed most for the classifier.

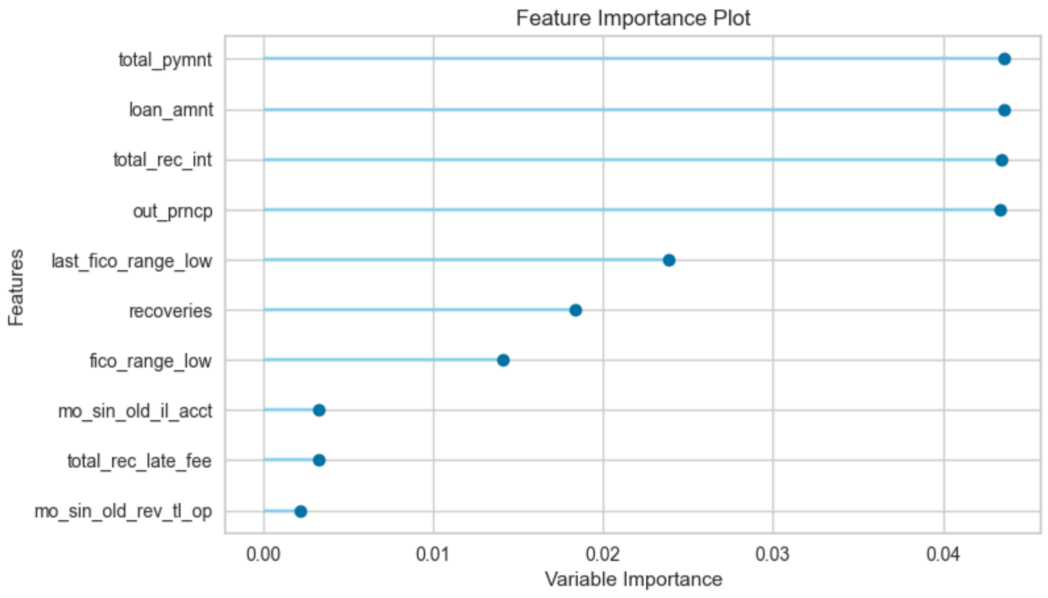
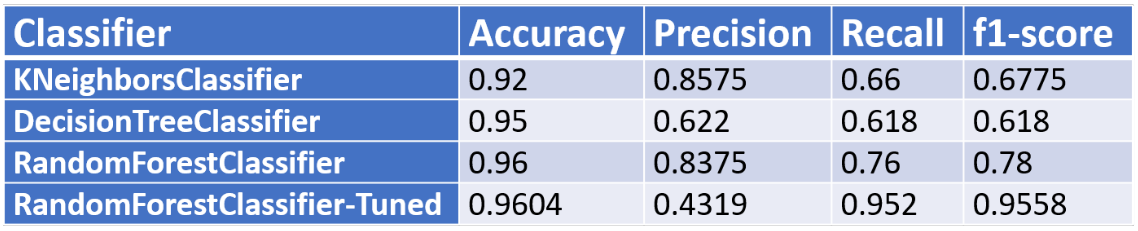


Figure 8: Feature importance plot

4.3.5 Comparison of supervised models performance

Figure 9 shows the comparison of the supervised learning models experimented. precision, recall, f1-score and accuracy are the evaluation metrics used.



Classifier	Accuracy	Precision	Recall	f1-score
KNeighborsClassifier	0.92	0.8575	0.66	0.6775
DecisionTreeClassifier	0.95	0.622	0.618	0.618
RandomForestClassifier	0.96	0.8375	0.76	0.78
RandomForestClassifier-Tuned	0.9604	0.4319	0.952	0.9558

Figure 9: Comparison of supervised models

5 Unsupervised clustering - Kmeans

Clustering is the grouping of similar data points together to form a subgroup i.e. forming homogeneous subgroups based on similarity measure like euclidean distance or correlation distance. Unlike supervised learning, there will not be any labelled data available to evaluate the model by comparing the output.

Kmeans clustering is one of the most used method for unsupervised learning, used to examine the structure of data. It is an iterative algorithm which partitions the data into K pre-defined clusters by keeping the clusters as diverse as possible. Expectation-Maximization is the approach used in Kmeans clustering (Dabbura, 2020).

5.1 Elbow method

For implementing any clustering algorithm, finding the correct number of clusters(K) is important. Elbow method is used for finding the number of clusters based on the sum of squared distance(SSE) between the data points and cluster's centroid. K value must be picked from the plot where the SSE starts flattening by forming an elbow curve (Dabbura, 2020). As shown in the Figure 10, the best K value for the given dataset is 5.

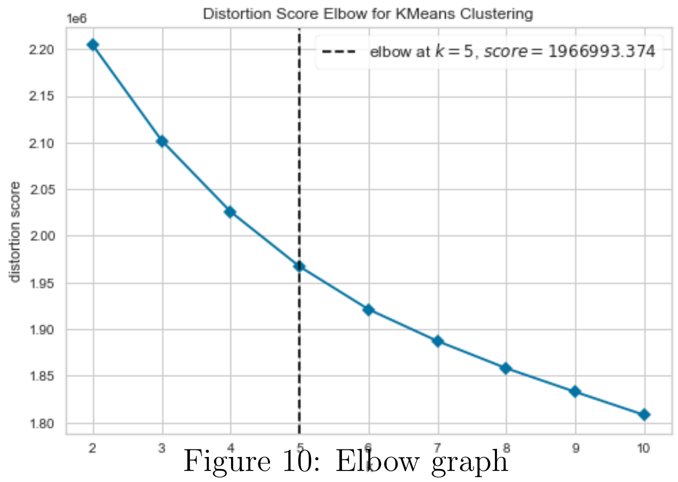


Figure 10: Elbow graph

5.2 PCA

Principal Component Analysis(PCA) is a statistical method used to reduce the dimensionality of data, which helps in understanding and plotting the data with fewer dimentions than in original data. PCA computes the vectors that have high variance and uncorrelated from the data and selects the vectors with high variance. Figure 11 shows the PCA of the given dataset with 5 clusters, which clearly indicates the overlap of clusters.

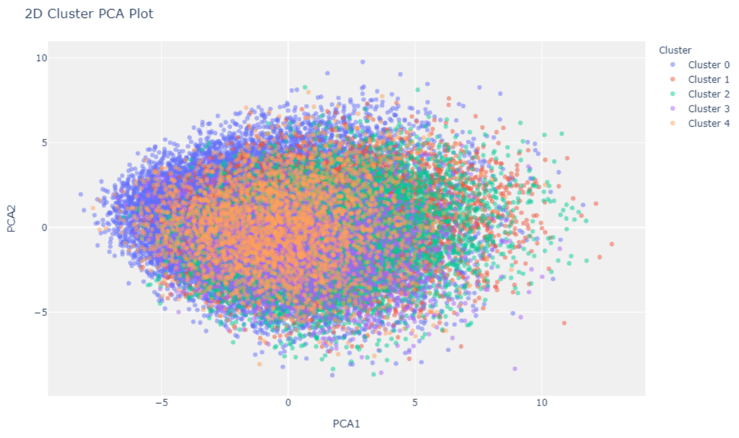


Figure 11: PCA

5.3 Evaluation - silhouette

Silhouette analysis can be used as an evaluation metric for Kmeans clustering. The degree of separation can be find using silhouette analysis by computing a coefficient for each sample. If the coefficient is 0 - sample is close to neighboring cluster, if the coefficient is 1 - sample is far from the neighboring cluster and if the coefficient is -1 - sample is assigned in the wrong cluster. So, it is desirable to have coefficient of all samples close to 1. With 5 clusters, the average of silhouette score is less than 0.1 as shown in Figure 12. Since the average score is less than 0.1, clustering is not a good approach for the given dataset.

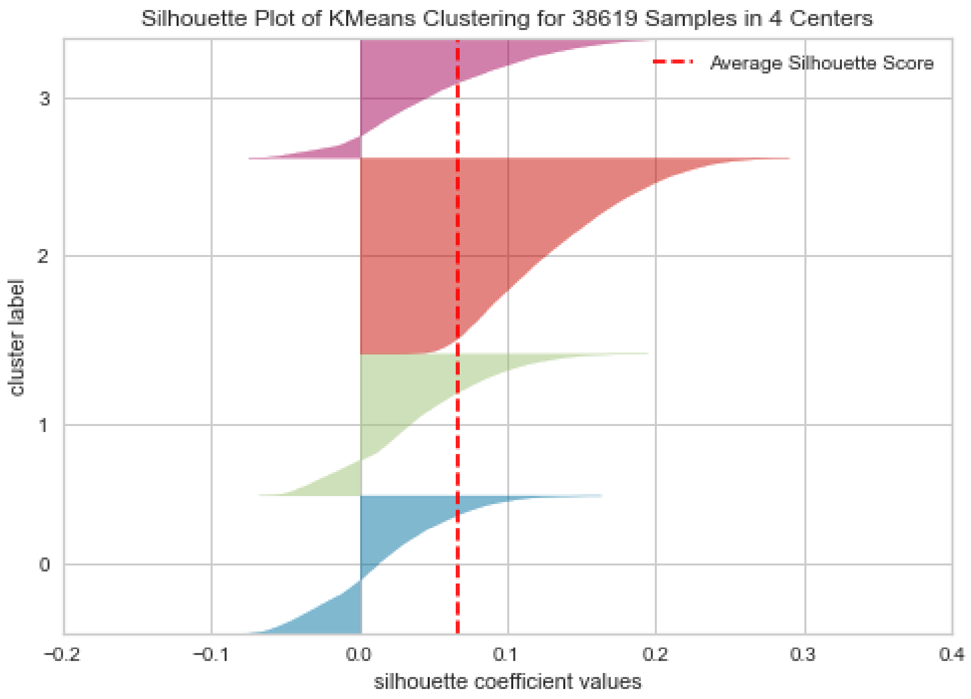


Figure 12: Silhouette evaluation

6 Conclusion

With the detalied analysis on the comparison of supervised algorithms, RandomForest classifier gives the highest accuracy of 0.96 when compared with KNN and DecisionTree classifiers when the target variable has highly imbalanced data. By finetuning the model with hyperparamets, the models accuracy has increased to 0.9604, but with a significant improvement in f1-score. Unsupervised learning implementation is done with KMeans clustering, but the results are not satisfactory as the silhouette score is close to 0.1. From this analysis, it is very clear that unsupervised clustering cannot give promisong results with the given dataset.

References

Akinfaderin, W. (2018). Missing data conundrum: Exploration and imputation techniques. https://medium.com/@WalePhenomenon/missing-data-conundrum-exploration-and-imputation-techniques-9f40abe0fd87.

Bhandari, A. (2020). Feature scaling: Standardization vs normalization. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

Dabbura, I. (2020). K-means clustering: Algorithm, applications, evaluation methods, and drawbacks. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a.

Dugar, D. (2020). Skew and kurtosis: 2 important statistics terms you need to know in data science. https://codeburst.io/2-important-statistics-terms-you-need-to-know-in-data-science-skewness-and-kurtosis-388fef94eeaa.

GeeksforGeeks (2019). Python: Visualize missing values (nan) values using missingno library. https://www.geeksforgeeks.org/python-visualize-missing-values-nan-values-using-missingno-library/.

is a Data Science enthusiast, S. S. (2020). 8 categorical data encoding techniques to boost your model in python! https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/.

Nilsson, M. and Shan, Q. (2018). Credit risk analysis with machine learning techniques in peer-to-peer lending market. Master's thesis, Stockholm University, Stockholm Business School.

Yash (2020). Kaggle:lending club 2007-2020q3. https://www.kaggle.com/ethon0426/lending-club-20072020q1.

Yemulwar, S. (2019). Outlier treatment. https://medium.com/analytics-vidhya/outlier-treatment-9bbe87384d02.