Here are the comprehensive explanations of my task for data scientist position.

1. Data Understanding

Before I indulge myself with the given dataset, it is important to understand the definition of each columns in the dataset. With that information, I can confidently make rationale and objective decisions in the process of data cleaning, data exploration, feature selection and feature engineering later. Here are the definitions of columns:

No.	Column Name	Definition
1	Loan_ID	Unique identifier (ID) for each loan.
2	Loan_Amount_Requested	The listed amount of the loan applied for by the borrower.
3	Length_Employed	Employment length in years.
4	Home_Owner	The home ownership status provided by the borrower during registration. Values are Rent, Own, Mortgage, Other.
5	Annual_Income	The annual income provided by the borrower during registration.
6	Income_Verified	Indicates if income was verified, not verified, or if the income source was verified.
7	Purpose_of_Loan	A category provided by the borrower for the loan request.
8	Debt_to_Income	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested loan, divided by the borrower's self-reported monthly income.
9	Inquiries_Last_6Mo	The number of inquiries by creditors during the past 6 months.
10	Months_Since_Delinquency	The number of months since the borrower's last delinquency.
11	Number_Open_Accounts	The number of open credit lines in the borrower's credit file.
12	Total_Accounts	The total number of credit lines currently in the borrower's credit file.
13	Gender	Gender of the borrower.
14	Interest_Rate	Target Variable: Interest Rate category (1/2/3) of the loan application.

2. Data Cleaning

Data cleaning procedures is the next step to be done after data understanding step. In this step, I will identify and treat duplicates, wrong data type assignments, missing data, and anomalies within the dataset given, if it exists. A good machine learning model comes from a clean dataset, after all. Other than that, it will smoothen the data exploration process. In this step, I was using Pandas and NumPy for data wrangling purposes.

a. Duplicates

Since the dataset has a unique identifier column, the difference in amount of unique number of that column and the total number of rows will indicate the presence of duplicate observations. Since there were no differences in train and test dataset, there was no duplicates identified.

b. Wrong Data Type Assignments

From the dataset, there was only 1 column with wrong data type assignment which is the Loan Amount Requested column. The wrong data type assignment was caused by the number input format, specifically, by the presence of comma(s) that indicates the decimal separator of the number. To fix this problem, I removed the comma(s) then reassigned the column to be float.

```
# Checking for the data
df['Loan_Amount_Requested'][0:5]
```

executed in 8ms, finished 15:03:35 2022-08-24

```
0     7,000
1    30,000
2    24,725
3    16,000
4    17,000
Name: Loan_Amount_Requested, dtype: object
```

```
# Creating an empty list to contain the correct values
correct_df = []
# Looping correction for train
for i in df['Loan_Amount_Requested']:
    i=i.split(',') # Splitting the data using comma, transforming it as a list
    if len(i) > 1: # For 6 characters
        i=i[0]+i[1]
    elif len(i) == 1: # For 3 characters
        i=i[0]
    correct_df.append(i) # Inserting i to correct list
```

c. Missing Data

First, we need to identify which columns have missing data. It can be easily checked with isna().sum() or isnull().sum(). For convenience, I used a function to automatically present it as a table.

```python
def missing_values_table(df):
    # Total missing values
    mis_val = df.isnull().sum()

    # Percentage of missing values
    mis_val_percent = 100 * df.isnull().sum() / len(df)

    # Make a table with the results
    mis_val_table = pd.concat([mis_val, mis_val_percent],axis=1)

    # Rename the columns
    mis_val_table_ren_columns = mis_val_table.rename(
        columns={0: 'Missing Values', 1: '% of Total Values'})

    # Sort the table by percentage of missing descending
    mis_val_table_ren_columns = (mis_val_table_ren_columns[
        mis_val_table_ren_columns.iloc[:, 1] != 0].sort_values(
        '% of Total Values', ascending=False).round(2))

    # Print some summary information
    print("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
          "There are " + str(mis_val_table_ren_columns.shape[0]) +
          " columns that have missing values.")

    # Return the dataframe with missing information
    return mis_val_table_ren_columns
```

```
missing_values_table(df)
```

executed in 53ms, finished 15:03:39 2022-08-24

```
Your selected dataframe has 14 columns.
There are 4 columns that have missing values.
```

	Missing Values	% of Total Values
Months_Since_Deliquency	88379	53.79
Home_Owner	25349	15.43
Annual_Income	25102	15.28
Length_Employed	7371	4.49

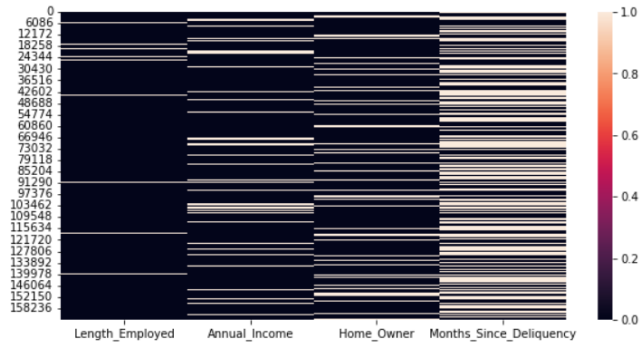
```
missing_values_table(tes)
```

executed in 48ms, finished 15:03:39 2022-08-24

```
Your selected dataframe has 13 columns.
There are 4 columns that have missing values.
```

	Missing Values	% of Total Values
Months_Since_Deliquency	58859	53.73
Annual_Income	16898	15.43
Home_Owner	16711	15.26
Length_Employed	4936	4.51

After identifying the columns with missing data, I proceed with identifying the type of missing values based on the location of missing data. Identifying that will be important information for us, on how to handle the missing data. To visualize it, I used heatmap. It is important to note that the missing data of Months Since Delinquency column indicates users with no history of credit delinquency. Thus, our focus only in 3 columns, other than Months Since Delinquency. Based on the heatmap visualization, there are no pattern of missing data, thus the missing data were missing completely at random (MCAR).



For the next step, the missing data needed to be filled. We have different options for that. The options are:

1. Deleting the observations or rows (listwise deletion): This is the easiest way; however, it will lead to potential loss of data information. Considering the MCAR and the percentage of missing data from each columns, approximately 30-35% data will be deleted. Thus, this option is not the best course of action.
2. Mean/Median/Mode Imputer: The second easiest way. However, these methods can affect the distribution of the column and reducing the variance.
3. Manual search: Using logical thinking and information from column definitions, we can search for associations or patterns between columns. It will help us to fill the missing values. For example, the length employed column logically will affect the annual income. Logically, the longer the duration of employment, the higher the annual income. If we can see a clear distinction between groups, then we can fill the missing values of annual income using median based on their length employed status. However, this approach is time consuming, inefficient and can be very subjective. Considering the time limit, I will avoid this approach.
4. Iterative imputer: Using machine learning model, they will automatically search the association between variables and use that information to automatically fill the missing values.

I believe that iterative imputer is the best missing values imputer for this task, since we have several columns that are logically connected, whether directly or indirectly. Because we have a categorical columns, I used MissForest imputer for this task. First, I searched for a column that statistically correlated to annual income, in hope that it will help the imputing process. Using spearman correlation result, I will use Loan Amount Requested column to aid the iterative imputing process.

	Annual_Income	Debt_To_Income	Loan_Amount_Requested	Number_Open_Accounts	Total_Accounts
Annual_Income	1.000000	-0.197558	0.492922	0.243814	0.341377
Debt_To_Income	-0.197558	1.000000	0.071675	0.320092	0.238470
Loan_Amount_Requested	0.492922	0.071675	1.000000	0.220147	0.248760
Number_Open_Accounts	0.243814	0.320092	0.220147	1.000000	0.672742
Total_Accounts	0.341377	0.238470	0.248760	0.672742	1.000000

In order to include the categorical columns into the imputer, we need to encode them first. I use transformer to easily encode them. The encoding details are:

```
# Mapping for categorical data
ordinal_mapping_length=[{'col':'Length_Employed',
                 'mapping':{'< 1 year':0, '1 year':1, '2 years':2, '3 years':3,
                            '4 years':4, '5 years':5, '6 years':6, '7 years':7, '8 years':8,
                            '9 years':9, '10+ years':10}}]
ordinal_mapping_home=[{'col':'Home_Owner',
                 'mapping':{'None':None,'Rent':0, 'Own':1, 'Mortgage':2, 'Other':3}}]
```

After encoding process, we need to assign the data type of categorical columns into ‘category’.

```
# Changing datatypes into category
x_preprocessed['Length_Employed']=x_preprocessed.Length_Employed.astype('category')
x_preprocessed['Home_Owner']=x_preprocessed.Home_Owner.astype('category')
```

Next, I proceed with the imputing.

```
# Get indices of categorical features
cat_cols = [x_preprocessed.columns.get_loc(col) for col in x_preprocessed.select_dtypes(['category']).columns.tolist()]
cat_cols
```

executed in 15ms, finished 15:03:51 2022-08-24

```
[0, 1]
```

```
# Creating imputer
imputer = MissForest(criterion=('squared_error','gini'),random_state=10)

# Impute missing values
imputed = imputer.fit_transform(x_preprocessed, cat_vars=cat_cols)
```

The ‘imputed’ variable will be table with complete data. Then, I assigned the value received in ‘imputed’ table into our dataset.

```
df['Home_Owner']=imputed['Home_Owner']
df['Length_Employed']=imputed['Length_Employed']
df['Annual_Income']=round(imputed['Annual_Income'],0)
```

For Months Since Delinquency column, I recategorize it into:

```
def change_delinquency(i):
    if i in range(0,4):
        return 'Recent'
    elif i in range(4,12):
        return 'Less than year, not recent'
    elif i in range(12,25):
        return '1-2 years'
    elif i in range(25,37):
        return '>2-3 years'
    elif i in range(37,61):
        return '>3-5 years'
    elif i > 60:
        return 'Longer'
    else:
        return 'No Delinquency'
```

executed in 9ms, finished 15:05:05 2022-08-24

```
df['Delinquency_Record']=df['Months_Since_Deliquency'].apply(lambda x: change_delinquency(x))
tes['Delinquency_Record']=tes['Months_Since_Deliquency'].apply(lambda x: change_delinquency(x))
```

d. Anomalies

I didn't found anomalies within the dataset. For example, in numerical columns, I searched for negative values and there wasn't. For categorical columns, I checked for its unique values (cardinality), crosschecking it with its column description.

```
# Checking the distribution of the column
df['Loan_Amount_Requested'].describe()
```

executed in 11ms, finished 15:03:36 2022-08-24

```
count    164309.00000
mean      14349.33692
std        8281.86870
min         500.00000
25%        8000.00000
50%       12075.00000
75%       20000.00000
max       35000.00000
Name: Loan_Amount_Requested, dtype: float64
```

```
object_columns = list(df.select_dtypes(include="object").columns)+
                 ['Home_Owner','Length_Employed']

for column in object_columns:
    print("{} has {} unique values."
          .format(column, df[column].nunique()))
```

executed in 52ms, finished 15:05:07 2022-08-24

```
Income_Verified has 3 unique values.
Purpose_Of_Loan has 14 unique values.
Gender has 2 unique values.
Delinquency_Record has 7 unique values.
Home_Owner has 4 unique values.
Length_Employed has 11 unique values.
```

e. Summary

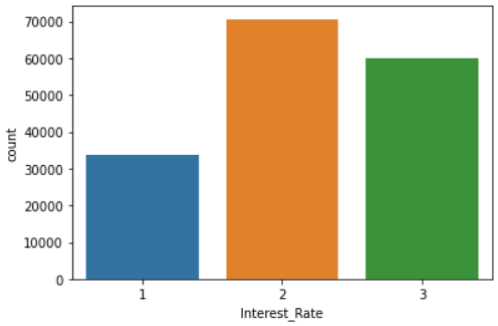
- In data cleaning step, I didn't drop any columns, yet.
- Missing values in dataset was imputed iteratively using MissForest method.
- Initial feature engineering for Months Since Delinquency was done to remove the missing data, also to facilitate the dataset with information of users with no history/record of credit delinquency.

3. Data Exploration

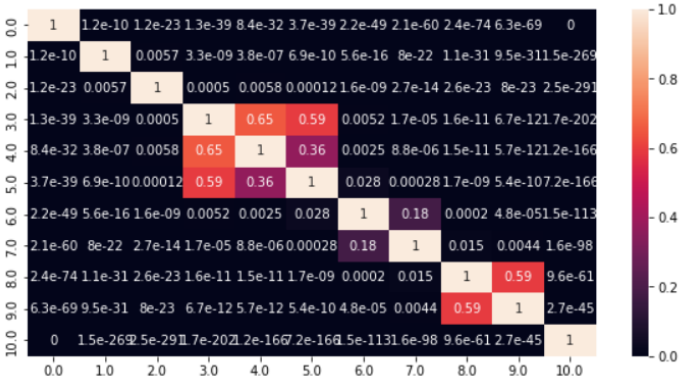
Also called as Exploratory Data Analysis (EDA), this step was performed to help me understand more about the characteristics of columns and their correlations. In the process, we can gain insights and information that will help the business decision-making process, also information related to feature selection, engineering, and data preprocessing procedures. I use Matplotlib and Seaborn packages for data visualization purposes.

Several interesting facts learned from this process:

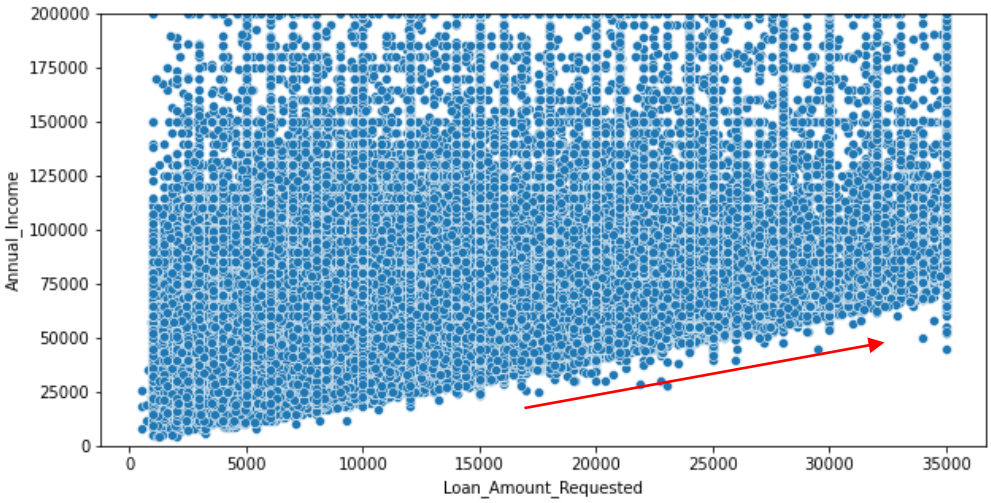
- Most people were labeled in interest rate class 2, followed by class 3 and class 1. The data was clearly imbalanced.



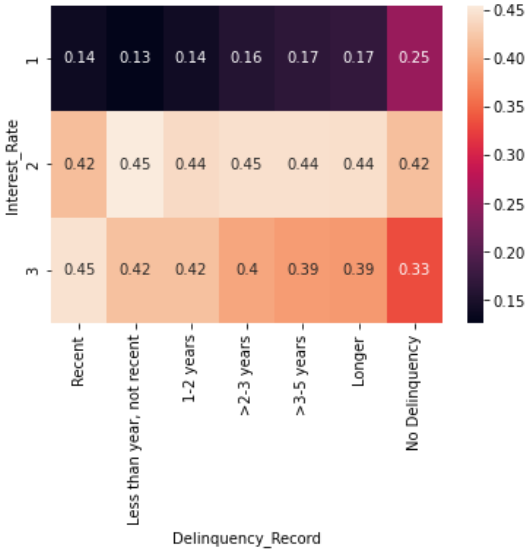
- Based on median value, there were a slight increase of annual income, based on length employed status. From the result of Conover post-hoc test, there were groups that were not statistically different to each other, such as group 3, 4, and 5 years; or 8 and 9 years.



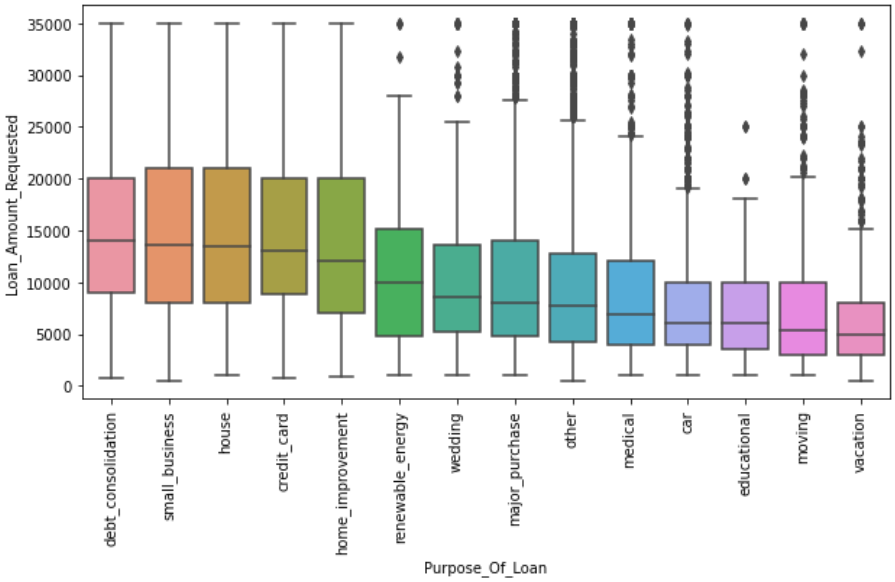
- Increase in loan amount requested was associated with increase in annual income.



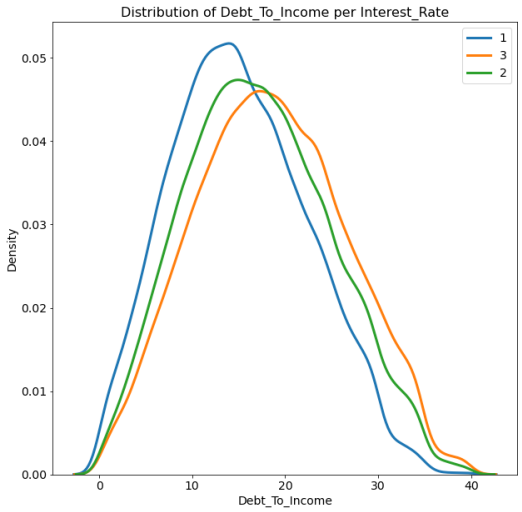
- Delinquency record affects the interest rate given. From this visualization analysis, it is clear that interest rate 1 was for users with low risks of insolvency, and interest rate 3 was for users with high risks of insolvency.



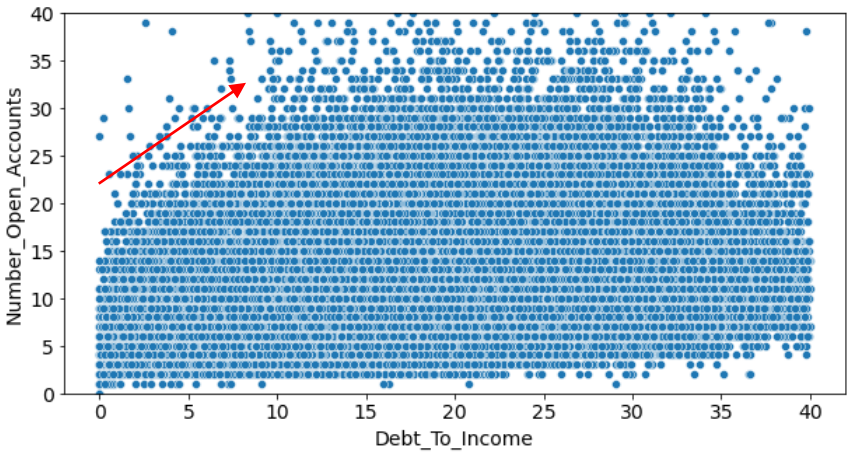
- Several purpose of loan has higher loan amount requested than the others.



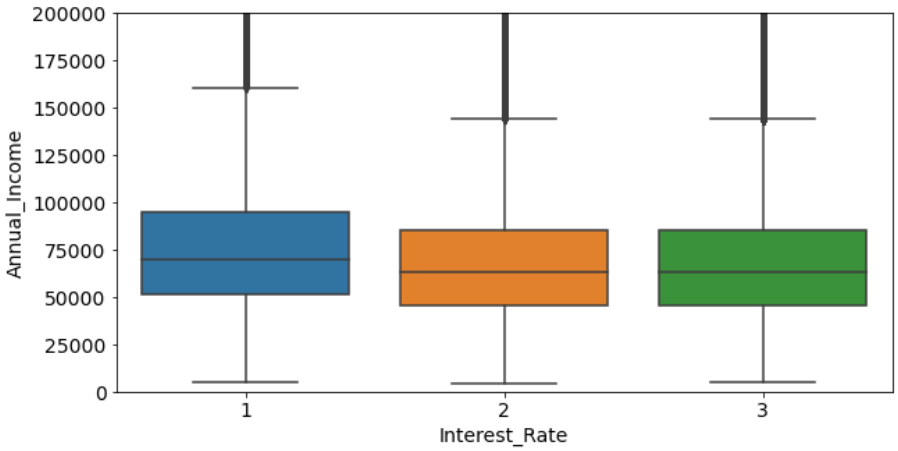
- Debt to income ratio affects the interest rate. After all, users with high debt-to-income ratio have more risks than users with low debt-to-income ratio. This finding confirms the information/evidence from the delinquency record analysis to interest rate.



- Number of open credit accounts affects the debt-to-income ratio.

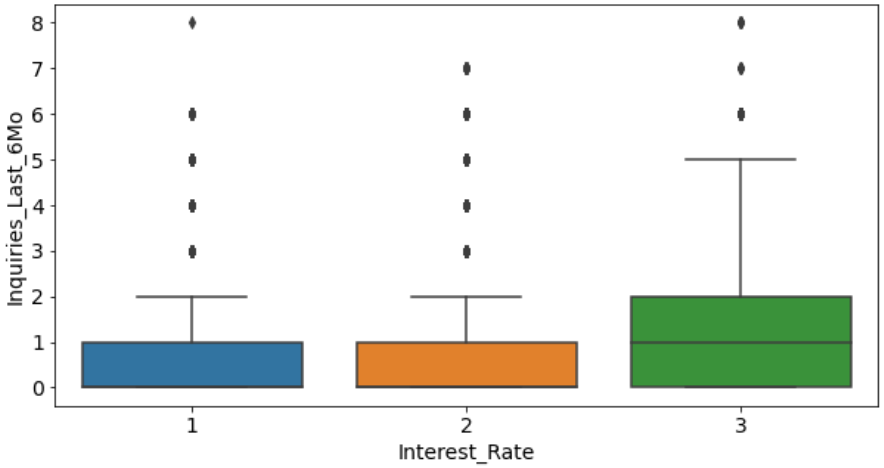


- Interest rate class 1 have higher annual income than class 2 and class 3.



- Verification status of income somehow affects the interest rate given to users. However, logically, the verification status didn't associate with risks of the insolvency. The verification status only affects the validity of the data collected. Thus, this column will be dropped in the process of prediction model building.

- The amount of credit inquiries in the last 6 months affects the interest rate of users, where interest rate class 3 has higher number of inquiries than class 1 and class 2.



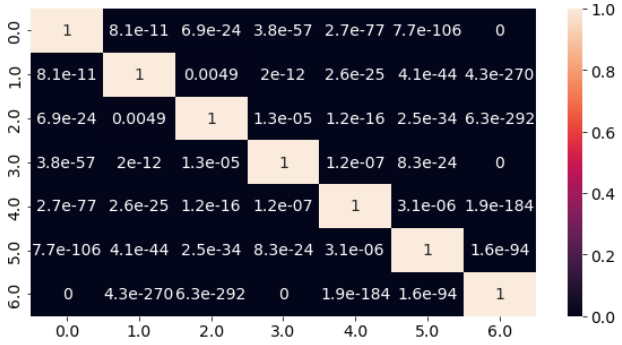
4. Feature Selection and Engineering

With the information acquired from EDA, now I can proceed with the process of feature selection and engineering.

Features Selection

First, I eliminate Loan ID, Income Verified from the dataset since their information won't help the prediction model. Months Since Delinquency column also dropped because already replaced by Delinquency Record column. Others will be used in model building process.

Feature Engineering



Based on the information of length employed vs annual income analysis, I recategorized the values of length employed column to be simpler, and to be more distinct to each other. I recategorized it into: >1 year (0), 1 year (1), 2 years (2), 3-5 years (3), 6-7 years (4), 8-9 years (5), 10+ years (6).

Then, I checked the multicollinearity first using variance inflation factor (VIF) values.

	feature	VIF
0	Annual_Income	3.110258
1	Debt_To_Income	5.009057
2	Loan_Amount_Requested	4.491174
3	Number_Open_Accounts	11.498267
4	Total_Accounts	10.832279
5	Inquiries_Last_6Mo	1.566407

From the result, Number Open Accounts and Total Accounts were highly correlated to each other. I can remove one of them to proceed. However, removing one of them will remove the information which is probably important. Thus, I merged the two columns.

```
# This new column will indicate the proportion of open credit line from the total credit line of the borrowers
# This column will contain the combined information from both columns
df['Open_Per_Total_Acc']=df['Number_Open_Accounts']/df['Total_Accounts']
df.drop(columns=['Number_Open_Accounts','Total_Accounts'], inplace=True)
```

5. Data Preprocessing

Scaling and Encoding

With clean dataset and defined selection of features, I can prepare the dataset for modeling purposes. First, I create a transformer to standardize some of the numerical features using Robust Scaler: Loan Amount Requested, Annual Income, and Debt to Income. Robust scaling was selected due to the presence of outliers in Annual Income column. Then, I need to encode several categorical columns using one hot encoder. Usually, I will use binary encoder for categorical columns with unique values more than 5; however, because the features were just a few, it won't hurt to use one hot encoder. It will also make a clear distinction of groups in features importance visualization process.

```
transformer = ColumnTransformer([
    ('robust', RobustScaler(),['Loan_Amount_Requested','Annual_Income','Debt_To_Income']),
    ('onehot', OneHotEncoder(drop='first'),['Purpose_Of_Loan','Gender','Delinquency_Record'])
], remainder = 'passthrough')
```

Data Splitting

I split the data into 70% of training data and 30% of test data.

```
x_train, x_test, y_train, y_test=train_test_split(x,y,stratify=y,test_size=0.3,random_state=2020)
```

6. Selecting Evaluation Metrics

Before modeling, I need to select the appropriate evaluation metrics. Because this is a multiclass classification problem and we are interested in all interest rate class, I will use ROC-AUC One-vs-One (OvO) evaluation metric. I assumed that the data imbalance will affect the model, thus I avoided using accuracy evaluation metric for this task.

7. Selecting Best Model Classifier

Cross-validation method was used to select the best model classifier from the selection of:

a. Logistic regression represents the basic method of model-based classification.
b. K-Nearest Neighbors represents the basic method of instance-based classification.
c. Random Forest represents the ensemble, tree-based classification using bagging procedure.
d. XGBoost represents the ensemble, tree-based classification using boosting procedure.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost.sklearn import XGBClassifier
```

executed in 67ms, finished 15:06:06 2022-08-24

```
model_logreg = LogisticRegression(random_state=10)
model_knn= KNeighborsClassifier()
model_xgbc= XGBClassifier(random_state=10)
model_forest= RandomForestClassifier(random_state=10)
```

executed in 12ms, finished 15:06:07 2022-08-24

For cross-validation, Stratified K-Fold procedure with GridSearch was used. Pipeline was used to prevent information/data leakage from training to test dataset. 3-fold cross-validation was enough due to large size of data the dataset has.

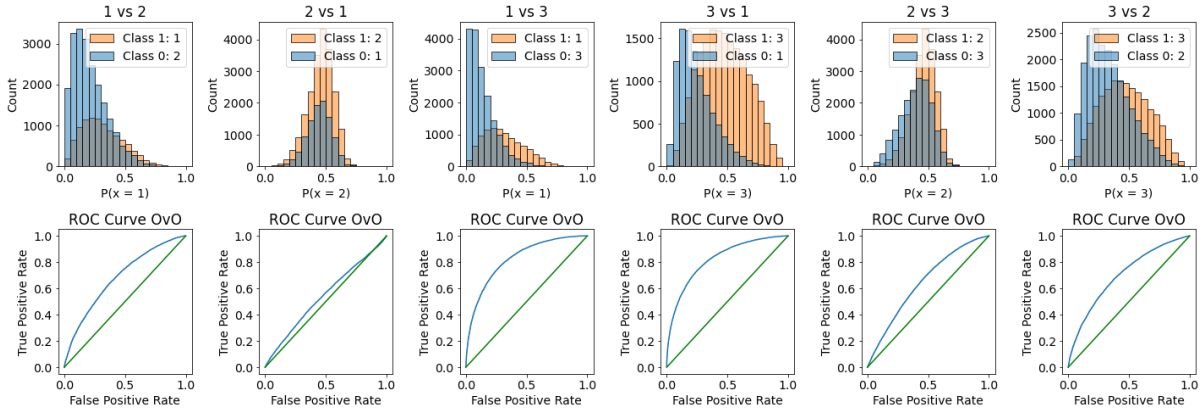
```
estimator_withoutresampling=Pipeline([
    ('preprocess',transformer),
    ('model',model_xgbc)
])
```

```
skfold=StratifiedKFold(n_splits=3)
grid=GridSearchCV(
    estimator_withoutresampling,
    param_grid=hyperparam_space,
    cv=skfold,
    scoring='roc_auc_ovo', # OVO for one-vs-one comparison, OVR for one-vs-rest comparison
    n_jobs=-1
)
```

The results were (1m 27s):

params	Methods	Score	Standard Deviation	Rank
0	Logistic Regression	0.680228	0.001051	2
1	XGBoost	0.697081	0.000315	1
2	K-Nearest Neighbors	0.605888	0.000954	4
3	Random Forest	0.669419	0.001013	3

From cross-validation result, XGBoost was selected to be the best classifier. Now, using model trained in test dataset, the results are:



The details are:

```
1 vs 2 ROC AUC OvO  : 0.682954068846541
2 vs 1 ROC AUC OvO  : 0.542593365389331
1 vs 3 ROC AUC OvO  : 0.822372110216381
3 vs 1 ROC AUC OvO  : 0.819522594477988
2 vs 3 ROC AUC OvO  : 0.643745855817078
3 vs 2 ROC AUC OvO  : 0.699854449420612
Average ROC AUC OvO : 0.701840407361322
```

From the results above, our initial model was barely accepted with average ROC-AUC OvO value of 70.18%. Since the ROC-AUC values of train and test dataset are comparable, our model is not overfit or underfit. With detailed OvO comparisons, we can clearly see that the model can make a good distinction between interest rate class 1 and class 3 with ROC-AUC values of 82.24% and 81.95%. The model has low ability to make a distinction between class 2 and the others because the awkward position of 2, in the middle of class 1 and class 3. To simplify it, the distance between class 1 and class 3 is far enough, however class 2 stands in the middle of both classes, make it not different/far enough compared to class 1 and/or class 3.

Can I improve the model performance?

8. Model Improvement Procedures

In order to improve the model performance, I did the following procedures:

a. Treating Data Imbalance

This procedures were done along with cross-validation step because I assume that the data imbalance will affect the model performance. To balance the proportion of class in the dataset, I used several resampling methods: oversampling using SMOTE, undersampling using NearMiss, and the combination of oversampling and undersampling using SMOTE-Tomek. The results:

```
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
```

executed in 100ms, finished 15:06:02 2022-08-24

```
smote=SMOTE()
nearmiss=NearMiss()
smotomek = SMOTETomek(tomek=TomekLinks(sampling_strategy='majority'))
```

executed in 14ms, finished 15:06:03 2022-08-24

```
estimator_smote=Pipeline([
    ('preprocess',transformer),
    ('resampling',smote),
    ('model',model_xgbc)
])
```

SMOTE (9m 53s)

	params	mean_test_score	std_test_score	rank_test_score
0	{'model': LogisticRegression(random_state=10)}	0.682936	0.000911	2
1	{'model': XGBClassifier(base_score=None, boost...	0.693417	0.000425	1
2	{'model': KNeighborsClassifier()}	0.604276	0.001770	4
3	{'model': RandomForestClassifier(random_state=...	0.669808	0.001217	3

NearMiss (6m 18s)

	params	mean_test_score	std_test_score	rank_test_score
0	{'model': LogisticRegression(random_state=10)}	0.630210	0.000628	1
1	{'model': XGBClassifier(base_score=None, boost...	0.613096	0.000781	2
2	{'model': KNeighborsClassifier()}	0.582603	0.000111	4
3	{'model': RandomForestClassifier(random_state=...	0.607379	0.001605	3

SMOTE-Tomek (1h 30m 15s)

	params	mean_test_score	std_test_score	rank_test_score
0	{'model': LogisticRegression(random_state=10)}	0.682891	0.000890	2
1	{'model': XGBClassifier(base_score=None, boost...	0.693558	0.001373	1
2	{'model': KNeighborsClassifier()}	0.603057	0.001360	4
3	{'model': RandomForestClassifier(random_state=...	0.669733	0.001799	3

From the results above, models the resampling methods have comparable performance with the model without resampling method. Thus, it confirmed that data imbalance wasn't a problem in this task. Considering the model fitting time, I considered model without resampling method to be tuned.

b. Hyperparameter Tuning

Can I improve the model by tweaking the model parameters?

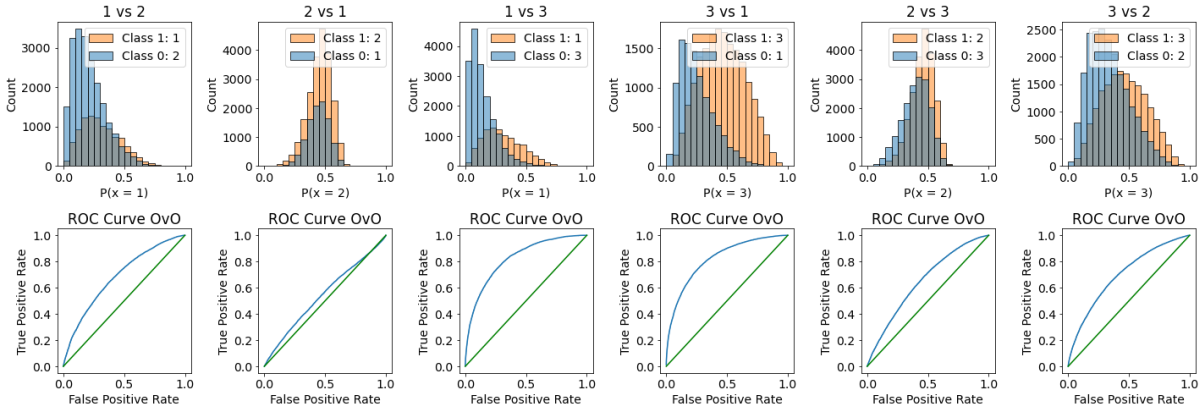
```
hyperparam_space={
    'model__max_depth':[5,10,15,20], # The depth of trees, default=None
    'model__n_estimators':[60,80,100], # The amount of trees generated, default=100
    'model__learning_rate':[0.01,0.1,0.2,0.3], # Model learning rate, boosting/weighting parameter, default=0.3
    'model__eval_metric':['merror','logloss','auc'], # Selecting eval metrics for multiclass problem
    'model__verbosity':[2] # For info
}
```

Considering the size of the dataset, I only use these parameters for hyperparameter tuning. First, I used max depth parameter, so the trees produced won't be too deep (taking longer time to be generated). Considering the dataset size, I use the combination of lower n estimator than the default, and slower model learning rate than the default. For evaluation metrics, I was just curious. The fitting procedure took 1h 3m 25s. The results:

	params	mean_test_score	std_test_score	rank_test_score
74	{'model__eval_metric': 'logloss', 'model__learning_rate': 0.2, 'model__max_depth': 5, 'model__n_estimators': 100, 'model__verbosity': 2}	0.700139	0.000667	1
122	{'model__eval_metric': 'auc', 'model__learning_rate': 0.2, 'model__max_depth': 5, 'model__n_estimators': 100, 'model__verbosity': 2}	0.700139	0.000667	1
26	{'model__eval_metric': 'merror', 'model__learning_rate': 0.2, 'model__max_depth': 5, 'model__n_estimators': 100, 'model__verbosity': 2}	0.700139	0.000667	1
85	{'model__eval_metric': 'logloss', 'model__learning_rate': 0.3, 'model__max_depth': 5, 'model__n_estimators': 80, 'model__verbosity': 2}	0.700065	0.000583	4
37	{'model__eval_metric': 'merror', 'model__learning_rate': 0.3, 'model__max_depth': 5, 'model__n_estimators': 80, 'model__verbosity': 2}	0.700065	0.000583	4

The model improved a little bit. The difference in evaluation metrics didn't affect the model performance in this case. The best parameters to be used: learning rate 0.2, max depth 5, n estimators 100. The performance in test dataset was also increased. The details:

Type	Train	Test
Pre-tuning	0.6970	0.7018
Post-tuning	0.7001	0.7024

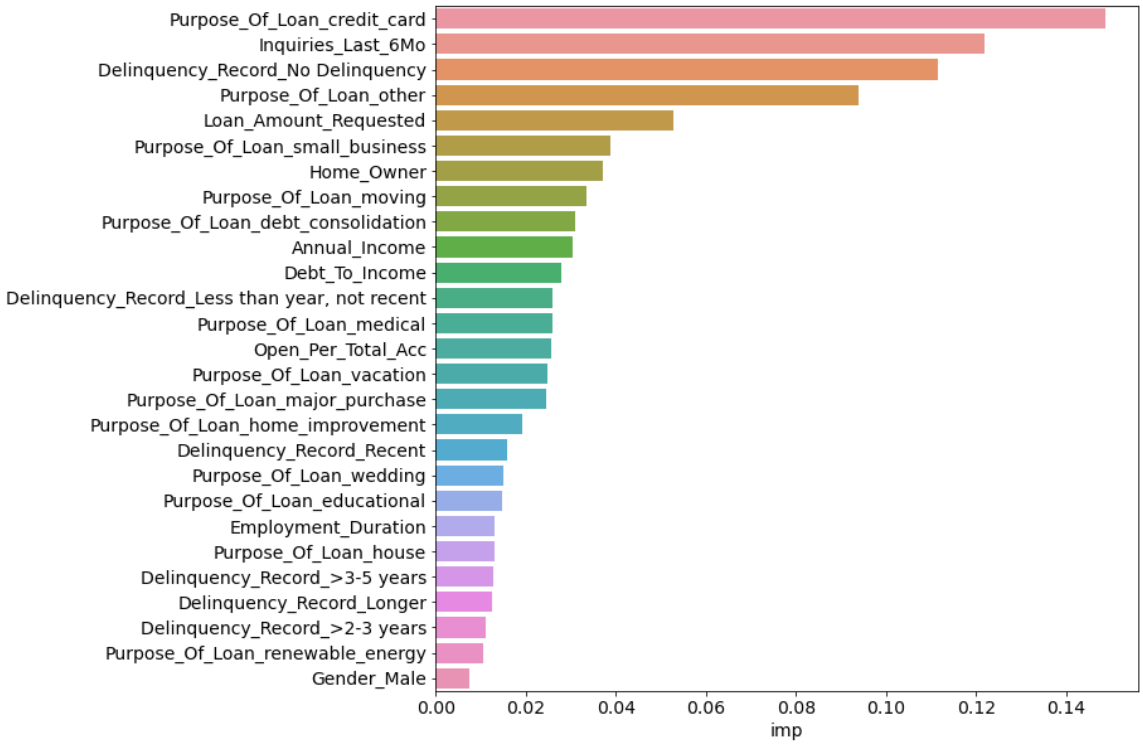


The detailed ROC AUC OvO values:

```
1 vs 2 ROC AUC OvO  : 0.683909934488961
2 vs 1 ROC AUC OvO  : 0.543934266037736
1 vs 3 ROC AUC OvO  : 0.821652202736275
3 vs 1 ROC AUC OvO  : 0.819155021007967
2 vs 3 ROC AUC OvO  : 0.647833195748444
3 vs 2 ROC AUC OvO  : 0.698492815870393
Average ROC AUC OvO : 0.702496239314963
```

From this model improvement exploration, I believe that new distinguishing features needs to be introduced in the next model. The features introduced must have the information that will distinguish class 2 and the others.

9. Features Importance



10. Conclusion

I acquired a prediction model using XGBoost classifier with average value of ROC-AUC OvO of approximately 70% which is barely acceptable.

I tried to use resampling methods and hyperparameter tuning to improve the model. However, the model wasn't improved significantly. Looking back at the feature selection and engineering process, I already maximize the information needed to be included in model building, by eliminating 1 column only which is income verified column, recategorizing length employed column into employment duration column, and merging total accounts column with number open accounts due to multicollinearity. Thus, I believe that the problem with this model is that interest rate 2 was placed in an awkward position between class 1 and class 3, thus making it undistinguishable to class 1 or class 3.

To improve the model further, new features needed to be introduced into model that will play a vital role in distinguishing interest rate 2 with the others.

11. Limitation

I believe that all of these processes need to be included in a pipeline for efficiency when deploying the model to new unseen data. However, due to lack of experience, I continue with the conventional way I learned from prior experience, especially from data science bootcamp.

12. References

The references used in this task can be found in my Jupyter Notebook file.