**End-to-End Guide to Building a Credit Scorecard Using Machine Learning**

**Open-Source Tools for Real-World Problems Series**

I am opening up a new series called **Open-Source Tools for Real-World Problems.**

In this series, I will first review an open-source tool and then show how to apply it to real-world problems. During this process, I will show all the coding, and list all terms, theorems, and algorithms you need to know.

**Potential Audience**:

* **Students** who want to put a thorough project into the Resume, and try to get more interviews
* **Data Scientists/ML engineers** who want to build a scorecard and push into production

In this blog, you will learn:

* Build a **scorecard** using **machine learning** with Python
* Skillset: **Logistic Regression**, **Gradient Boosting**, **Weight of Evidence (WOE)**, **Information Value (IV)**, **Binning**, **Chi-square Binning**

Building a credit scorecard is a very typical industry-level problem, such as

1. evaluating a **transaction** or **customer’s credibility** to perform further actions such as **issuing a credit card** or **giving a balance transfer offer** for **high-credit customers** in a credit card company,
2. **giving promotions or premium rights** to **high-value customers** in an e-commerce platform,
3. providing **good customer segmentation** to reach the **right people** in a marketing firm.

You need to build a system to **score customers**, and it has to be explainable to **non-tech people**, because when something goes wrong (False Alarm), you will know how to explain it to the manager/customer/business side.

**Introduction:**

The code and data for this blog are available here:

**[GitHub - BruceYanghy/End-to-End-Guide-to-Building-a-Credit-Scorecard-Using-Machine-Learning…](https://github.com/BruceYanghy/End-to-End-Guide-to-Building-a-Credit-Scorecard-Using-Machine-Learning" \t "_blank)**

**[You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or…](https://github.com/BruceYanghy/End-to-End-Guide-to-Building-a-Credit-Scorecard-Using-Machine-Learning" \t "_blank)**

[github.com](https://github.com/BruceYanghy/End-to-End-Guide-to-Building-a-Credit-Scorecard-Using-Machine-Learning" \t "_blank)

**Open Source Tool:** [**Toad**](https://github.com/amphibian-dev/toad)

Toad is a production-to-go library for building scorecards; it offers EDA, feature engineering, and scorecard generation. Its key functionality streamlines the most critical and time-consuming processes such as feature selection and fine binning.

**Dataset: Default of Credit Card Clients Dataset**

The description can be found on [**Kaggle**](https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset). The original dataset can be found at [UCI](https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients).

There are 25 variables:

* **ID**: the customer ID
* **23 features include**: limit balance, sex, education, marriage, age, repayment, amount of bill statement, amount of previous payment
* **Label**: Default payment (1=yes, 0=no)

**Assuming** that you are **a data scientist at a credit card company**, in order to promote credit card usage, your manager told you to **build a scorecard system for existing customers, to send them some balance transfer offers**. **Let’s say 6 months with 0% APR + 3% fees.**

You have **historical data** about these customers. You don’t want to send to the ones who constantly **default** because they will probably take the money and run away. Your company will **not be able to collect these transfer balances** back after 6 months, and this will become **a loss for your company**. You may also need to hire a collection agency to collect these debts back. That isn’t good.

So everything depends on your scorecard system!

Let’s begin.

**1. Data Preprocessing**

In real life you may need to fetch and get the raw data from your company’s database using SQL-like query; this may take you a lot of time.

In this blog, we will just use the CSV file.

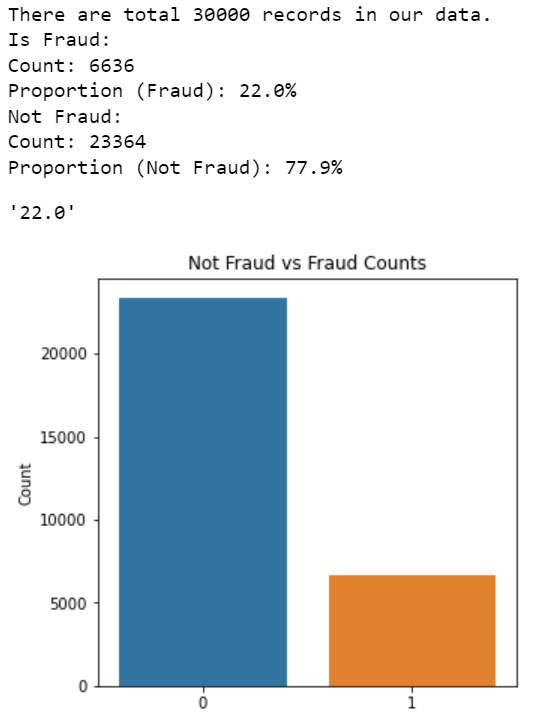
**Install and import packages**

import pkg\_resources  
import pip  
installedPackages = {pkg.key for pkg in pkg\_resources.working\_set}  
required = { 'pandas','numpy', 'matplotlib', 'seaborn','toad','pickle','sklearn'}  
missing = required - installedPackages  
if missing:  
 !pip install pandas  
 !pip install numpy  
 !pip install matplotlib  
 !pip install seaborn  
 !pip install toad  
 !pip install pickle  
 !pip install sklearn  
  
import pandas as pd  
from sklearn.metrics import roc\_auc\_score,roc\_curve,auc,precision\_recall\_curve  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression   
from sklearn.ensemble import GradientBoostingClassifier  
  
import numpy as np  
import glob  
import math  
import seaborn as sns   
import matplotlib.pyplot as plt  
import toad  
import pickle

**Load data and check the default label rate**

# use pandas to load the csv file  
data = pd.read\_csv('UCI\_Credit\_Card.csv')  
# check the size of the data  
data.shape  
# check few lines  
data.head()  
#use the world 'label'  
data['label']=data['default.payment.next.month']  
data=data.drop(columns=['default.payment.next.month'])  
#check the fraud proportion of the data  
target\_info(data['label'])  
# set an exclude list for the scorecard package Toad  
exclude\_list = ['ID','label']

**22% fraud rate** (default people), that’s a pretty high default rate in the historical data.

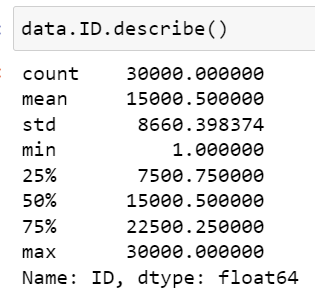


Copyright by AI4Finance-Education

**Train & Test Split**

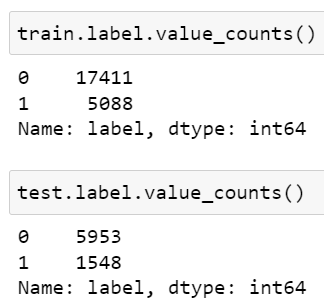
In a real-life project, you may want to split the train and test by date/time, it’s not good to do a random split, because it will break the time series and may cause overfitting.

We will use the user ID as the time to do the splitting.



Copyright by AI4Finance-Education

# use the ID column to split the train-test data  
data.ID.describe()  
train = data\_split(data,start = 0, end=22500,date\_col='ID')  
test = data\_split(data,start = 22500, end=172792,date\_col='ID')

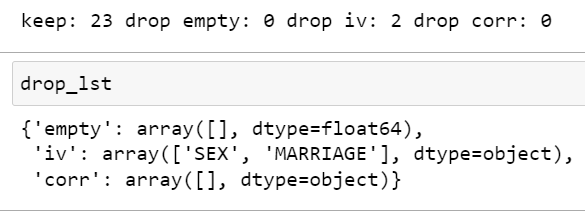


Copyright by AI4Finance-Education

**2. Feature Filtering**

First, we need to perform feature filtering to drop the features that have low information value and high correlation.

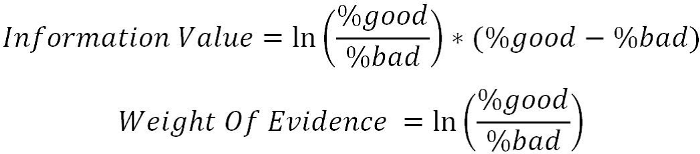
###feature filtering by missing value, IV & corrrelation：  
##If the missing value rate is greater than the threshold, delete the feature  
##If the correlation coefficient is greater than the threshold, delete the feature  
##If the IV is smaller than the threshold, delete the features  
  
train\_selected, drop\_lst= toad.selection.select(frame = train,  
 target=train['label'],   
 empty = 0.7,   
 iv = 0.02, corr = 1,   
 return\_drop=True,   
 exclude=exclude\_list)  
print("keep:",train\_selected.shape[1],  
 "drop empty:",len(drop\_lst['empty']),  
 "drop iv:",len(drop\_lst['iv']),  
 "drop corr:",len(drop\_lst['corr']))



Copyright by AI4Finance-Education

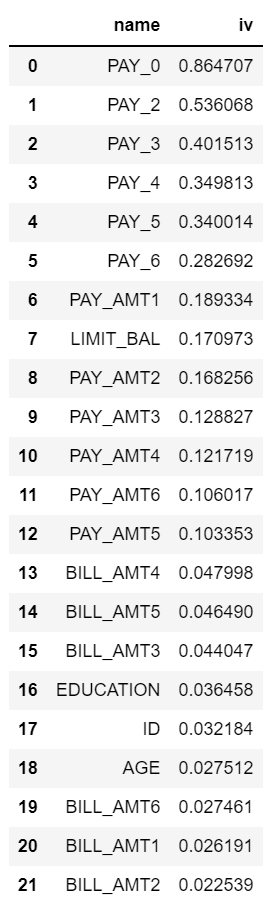
So we will keep 23 features in total, and drop 2 features **[‘SEX’, ‘MARRIAGE’]** because they have **low IV**.

* **Weight of evidence (WOE)** — describes the **relationship** between a predictive variable and a binary target variable
* **Information Value (IV)** — measures the strength of that relationship based on WOE. The industry level is to drop features with an **IV lower than 0.02**



# output the iv table to a dataframe  
def output\_iv\_importance(train\_selected,label\_col):  
 feat\_import\_iv = toad.quality(train\_selected,label\_col,iv\_only=True)  
 feat\_import\_iv=feat\_import\_iv['iv']  
 feat\_import\_iv = feat\_import\_iv.reset\_index()  
 feat\_import\_iv.columns = ['name','iv']  
 return feat\_import\_iv  
  
df\_iv=output\_iv\_importance(train\_selected,'label')  
df\_iv.head(30)

This is the IV ranking for all the features. We can see that **PAY\_0** has the highest IV, which makes sense because this **feature indicates the most recent repayment status**. EDUCATION and AGE have a low IV compared to payment status and amount.



Copyright by AI4Finance-Education

**3. Feature Binning**

Feature binning is to transform a continuous or numerical variable into a categorical feature.

**Advantages of Feature Binning**:

1. It **simplifies** the logistic regression model and **reduces the risk of model overfitting**
2. Logistic regression is a **generalized linear model**, and its expressive ability is limited; Feature binning can introduce **nonlinearity** into the model, which can improve the expressive ability of the model and help better model fitting
3. The **discretized features** are very **robust** to **abnormal data**: for example, the value of **a feature is 1 if age > 30, and 0 otherwise**. If the features are not discretized, an abnormal data point “300 years old” will impact the model fitting
4. It can treat null data as an **individual class**

**Steps for feature binning:**

**Step 1**. Initialization: c = toad.transform.Combiner()

**Step 2.** Training binning:

c.fit(dataframe,   
 y = 'target',   
 method = 'chi',   
 min\_samples = 0.05,   
 n\_bins = None,   
 empty\_separate = False)

* **y**: target column
* **method**: binning method, supports **chi** (chi-square binning), **dt** (decision tree binning), **kmean**, **quantile**, **step** (equal step size binning)
* **min\_samples**: Each box contains **the least number of samples**, which can be a **number** or a **proportion**
* **n\_bins**: the number of bins; If it is not possible to divide so many boxes the maximum number of bins will be divided.
* **empty\_separate**: Whether to separate **empty boxes** separately

**Step 3**. **check binning nodes**: **c.export()**

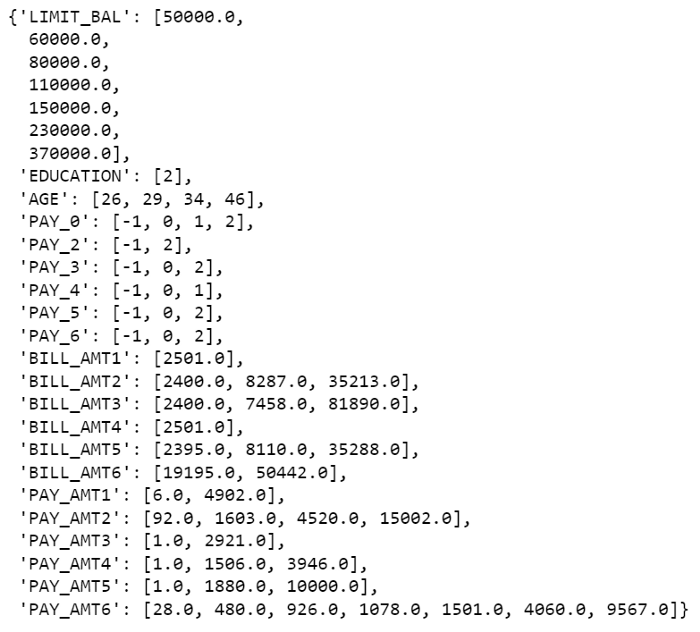
**Step 4**. **Manually adjust binning**: **c.load(dict)**

**Step 5**. **Apply binning results**: **c.transform(dataframe, labels=False)**

* **labels**: Whether to convert the binning results into box labels. If **False**, output **0, 1, 2**… (discrete variables are sorted according to the proportion), and if **True** output **(-inf, 0], (0,10], (10, inf)**.

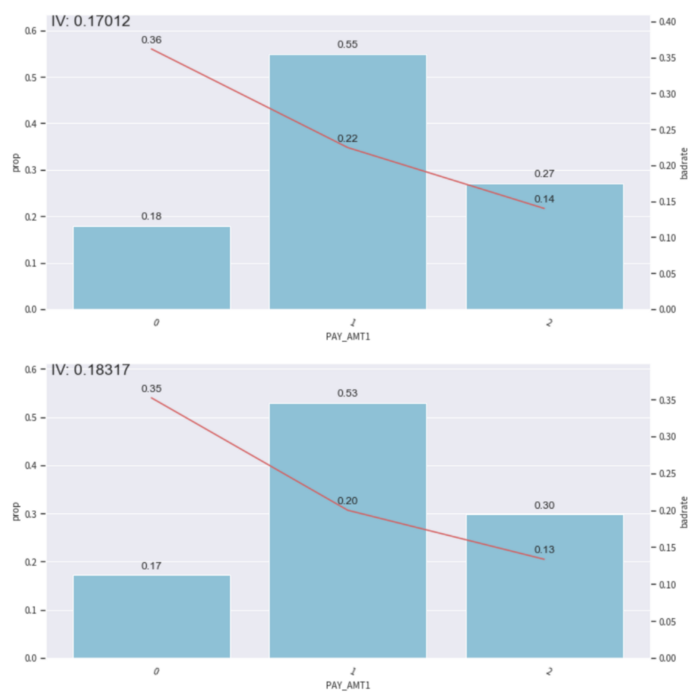
import time  
start = time.time()  
combiner = toad.transform.Combiner()  
# use the filtered features for training  
# Use the stable chi-square binning,   
# specifying that each bin has at least 5% data to ensure stability  
# empty values ​​will be automatically assigned to the best bin  
combiner.fit(X=train\_selected,  
 y=train\_selected['label'],  
 method='chi',  
 min\_samples = 0.05,  
 exclude=exclude\_list)  
end = time.time()  
print((end-start)/60)  
  
#output binning  
bins = combiner.export()

The binning result:



Copyright by AI4Finance-Education

#apply binning  
train\_selected\_bin = combiner.transform(train\_selected)  
test\_bin = combiner.transform(test[train\_selected\_bin.columns])  
  
#Fine tune bins  
from toad.plot import bin\_plot,badrate\_plot  
bin\_plot(train\_selected\_bin,x='PAY\_AMT1',target='label')  
bin\_plot(test\_bin,x='PAY\_AMT1',target='label')



Copyright by AI4Finance-Education

In this plot, the **bar plot** represents the **proportion of the data in the corresponding bin**; the **red line** represents the **proportion of default customers**.

We need to make sure that the binning has **monotonicity**, which means the **line is trending in the same direction** with **no sudden jump or drop**.

This plot looks ok, if there is a sudden jump or drop, we need to use **c.set\_rules(dict)** to combine the binning.

For example.

#setting rules  
rule = {'PAY\_AMT1':[['0', 'nan'],['1'], ['2'], ['3']]}  
  
#Adjust binning  
c.set\_rules(rule)

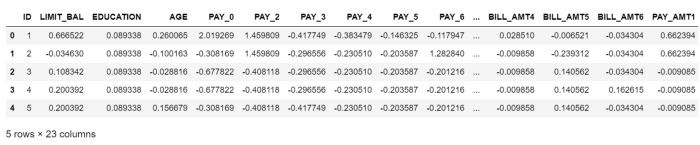
**4. Transform to WOE and Calculate PSI**

WOE transformation is performed after the binning is Done.

The steps are as follows:

1. Use the above-adjusted **Combiner c** to transform the data
2. **Initialize woe transform** t: **t= toad.transform.WOETransformer()**
3. **Training** the t: **t.fit\_transform** trains and outputs woe transformed data for the trainset
4. **target**: target column data (not column name)
5. **exclude**: columns that do not need to be transformed by WOE. Note: All columns will be transformed, including the columns that have not been binning, and the columns that do not need to be converted by WOE will be deleted through exclude, especially the target column.
6. **Transform the test/OOT data**: transer.transform

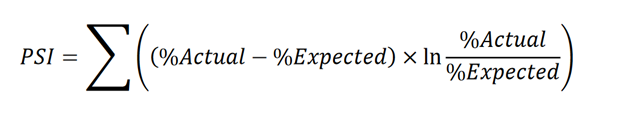
##transform to WOE  
t=toad.transform.WOETransformer()  
#transform training set  
train\_woe = t.fit\_transform(X=train\_selected\_bin,  
 y=train\_selected\_bin['label'],   
 exclude=exclude\_list)  
#transform testing set  
test\_woe = t.transform(test\_bin)  
  
final\_data\_woe = pd.concat([train\_woe,test\_woe])



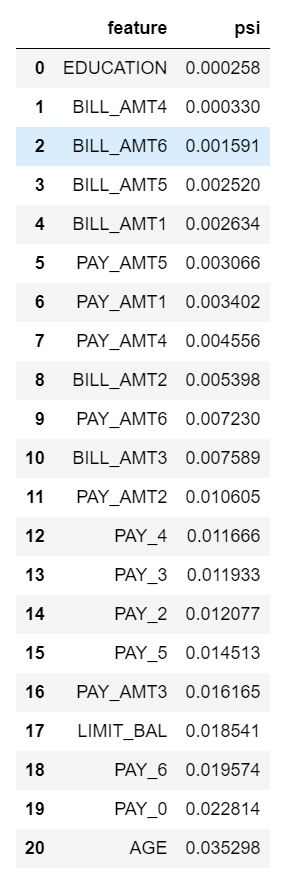
Copyright by AI4Finance-Education

**Calculate PSI**

PSI (Population Stability Index) reflects the **stability of the distribution**. We often use it to screen features and **evaluate model stability**. The industry level is to drop features with a PSI greater than **0.2**



#get the feature name  
features\_list = [feat for feat in train\_woe.columns if feat not in exclude\_list]  
#calculate PSI using toad  
psi\_df = toad.metrics.PSI(train\_woe[features\_list], test\_woe[features\_list]).sort\_values(0)  
#put into a dataframe  
psi\_df = psi\_df.reset\_index()  
psi\_df = psi\_df.rename(columns = {'index' : 'feature',0:'psi'})  
  
# features less than 0.25  
psi005 = list(psi\_df[psi\_df.psi<0.25].feature)  
# features geater than 0.25  
psi\_remove = list(psi\_df[psi\_df.psi>=0.25].feature)  
  
# keep exclude list  
for i in exclude\_list:  
 if i in psi005:  
 pass  
 else:  
 psi005.append(i)   
# remove features that are geater than 0.25  
train\_selected\_woe\_psi = train\_woe[psi005]  
off\_woe\_psi = test\_woe[psi005]  
  
# output our final data table  
final\_data\_woe = pd.concat([train\_selected\_woe\_psi,off\_woe\_psi])

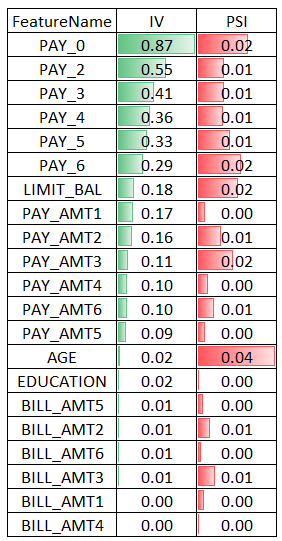


Copyright by AI4Finance-Education

**5. Output Final IV**

This step is to output the IV after the **WOE transformation**, it’s a little bit different than the raw features’ IV.

# output the IV  
features\_use = [feat for feat in final\_data\_woe.columns if feat not in exclude\_list]  
len(features\_use)  
  
df\_iv=output\_iv\_importance(final\_data\_woe[features\_use+['label']],'label')



Copyright by AI4Finance-Education

The idea is to get the features with the **highest IV and lowest PSI**.

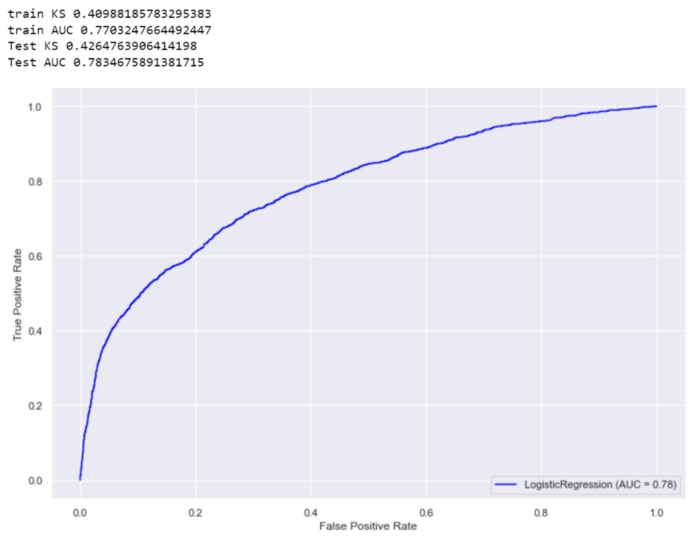
**6. Model Tuning**

**Logistic Regression**

The most used algorithm in the credit scorecard modeling process is Logistic Regression. The reasons are as follows:

* **Simple linear relationship**: the relationship between variables is a linear relationship
* **Good Interpretability**: the effect of input variables on target variables is readily available
* **Give probabilities instead of discriminative classes**: the customer’s **characteristic information** (such as marriage, age, historical credit performance, etc.) can be integrated and converted into a **probability value**, which provides an **intuitive** basis to predict whether the customer is good or bad. That is, the larger the value, the smaller the probability that the customer will default in the future.
* **Easy to deploy**: testing, deployment, monitoring, tuning, etc., are relatively simple

def check\_train\_test\_auc(x\_train,y\_train,x\_test,y\_test):  
 from sklearn.linear\_model import LogisticRegression  
 lr = LogisticRegression(random\_state=42,C= 0.1, penalty='l2', solver='newton-cg')  
  
   
 lr = LogisticRegression(class\_weight='balanced')  
 lr.fit(x\_train, y\_train)  
  
 pred\_train = lr.predict\_proba(x\_train)[:,1]  
 from toad.metrics import KS, AUC  
  
 print('train KS',KS(pred\_train, y\_train))  
 print('train AUC',AUC(pred\_train, y\_train))  
   
 pred\_OOT =lr.predict\_proba(x\_test)[:,1]  
 print('Test KS',KS(pred\_OOT, y\_test))  
 print('Test AUC',AUC(pred\_OOT, y\_test))  
   
 from sklearn.metrics import confusion\_matrix, accuracy\_score, roc\_auc\_score, plot\_roc\_curve, classification\_report  
  
 fig, ax = plt.subplots(figsize=(12, 8))  
 plot\_roc\_curve(lr, x\_test, y\_test, color='blue', ax=ax)  
  
#train & test  
check\_train\_test\_auc(x\_train = train\_woe[features\_use],y\_train=train\_woe['label'],  
 x\_test =test\_woe[features\_use] ,y\_test = test\_woe['label'])



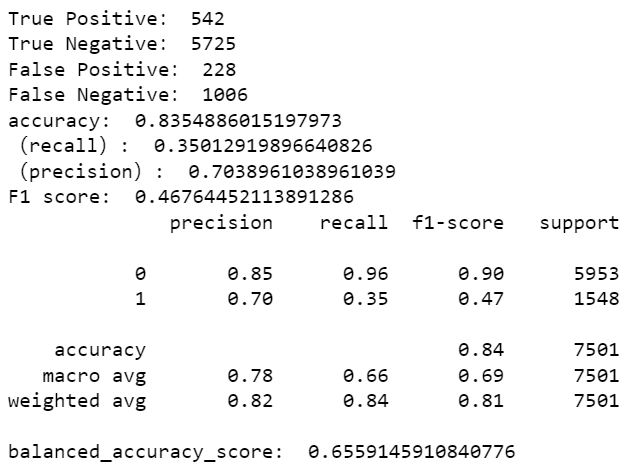
Copyright by AI4Finance-Education

We can see that there is **not a big difference** between the train AUC and test AUC or train KS and test KS. This means our model does not overfit.

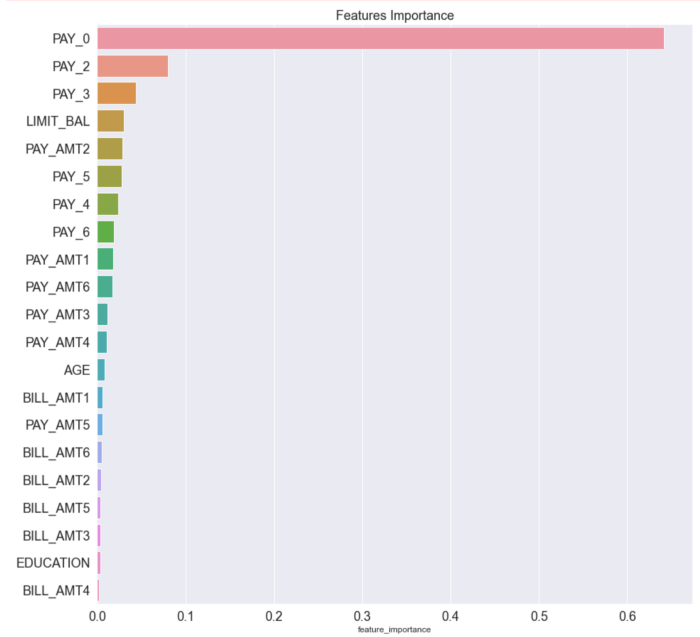
**Train a GradientBoostingClassifier and check the feature importance table**

To see if a GBDT model will perform better than LR and compare the feature importance table with IV.

def get\_evaluation\_scores(label, predictions):  
 from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score  
 from sklearn.metrics import balanced\_accuracy\_score  
 tp, fn, fp, tn = confusion\_matrix(label,predictions,labels=[1,0]).reshape(-1)  
 print('True Positive：',tp)  
 print('True Negative：',tn)  
 print('False Positive：',fp)  
 print('False Negative：',fn)  
 accuracy = (tp+tn)/(tp+fn+fp+tn)  
 print('accuracy: ',accuracy)  
 recall = tp/(tp+fn)  
 print('（recall）: ',recall)  
 precision = tp/(tp+fp)  
 print('（precision）: ',precision)  
 #f1 score = 2\*(P\*R)/(P+R)  
 f1 = 2\*precision\*recall/(precision+recall)  
 print('F1 score: ',f1)  
   
 print(classification\_report(label, predictions))  
   
 print('balanced\_accuracy\_score: ',balanced\_accuracy\_score(label,predictions))  
 return precision, recall  
  
def evaluate\_result(df\_train,df\_test,features\_name):  
 from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, RandomForestClassifier, ExtraTreesClassifier  
 import seaborn as sns  
 import matplotlib.pyplot as plt  
 start = time.time()  
 x\_train = df\_train[features\_name]  
 y\_train = df\_train['label']  
  
 x\_test = df\_test[features\_name]  
 y\_test = df\_test['label']  
  
 model = GradientBoostingClassifier(n\_estimators=250,random\_state=0)  
 model.fit(x\_train,y\_train)  
 predictions = model.predict(x\_test)  
 get\_evaluation\_scores(label = y\_test, predictions=predictions)  
 feat\_importances = pd.Series(model.feature\_importances\_, index=features\_name)  
 feat\_importances=pd.DataFrame(feat\_importances).reset\_index()  
 feat\_importances.columns=['feature\_name','feature\_importance']  
 feat\_importances=feat\_importances.sort\_values(['feature\_importance'],ascending=False)  
 import matplotlib.pyplot as plt  
 plt.figure(figsize=(15,15))  
  
 sns\_plot1=sns.barplot(feat\_importances.feature\_importance,feat\_importances.feature\_name,estimator=sum)  
 plt.title("Features Importance",size=18)  
 plt.ylabel('', size = 15)  
 plt.tick\_params(labelsize=18)  
 return feat\_importances,model,x\_train,y\_train,x\_test,y\_test  
  
fet\_importance\_GBDT\_reason,model,x\_train,y\_train,x\_test,y\_test = evaluate\_result(df\_train=train\_woe,  
 df\_test=test\_woe,  
 features\_name=features\_use)



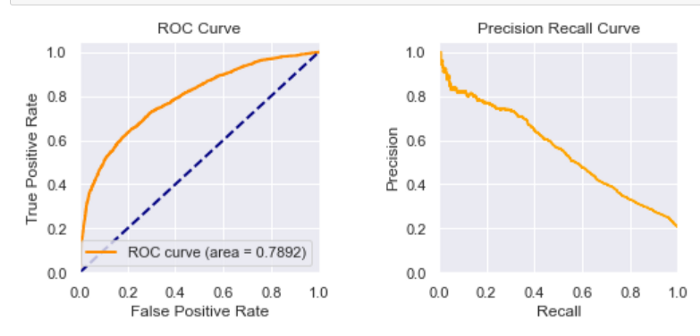
Copyright by AI4Finance-Foundation



Copyright by AI4Finance-Education

As we can see from the feature importance table, GBDT puts a lot of weight (64%) on the PAY\_0 feature.

def plot\_roc\_pre\_recall\_curve(labels, probs):  
 from sklearn.metrics import precision\_recall\_curve  
 # Get ROC curve FPR and TPR from true labels vs score values  
 fpr, tpr, \_ = roc\_curve(labels, probs)  
  
 # Calculate ROC Area Under the Curve (AUC) from FPR and TPR data points  
 roc\_auc = auc(fpr, tpr)  
  
 # Calculate precision and recall from true labels vs score values  
 precision, recall, \_ = precision\_recall\_curve(labels, probs)  
  
 plt.figure(figsize=(8, 3))  
  
 plt.subplot(1,2,1)  
 lw = 2  
 plt.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.4f)' % roc\_auc)  
 plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')  
 plt.xlim([0.0, 1.0])  
 plt.ylim([0.0, 1.05])  
 plt.xlabel('False Positive Rate')  
 plt.ylabel('True Positive Rate')  
 plt.title('ROC Curve')  
 plt.legend(loc="lower right")  
 plt.grid(True)  
  
 plt.subplot(1,2,2)  
 plt.step(recall, precision, color='orange', where='post')  
 # plt.fill\_between(recall, precision, step='post', alpha=0.5, color='orange')  
 plt.xlabel('Recall')  
 plt.ylabel('Precision')  
 plt.ylim([0.0, 1.05])  
 plt.xlim([0.0, 1.0])  
 plt.title('Precision Recall Curve')  
 plt.grid(True)  
  
 left = 0.125 # the left side of the subplots of the figure  
 right = 0.9 # the right side of the subplots of the figure  
 bottom = 0.1 # the bottom of the subplots of the figure   
 top = 0.9 # the top of the subplots of the figure  
 wspace = 0.5 # the amount of width reserved for blank space between subplots  
 hspace = 0.2 # the amount of height reserved for white space between subplots  
 plt.subplots\_adjust(left, bottom, right, top, wspace, hspace)  
 plt.show()  
  
probs = model.predict\_proba(x\_test)[:,1]  
sns.set(font\_scale = 1)  
plot\_roc\_pre\_recall\_curve(y\_test, probs)



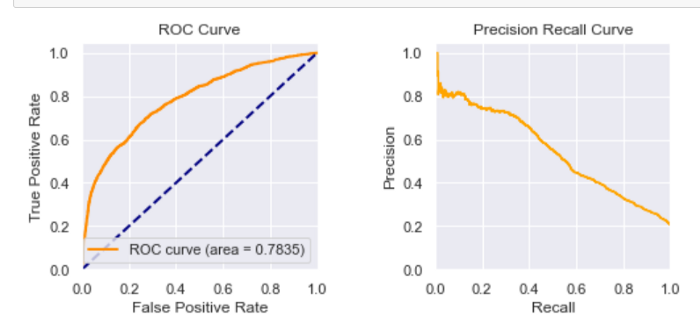
Copyright by AI4Finance-Education

The ROC and Precision-Recall curve look ok.

**7. Model Production**

Let’s train our production model for Logistic Regression

#prepare train & test data  
x\_train = train\_woe[features\_use]  
y\_train=train\_woe['label']  
x\_test =test\_woe[features\_use]   
y\_test = test\_woe['label']  
  
#Train LR  
#lr = LogisticRegression(random\_state=42,C= 0.1, penalty='l2', solver='newton-cg')  
lr = LogisticRegression(class\_weight = 'balanced')  
lr.fit(x\_train, y\_train)  
  
#check AUC  
probs = lr.predict\_proba(x\_test)[:,1]  
sns.set(font\_scale = 1)  
plot\_roc\_pre\_recall\_curve(y\_test, probs)



Copyright by AI4Finance-Education

AUC of LR: 0.7835

AUC of GBDT: 0.7892

Not a big difference between these models. So it’s ok to use LR to build a scorecard.

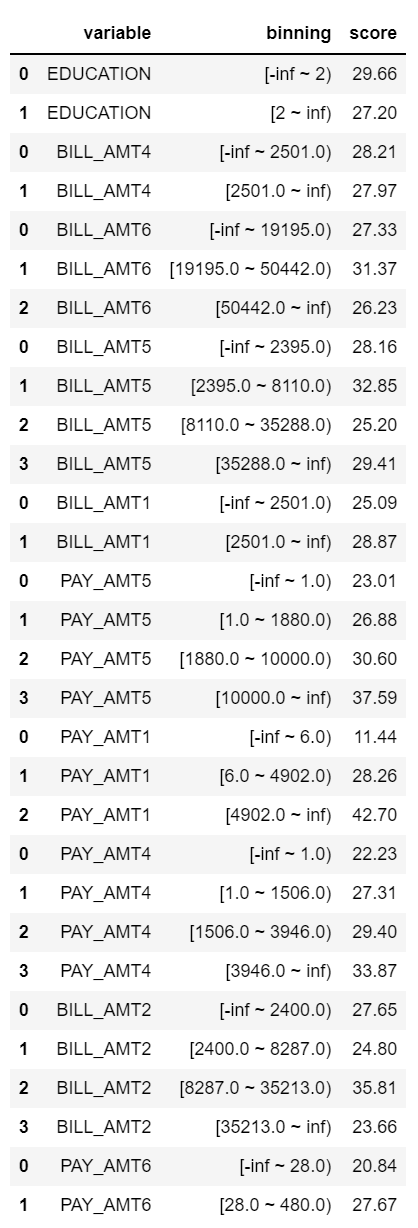
**8. Scorecard Tuning**

The following parameters are the most important ones for Scorecard Tuning.

* **base\_score** = 1000, **base\_odds** = 35 , **pdo** = 80, **rate** = 2

The actual meaning is that when the base odds are **35**, the **benchmark score is 1000**, and when the **ratio is twice the benchmark**, the benchmark score drops by 80 points.

# scorecard tuning  
card = toad.ScoreCard(  
 combiner = combiner,  
 transer = t,  
 class\_weight = 'balanced',  
 C=0.1,  
 base\_score = 1000,  
 base\_odds = 35 ,  
 pdo = 80,  
 rate = 2  
)  
  
card.fit(train\_woe[features\_use], train\_woe['label'])  
  
#inference on test data  
test['CreditScore'] = card.predict(test)  
test['CreditScore'].describe()  
  
#output the scorecard  
final\_card\_score=card.export()  
len(final\_card\_score)  
  
#transform the scorecard into dataframe and save to csv  
keys = list(card.export().keys())  
score\_card\_df = pd.DataFrame()  
for n in keys:  
 temp = pd.DataFrame.from\_dict(final\_card\_score[n], orient='index')  
 temp = temp.reset\_index()  
 temp.columns= ['binning','score']  
 temp['variable'] = n  
 temp = temp[['variable','binning','score']]  
 score\_card\_df=score\_card\_df.append(temp)  
score\_card\_df.head(30)



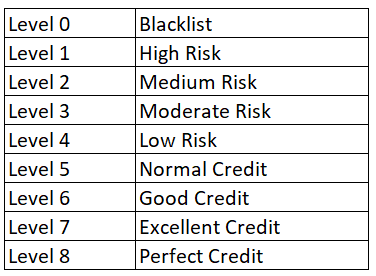
Copyright by AI4Finance-Education

Here we have our scorecard; once we get this table, we can throw this CSV file to the **developer** and let them develop a service to score each customer.

But there is something more we need to do; for a typical scorecard, we need to have a score range, and each range presents a level of trust.

In this way, the business side people are more easily to perform an action **against different level customers**.

For example, we can set the **credit level** like this:



Copyright by AI4Finance-Education

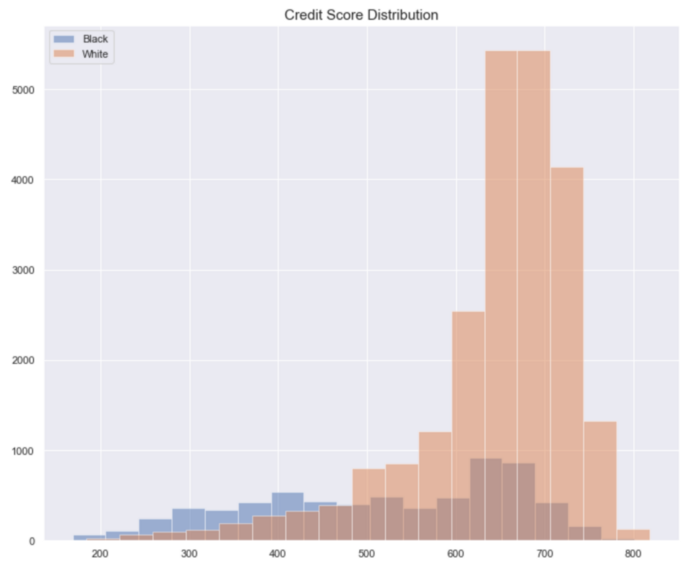
**9. Distribution Analysis**



Image by [lukechesser](https://unsplash.com/@lukechesser) on [Unsplash](https://unsplash.com/photos/JKUTrJ4vK00)

We need to plot the **score distribution** of default customers v.s. good customers in order to split the **credit levels** (level 0 to level 8).

plt.figure(figsize=(12,10))  
import random  
import numpy  
from matplotlib import pyplot as plt  
  
w = 40  
n = math.ceil((data['CreditScore'].max() - data['CreditScore'].min())/w)  
#bins = numpy.linspace(-10, 10, 100)  
  
plt.hist(data[data.label==1].CreditScore, alpha=0.5, label='Black',bins = n)  
plt.hist(data[data.label==0].CreditScore, alpha=0.5, label='White',bins = n)  
plt.legend(loc='upper left')  
plt.title('Credit Score Distribution: Test Set',size=15)  
plt.show()



Copyright by AI4Finance-Education

**Black** means **default customers**, and **white** means **good customers**.

A good model will clearly separate the **black/white distribution**.

The ideal distribution is **a smile shape**

* Good customers with high credit scores->to the very right
* Default customers with low credit scores-> to the very left.

**Our current model doesn’t separate these distributions very well**. For me, I would go back to explore **more features** to increase the **predictive power**. But we can always **build a baseline model first** and improve based on that.

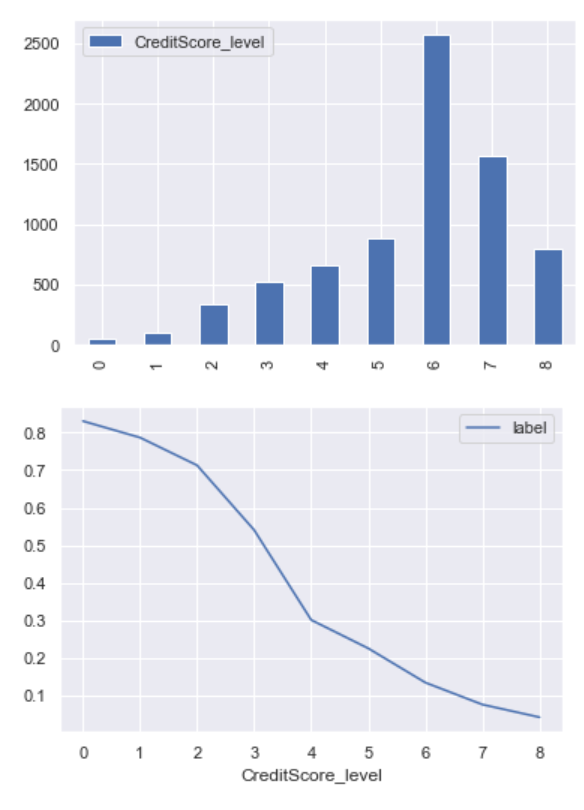
**10. Threshold Tuning**

We need to perform **threshold tuning for different credit levels**, and that’s a trade-off between loss & coverage.

Let’s say that your boss is OK with some loss, but you have to cover **70% of good customers** for next month.

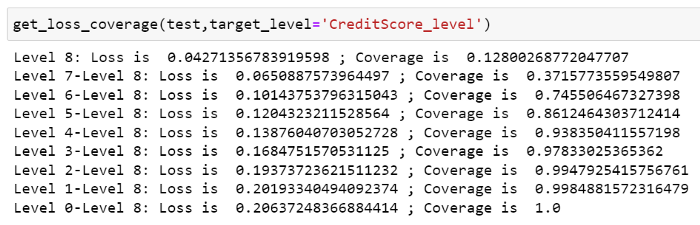
In other words, your goal is to find a threshold with **≤10% loss and ≥70% coverage**.

def get\_credit\_level(  
 test,  
 target\_score ='order\_score',  
 out\_col = 'order\_level',  
 left\_bound = -100,  
 level\_0 = 100,  
 level\_1 = 200,   
 level\_2 = 250,   
 level\_3 = 300,   
 level\_4 = 350,   
 level\_5 = 400,   
 level\_6 = 450,  
 level\_7 = 500,  
 level\_8 = 800):  
 level = []  
 for i in range(len(test)):  
 if (test[target\_score][i]>left\_bound) & (test[target\_score][i]<=level\_0):  
 level.append(0)  
 elif (test[target\_score][i]>level\_0) & (test[target\_score][i]<=level\_1):  
 level.append(1)  
 elif (test[target\_score][i]>level\_1) & (test[target\_score][i]<=level\_2):  
 level.append(2)  
 elif (test[target\_score][i]>level\_2) & (test[target\_score][i]<=level\_3):  
 level.append(3)  
 elif (test[target\_score][i]>level\_3) & (test[target\_score][i]<=level\_4):  
 level.append(4)  
 elif (test[target\_score][i]>level\_4) & (test[target\_score][i]<=level\_5):  
 level.append(5)  
 elif (test[target\_score][i]>level\_5) & (test[target\_score][i]<=level\_6):  
 level.append(6)  
 elif (test[target\_score][i]>level\_6) & (test[target\_score][i]<=level\_7):  
 level.append(7)  
 elif (test[target\_score][i]>level\_7 )& (test[target\_score][i]<=level\_8):  
 level.append(8)  
   
 test[out\_col] = level  
 return test  
  
def plot\_bts\_level\_loss(test, target\_col):  
 bts\_level\_df = test[target\_col].value\_counts()  
 bts\_level\_df=pd.DataFrame(bts\_level\_df)  
 df\_label\_level= test[test.label==1].groupby(target\_col)['label'].count()/ test.groupby(target\_col)['label'].count()  
 df\_label\_level = pd.DataFrame(df\_label\_level)  
 bts\_level\_df.sort\_index().plot.bar(title='')  
 df\_label\_level.plot()  
  
test = get\_credit\_level(test,  
 target\_score ='CreditScore',  
 out\_col = 'CreditScore\_level',  
 left\_bound = -1000,  
 level\_0 = 250,  
 level\_1 = 300,   
 level\_2 = 400,   
 level\_3 = 500,   
 level\_4 = 580,   
 level\_5 = 630,   
 level\_6 = 690,  
 level\_7 = 730,  
 level\_8 = 1000  
 )  
plot\_bts\_level\_loss(test,target\_col='CreditScore\_level')  
def get\_loss\_coverage(test,target\_level):  
 #level 5-Leve 8 Loss (percentage of default people)  
 L5\_loss = test[test[target\_level]>=5 ].label.value\_counts()/len(test[test[target\_level]>=5 ])  
 #level 5- level 8 Coverage (percentage of good people)  
 L5\_coverage=test[test[target\_level]>=5 ].label.value\_counts()[0]/test[test.label==0].shape[0]  
 print("Level 5-Level 8: Loss is ",L5\_loss[1], "; Coverage is ",L5\_coverage)  
 #level 6-level 8 Loss  
 L6\_loss=test[test[target\_level]>=6 ].label.value\_counts()/len(test[test[target\_level]>=6 ])  
 #level 6-level 8 Coverage  
 L6\_coverage=test[test[target\_level]>=6].label.value\_counts()[0]/test[test.label==0].shape[0]  
 print("Level 6-Level 8: Loss is ",L6\_loss[1], "; Coverage is ",L6\_coverage)

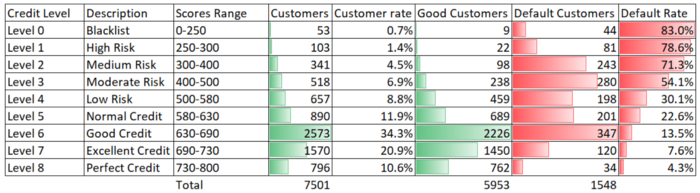


Copyright by AI4Finance-Education

This plot indicates the distribution of each **credit level** and the **credit default rate** at that level.



Copyright by AI4Finance-Education



Copyright by AI4Finance-Education

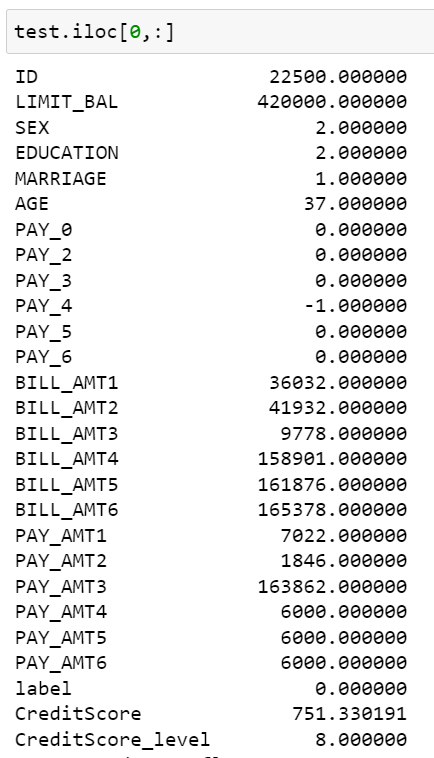
By examining each level, you will have **a loss and coverage table**. For example, if you send the balance transfer offers to all the people **7501** (**L0-L8**) you will incur a **21% loss rate (1548/7501)**.

To reach the goal (≤10% loss and ≥70% coverage), you need to **pick L6-L8 with a 10.1% loss (347+120+34)/(2573+1570+796) and a 75% coverage (2226+1450+762)/5953**.

Basically, next month (assuming that the test set is next month’s data) the business-side people will send the balance transfer offers to customers with **a Level 6 rating or above, a total of 4939 customers.**

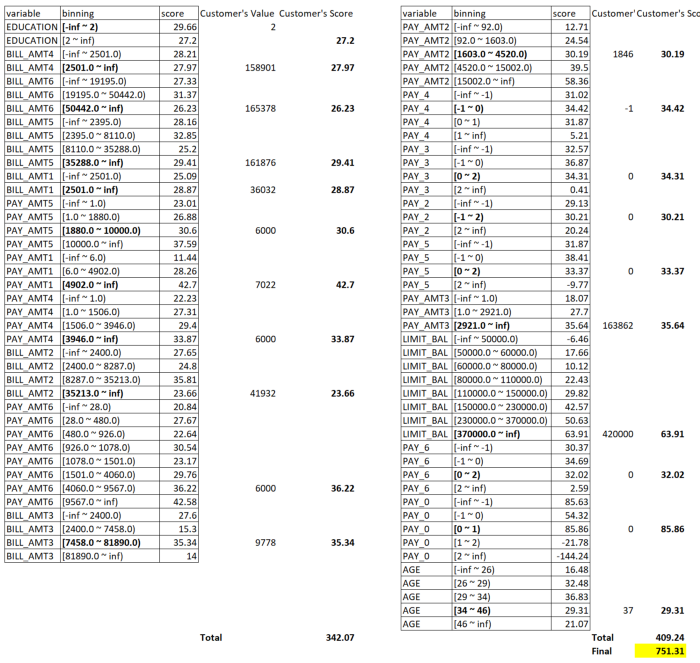
**11. Manually Test our Scorecard**

Can we send a balance transfer offer to a customer with the following information next month?



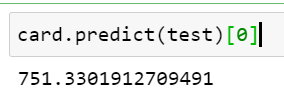
Copyright by AI4Finance-Education

Let’s manually check the scorecard.



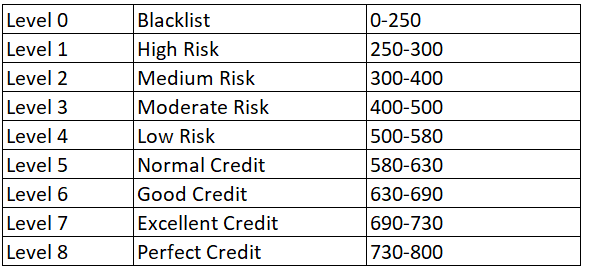
Copyright by AI4Finance-Education

Inference by toad:



Copyright by AI4Finance-Education

By checking our Credit Level table:



Copyright by AI4Finance-Education

We can see that this customer is a **Level 8 customer with Perfect Credit**. So we can issue him/her a balance transfer offer.

**Conclusion:**

This blog walks through the end-to-end process of building a credit scorecard based on the open-source ML tool [**Toad**](https://github.com/amphibian-dev/toad)**.**

The following ML buzzwords are discussed in this blog:

**Information Value (IV), Weight of evidence (WOE), Population Stability Index (PSI), AUC (Area under the ROC Curve), KS (Kolmogorov-Smirnov), Logistic Regression (LR), GBDT (Gradient Boosting Decision Tree)**

For future work, as a data scientist, you could try using other ML models to build the scorecard, such as Deep Neural Networks, to increase the accuracy further and lower the false positive rate to improve customer satisfaction.

Thanks.

***Note from Towards Data Science’s editors:*** *While we allow independent authors to publish articles in accordance with our* [*rules and guidelines*](https://towardsdatascience.com/questions-96667b06af5)*, we do not endorse each author’s contribution. You should not rely on an author’s works without seeking professional advice. See our* [*Reader Terms*](https://towardsdatascience.com/readers-terms-b5d780a700a4) *for details.*