Credit Risk Model and Credit Scorecard

Problem Setting:

This project focuses on creating a data-driven credit risk model using Python. The goal is to predict default probabilities (PD) and assign credit scores to borrowers. Utilizing an interpretable scorecard, the project simplifies the credit score calculation process. The objective is the step-by-step development of an accessible and effective credit risk model, enhancing understanding and predictive capabilities.

Dataset and Data Overview:

Dataset: Credit_Modeling_Scoring

We will use a dataset made available on Kaggle that relates to consumer loans issued by the Lending Club, a US P2P lender. The raw data includes information on over 450,000 consumer loans issued between 2007 and 2014 with almost 75 features, including the current loan status and various attributes related to both borrowers and their payment behavior. Refer to the data dictionary for further details on each column.

Initial data exploration reveals the following:

- 18 features with more than 80% of missing values. Given the high proportion of missing values, any technique to impute them will most likely result in inaccurate results.
- Certain static features not related to credit risk, e.g., id, member_id, url, title.
- Other forward-looking features that are expected to be populated only once the borrower has defaulted, e.g., recoveries, collection_recovery_fee. Since our objective here is to predict the future probability of default, having such features in our model will be counterintuitive, as these will not be observed until the default event has occurred.

We will drop all the above features.

Based on the data exploration, our target variable appears to be loan_status. A quick look at its unique values and their proportion thereof confirms the same.

```
# explore the unique values in loan_status column
loan_data['loan_status'].value_counts(normalize = True)
```

```
Current                                                0.480878
Fully Paid                                             0.396193
Charged Off                                            0.091092
Late (31-120 days)                                     0.014798
In Grace Period                                        0.006747
Does not meet the credit policy. Status:Fully Paid     0.004263
Late (16-30 days)                                      0.002612
Default                                                0.001784
Does not meet the credit policy. Status:Charged Off    0.001632
Name: loan_status, dtype: float64
```

Based on domain knowledge, we will classify loans with the following loan_status values as being in default (or 0):

- Charged Off
- Default
- Late (31–120 days)
- Does not meet the credit policy. Status:Charged Off

All the other values will be classified as good (or 1).

Data Split:

Let us now split our data into the following sets: training (80%) and test (20%). We will perform Repeated Stratified k Fold testing on the training test to preliminary evaluate our model while the test set will remain untouched till final model evaluation. This approach follows the best model evaluation practice.

Data Cleaning:

Next comes some necessary data cleaning tasks as follows:

- Remove text from the emp_length column (e.g., years) and convert it to numeric
- For all columns with dates: convert them to Python's datetime format, create a new column as a difference between model development date and the respective date feature and then drop the original feature
- Remove text from the term column and convert it to numeric

We will define helper functions for each of the above tasks and apply them to the training dataset. Having these helper functions will assist us with performing these same tasks again on the test dataset without repeating our code.

Feature Selection:

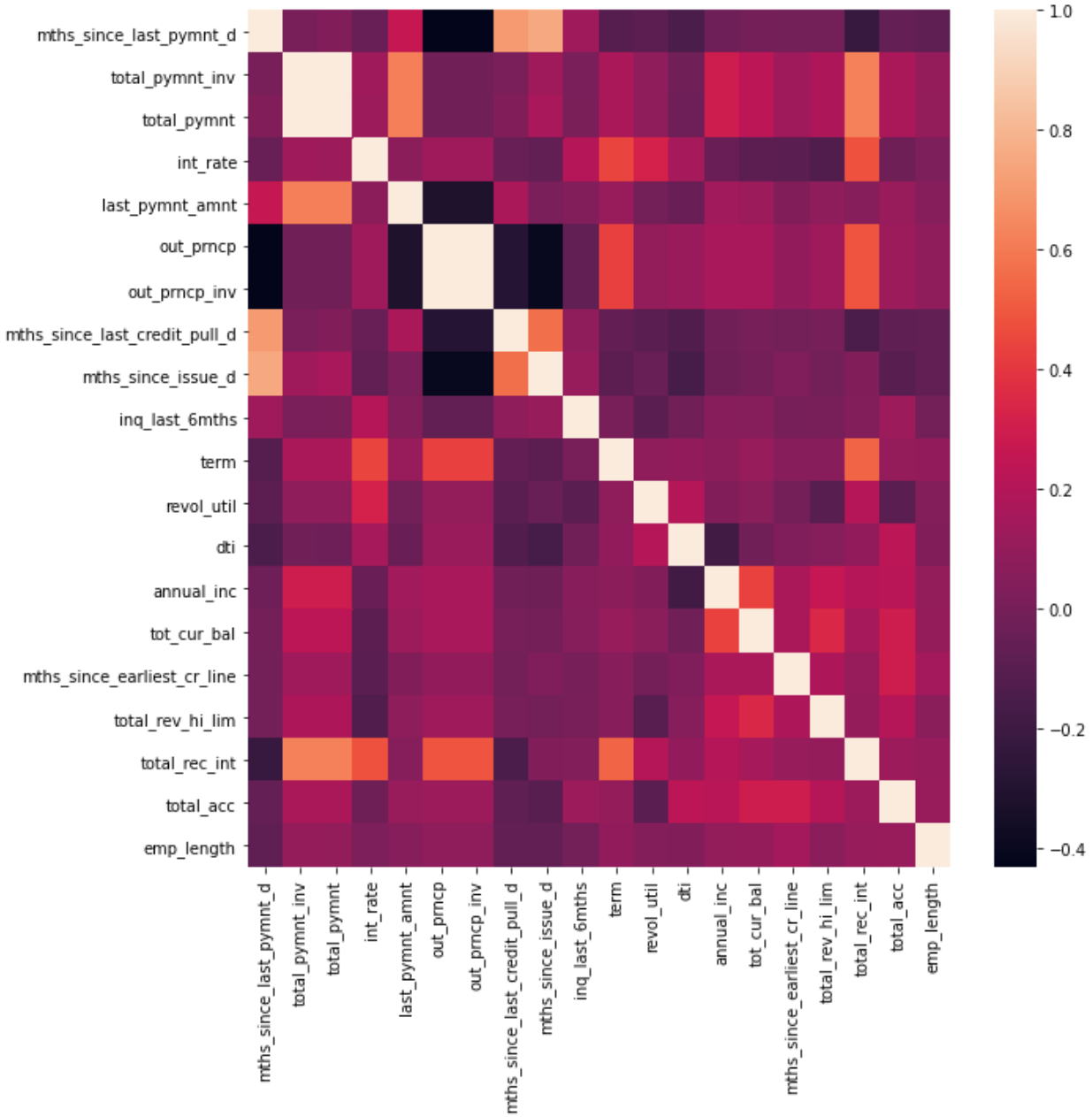
Next up, we will perform feature selection to identify the most suitable features for our binary classification problem using the Chi-squared test for categorical features and ANOVA F-statistic for numerical features. The p-values, in ascending order, from our Chi-squared test on the categorical features are as below:

	Feature	p-value
0	grade	0.000000
1	home_ownership	0.000000
2	verification_status	0.000000
3	purpose	0.000000
4	addr_state	0.000000
5	initial_list_status	0.000000
6	pymnt_plan	0.000923
7	application_type	1.000000

For the sake of simplicity, we will only retain the top four features and drop the rest.

The ANOVA F-statistic for 34 numeric features shows a wide range of F values, from 23,513 to 0.39. We will keep the top 20 features and potentially come back to select more in case our model evaluation results are not reasonable enough.

Next, we will calculate the pair-wise correlations of the selected top 20 numerical features to detect any potentially multicollinear variables. A heat-map of these pair-wise correlations identifies two features (out_prncp_inv and total_pymnt_inv) as highly correlated. Therefore, we will drop them also for our model.



Next, we will simply save all the features to be dropped in a list and define a function to drop them.

One-Hot Encoding and Update Test Dataset:

Next, we will create dummy variables of the four final categorical variables and update the test dataset through all the functions applied so far to the training dataset.

Note a couple of points regarding the way we create dummy variables:

- We will use a particular naming convention for all variables: original variable name, colon, category name
- Generally speaking, in order to avoid multicollinearity, one of the dummy variables is dropped through the drop_first parameter of pd.get_dummies. However, we will not do so at this stage as we require all the dummy variables to calculate the Weight of Evidence (WoE) and Information Values (IV) of our categories — more on this later. We will drop one dummy variable for each category later on
- We will also not create the dummy variables directly in our training data, as doing so would drop the categorical variable, which we require for WoE calculations. Therefore, we will create a new dataframe of dummy variables and then concatenate it to the original training/test dataframe.

Next up, we will update the test dataset by passing it through all the functions defined so far. Pay special attention to reindexing the updated test dataset after creating dummy variables. Let me explain this by a practical example.

Consider a categorical feature called grade with the following unique values in the pre-split data: A, B, C, and D. Suppose that the proportion of D is very low, and due to the random nature of train/test split, none of the observations with D in the grade category is selected in the test set. Therefore, grade's dummy variables in the training data will be grade:A, grade:B, grade:C, and grade:D, but grade:D will not be created as a dummy variable in the test set. We will be unable to apply a fitted model on the test set to make predictions, given the absence of a feature expected to be present by the model. Therefore, we reindex the test set to ensure that it has the same columns as the training data, with any missing columns being added with 0 values. A 0 value is pretty intuitive since that category will never be observed in any of the test samples.

WoE Binning and Feature Engineering:

Creating new categorical features for all numerical and categorical variables based on WoE is one of the most critical steps before developing a credit risk model, and also quite time-consuming. There are specific custom Python packages and functions available on GitHub and elsewhere to perform this exercise. However, I prefer to do it manually as it allows me a bit more flexibility and control over the process.

But What is WoE and IV?

Weight of Evidence (WoE) and Information Value (IV) are used for feature engineering and selection and are extensively used in the credit scoring domain.

WoE is a measure of the predictive power of an independent variable in relation to the target variable. It measures the extent a specific feature can differentiate between target classes, in our case: good and bad customers.

IV assists with ranking our features based on their relative importance.

According to Baesens et al.[1] and Siddiqi[2], WOE and IV analyses enable one to:

- Consider each variable's independent contribution to the outcome
- Detect linear and non-linear relationships
- Rank variables in terms of its univariate predictive strength
- Visualize the correlations between the variables and the binary outcome
- Seamlessly compare the strength of continuous and categorical variables without creating dummy variables
- Seamlessly handle missing values without imputation. (Note that we have not imputed any missing values so far, this is the reason why. Missing values will be assigned a separate category during the WoE feature engineering step)
- Assess the predictive power of missing values

Weight of Evidence (WoE):

The formula to calculate WoE is as follow:

$$WoE = \ln\left(\frac{\%\ of\ good\ customers}{\%\ of\ bad\ customers}\right)$$

A positive WoE means that the proportion of good customers is more than that of bad customers and vice versa for a negative WoE value.

Information Value (IV):

IV is calculated as follows:

$$IV = \sum(\%\ of\ good\ customers - \%\ of\ bad\ customers) \times WoE$$

According to Siddiqi[2], by convention, the values of IV in credit scoring is interpreted as follows:

Information Value	Variable Predictiveness
Less than 0.02	Not useful for prediction
0.02 to 0.1	Weak predictive Power
0.1 to 0.3	Medium predictive Power
0.3 to 0.5	Strong predictive Power
>0.5	Suspicious Predictive Power

IV is only useful as a feature selection and importance technique when using a binary logistic regression model.

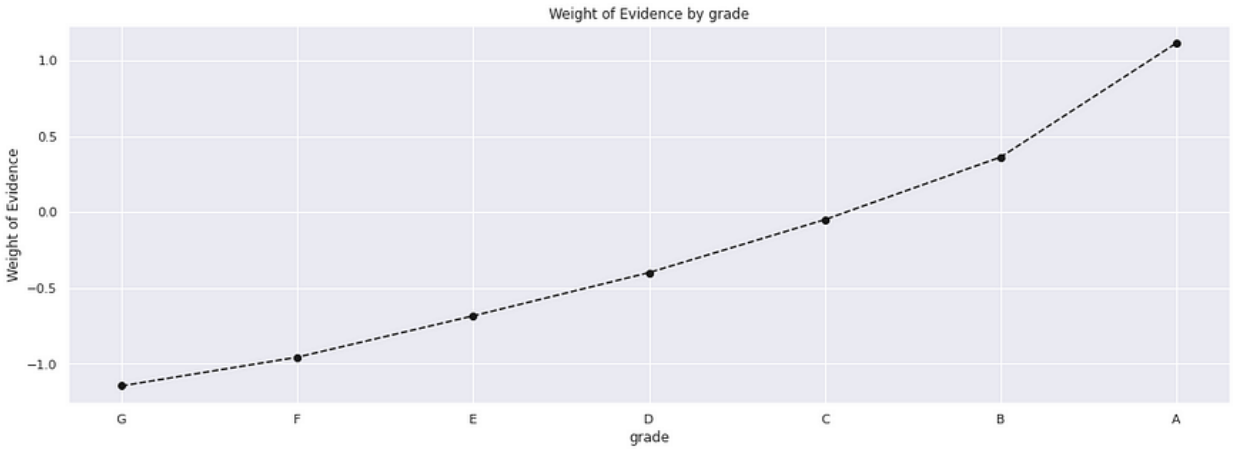
WoE Feature Engineering and IV Calculation for our Data:

Enough with the theory, let's now calculate WoE and IV for our training data and perform the required feature engineering. We will define three functions as follows, each one to:

- calculate and display WoE and IV values for categorical variables
- calculate and display WoE and IV values for numerical variables
- plot the WoE values against the bins to help us in visualizing WoE and combining similar WoE bins

Sample output of these two functions when applied to a categorical feature, grade, is shown below:

	grade	n_obs	prop_good	prop_n_obs	n_good	n_bad	prop_n_good	prop_n_bad	WoE	diff_prop_good	diff_WoE	IV
0	G	2623	0.721693	0.007032	1893.0	730.0	0.005697	0.017904	-1.144981	NaN	NaN	0.292145
1	F	10606	0.758061	0.028432	8040.0	2566.0	0.024198	0.062932	-0.955774	0.036369	0.189207	0.292145
2	E	28590	0.804477	0.076643	23000.0	5590.0	0.069224	0.137097	-0.683340	0.046416	0.272434	0.292145
3	D	61713	0.845527	0.165438	52180.0	9533.0	0.157049	0.233801	-0.397915	0.041050	0.285425	0.292145
4	C	100342	0.885870	0.268993	88890.0	11452.0	0.267536	0.280865	-0.048620	0.040343	0.349295	0.292145
5	B	109344	0.921422	0.293125	100752.0	8592.0	0.303238	0.210723	0.363975	0.035552	0.412595	0.292145
6	A	59810	0.961361	0.160336	57499.0	2311.0	0.173057	0.056678	1.116232	0.039939	0.752257	0.292145



Once we have calculated and visualized WoE and IV values, next comes the most tedious task to select which bins to combine and whether to drop any feature given its IV. The shortlisted features that we are left with until this point will be treated in one of the following ways:

- There is no need to combine WoE bins or create a separate missing category given the discrete and monotonic WoE and absence of any missing values: grade, verification_status, term
- Combine WoE bins with very low observations with the neighboring bin: home_ownership, purpose

- Combine WoE bins with similar WoE values together, potentially with a separate missing
  category: int_rate, annual_inc, dti, inq_last_6mths, revol_util, out_prncp, total_pymnt, total_rec_int, total_rev_hi_lim, mths_since_earliest_cr_line, mths_since_issue_d, mths_since_last_credit_pull_d
- Ignore features with a low or very high IV
  value: emp_length, total_acc, last_pymnt_amnt, tot_cur_bal, mths_since_last_pymnt_d_factor

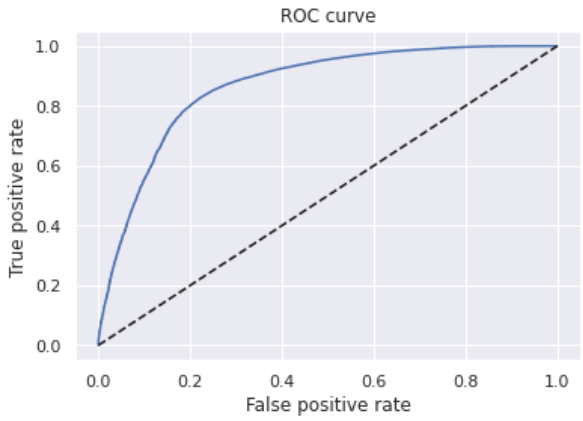
Note that for certain numerical features with outliers, we will calculate and plot WoE after excluding them that will be assigned to a separate category of their own.

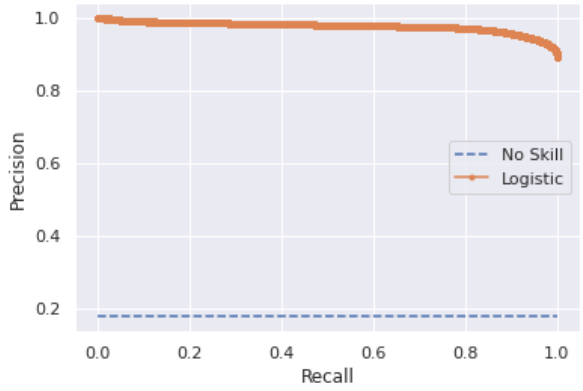
Model Implementation:

Finally, we come to the stage where some actual machine learning is involved. We will fit a logistic regression model on our training set and evaluate it using RepeatedStratifiedKFold. Note that we have defined the class_weight parameter of the LogisticRegression class to be balanced. This will force the logistic regression model to learn the model coefficients using cost-sensitive learning, i.e., penalize false negatives more than false positives during model training.

Model Evaluation:

We will draw a ROC curve, PR curve, and calculate AUROC and Gini. Our AUROC on test set comes out to 0.866 with a Gini of 0.732, both being considered as quite acceptable evaluation scores. Our ROC and PR curves will be something like this:





Scorecard Development:

The final piece of our puzzle is creating a simple, easy-to-use, and implement credit risk scorecard that can be used by any layperson to calculate an individual's credit score given certain required information about him and his credit history.

Remember the summary table created during the model training phase? We will append all the reference categories that we left out from our model to it, with a coefficient value of 0, together with another column for the original feature name (e.g., grade to represent grade:A, grade:B, etc.).

We will then determine the minimum and maximum scores that our scorecard should spit out. As a starting point, we will use the same range of scores used by FICO: from 300 to 850.

The coefficients returned by the logistic regression model for each feature category are then scaled to our range of credit scores through simple arithmetic. An additional step here is to update the model intercept's credit score through further scaling that will then be used as the starting point of each scoring calculation.

At this stage, our scorecard will look like this (the Score-Preliminary column is a simple rounding of the calculated scores):

Feature name	Coefficients	Original feature name	Score - Calculation	Score - Preliminary
Intercept	2.948892	Intercept	598.515609	599.0
grade:A	0.980200	grade	24.463996	24.0
grade:B	0.792926	grade	19.789993	20.0
grade:C	0.609305	grade	15.207145	15.0
grade:D	0.487386	grade	12.164269	12.0
grade:E	0.331929	grade	8.284354	8.0
grade:F	0.190615	grade	4.757402	5.0
home_ownership:OWN	-0.048374	home_ownership	-1.207327	-1.0
home_ownership:OTHER_NONE_RENT	-0.104195	home_ownership	-2.600515	-3.0
verification_status:Source Verified	-0.281427	verification_status	-7.023912	-7.0
verification_status:Verified	-0.459417	verification_status	-11.466206	-11.0
purpose:debt_consolidation	-0.315579	purpose	-7.876281	-8.0
purpose:credit_card	-0.222064	purpose	-5.542316	-6.0
purpose:educ__ren_en__sm_b__mov	-0.481318	purpose	-12.012820	-12.0
purpose:vacation__house__wedding__med__oth	0.003981	purpose	0.099355	0.0
term:36	-0.094122	term	-2.349118	-2.0
int_rate:<7.071	0.955408	int_rate	23.845255	24.0
int_rate:7.071-10.374	0.285426	int_rate	7.123724	7.0
int_rate:10.374-13.676	0.056631	int_rate	1.413405	1.0
int_rate:13.676-15.74	0.039281	int_rate	0.980393	1.0

Depending on your circumstances, you may have to manually adjust the Score for a random category to ensure that the minimum and maximum possible scores for any given situation remains 300 and 850. Some trial and error will be involved here.

Calculate Credit Scores for Test Set:

Once we have our final scorecard, we are ready to calculate credit scores for all the observations in our test set. Remember, our training and test sets are a simple collection of dummy variables with 1s and 0s representing whether an observation belongs to a specific dummy variable. For example, in the image below, observation 395346 had a C grade, owns its own home, and its verification status was Source Verified.

	Intercept	grade:A	grade:B	grade:C	grade:D	grade:E	grade:F	home_ownership:OWN	home_ownership:OTHER_NONE_RENT	verification_status:Source Verified	verification_status:Verified
395346	1	0	0	1	0	0	0	1	0	1	0
376583	1	1	0	0	0	0	0	0	1	0	0
297790	1	0	0	1	0	0	0	0	1	1	0
47347	1	0	1	0	0	0	0	0	1	0	1
446772	1	0	0	0	1	0	0	0	0	1	0

Accordingly, after making certain adjustments to our test set, the credit scores are calculated as a simple matrix dot multiplication between the test set and the final score for each category. Consider the above observations together with the following final scores for the intercept and grade categories from our scorecard:

Feature name	Final Score
Intercept	598
grade:A	24
grade:B	20
grade:C	15
grade:D	12
grade:E	8
grade:F	5

Intuitively, observation 395346 will start with the intercept score of 598 and receive 15 additional points for being in the grade:C category. Similarly, observation 3766583 will be assigned a score of 598 plus 24 for being in the grade:A category. We will automate these calculations across all feature categories using matrix dot multiplication. The final credit score is then a simple sum of individual scores of each feature category applicable for an observation.

Setting Loan Approval Cut-offs:

At first, this ideal threshold appears to be counterintuitive compared to a more intuitive probability threshold of 0.5. But we used the class_weight parameter when fitting the logistic regression model that would have penalized false negatives more than false positives.

We then calculate the scaled score at this threshold point. As shown in the code example below, we can also calculate the credit scores and expected approval and rejection rates at each threshold from the ROC curve. This can help the business to further manually tweak the score cut-off based on their requirements.