Default of Credit Card Clients

Default of credit card clients data set stored in UCI Center for Machine Learning and Intelligent System 'Machine Learning Repository'.

Data set consists of 24 attributes and 30,000 instances.

The first 'six' attributes include clients' balance limit and other statistics such as age, sex, marriage, age and education.

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
0	1	20000	2	2	1	24	2	2	-1	-1
1	2	120000	2	2	2	26	-1	2	0	0
2	3	90000	2	2	2	34	0	0	0	0
3	4	50000	2	2	1	37	0	0	0	0
4	5	50000	1	2	1	57	-1	0	-1	0

DATA VISUALIZATION :

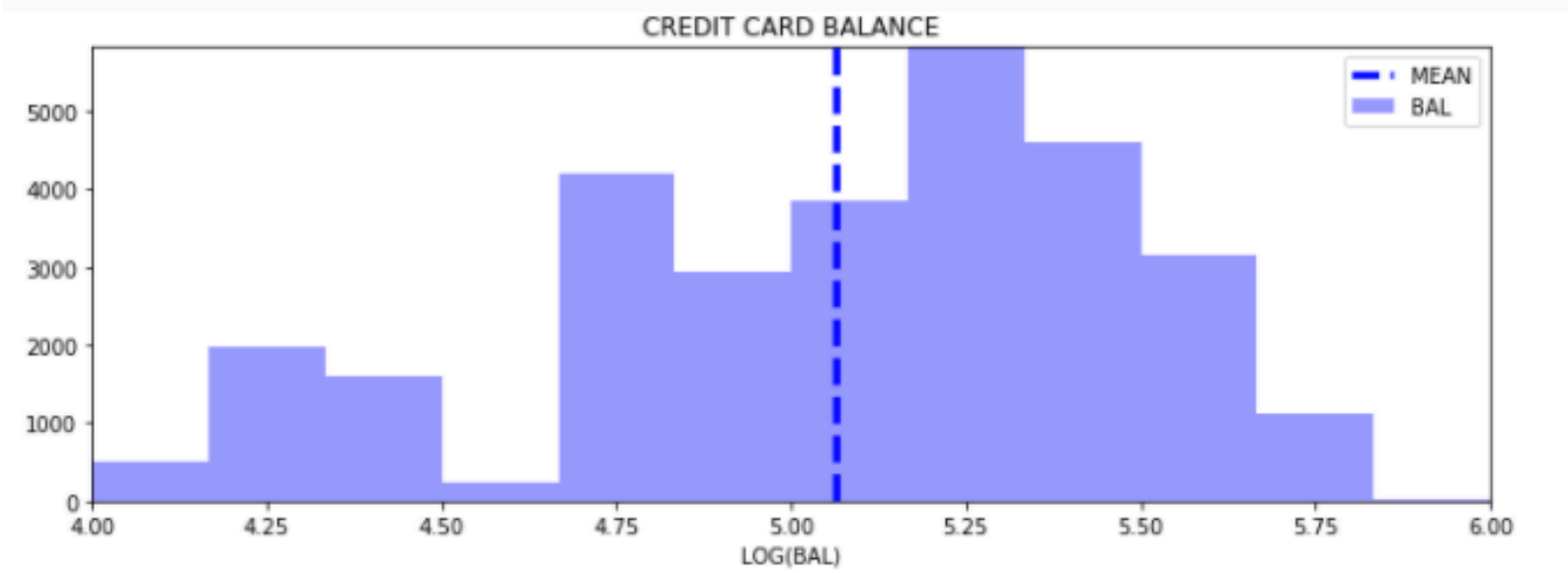
BALANCE LIMIT

Default of credit card clients data set stored in UCI Center for Machine Learning and Intelligent System ‘Machine Learning Repository’.

Data set consists of

=> 24 attributes and

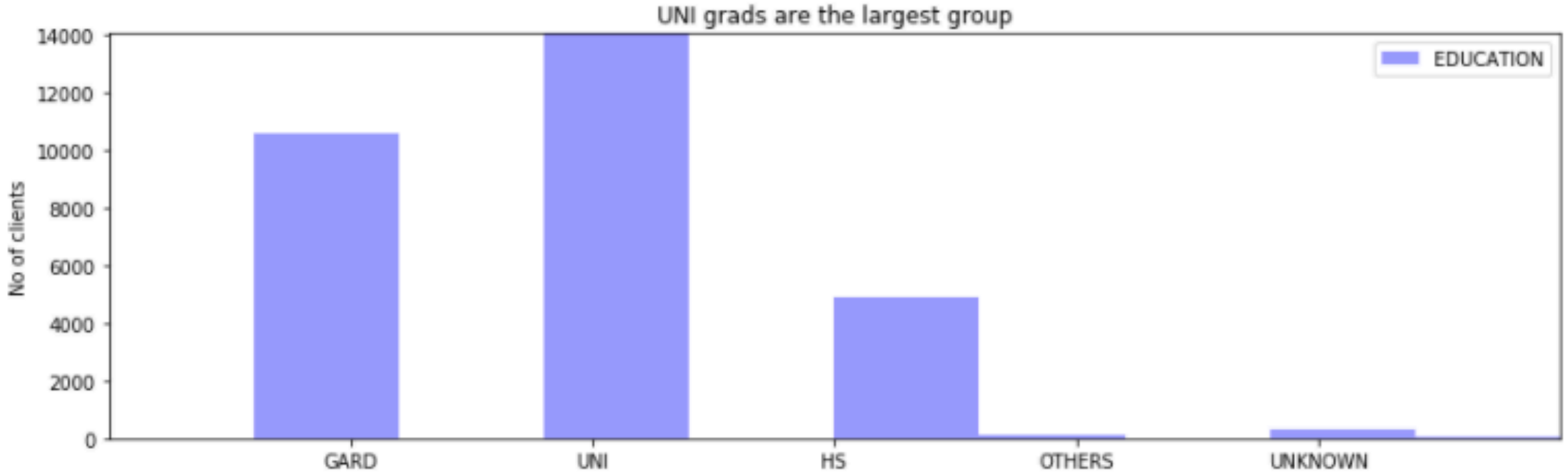
=> 30,000 instances



The first ‘six’ attributes include clients’ balance limit and other statistics such as age, sex, marriage, age and education.

Hist shows balance distribution plotted on log10 scale, average balance more than 100K.

CLIENTS' EDUCATION:




```
df['EDUCATION'].value_counts()/df['EDUCATION'].value_counts().sum()
```

```
2    0.467667
1    0.352833
3    0.163900
5    0.009333
4    0.004100
6    0.001700
0    0.000467
```

EDUCATION => GARD : UNI : HS : OTHERS = 35 : 47 : 16 : 1

CLIENTS' AGE:

Age group:

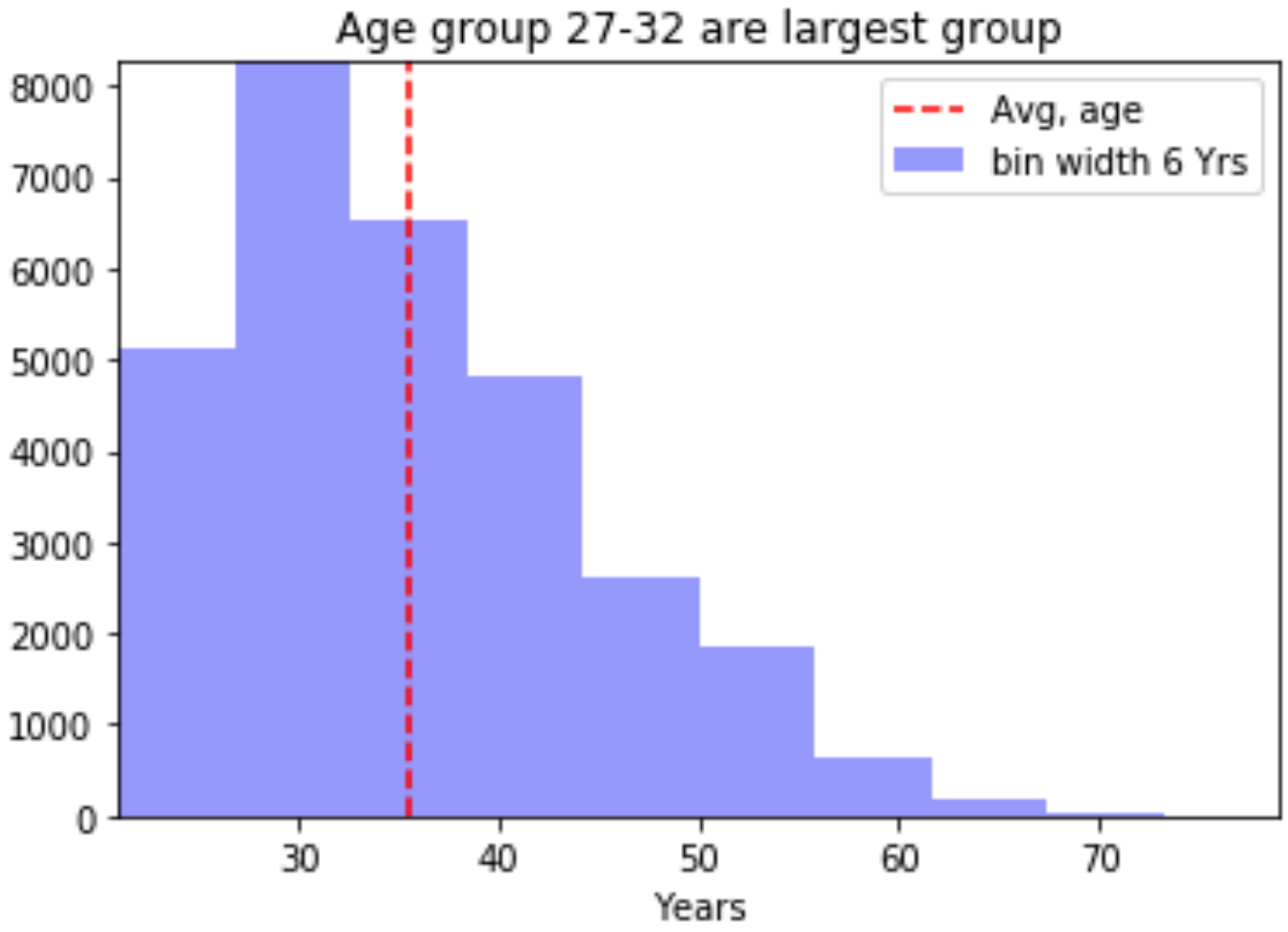
Largest group of customer

In terms of age:

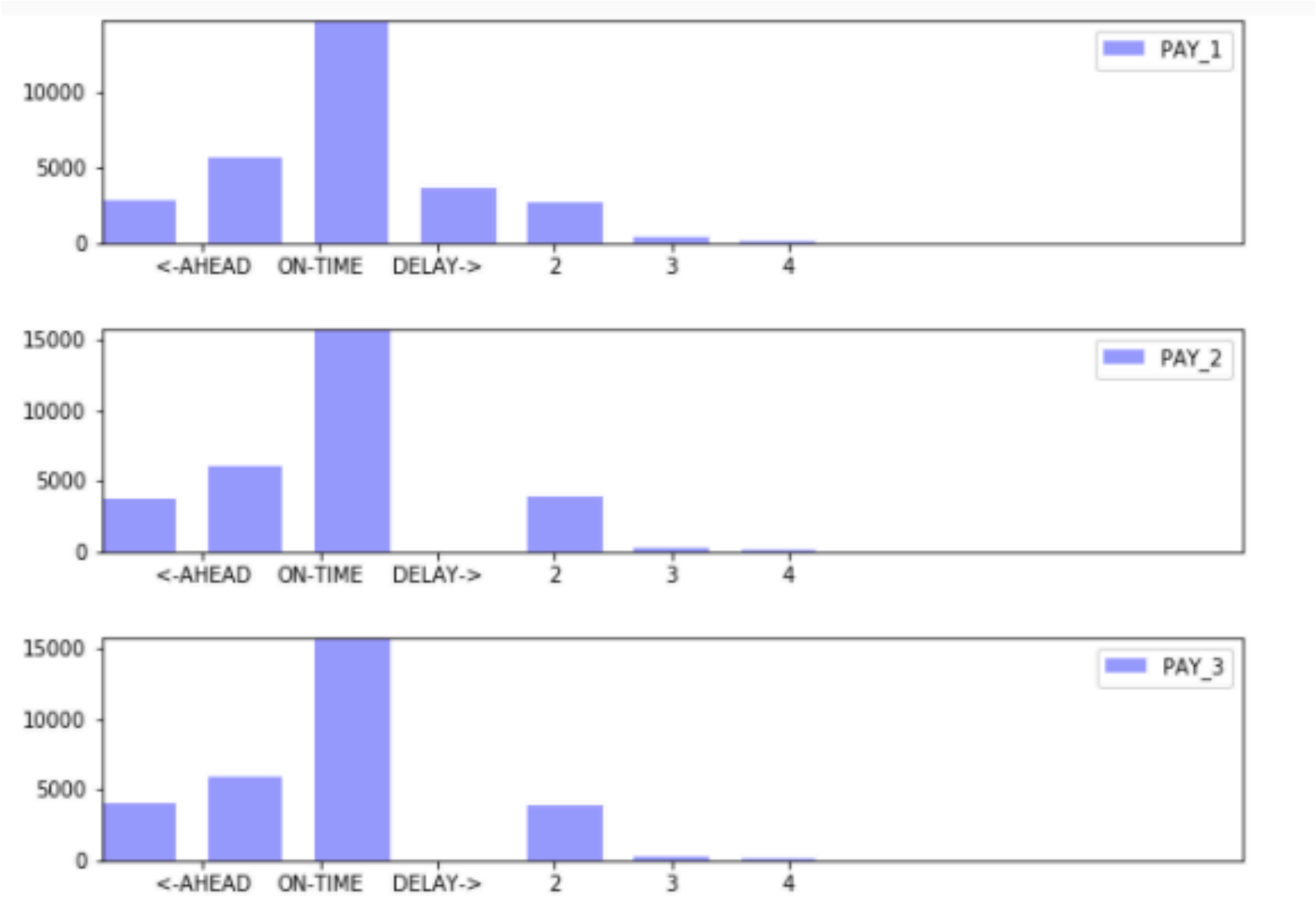
27-32 Years

Next largest group:

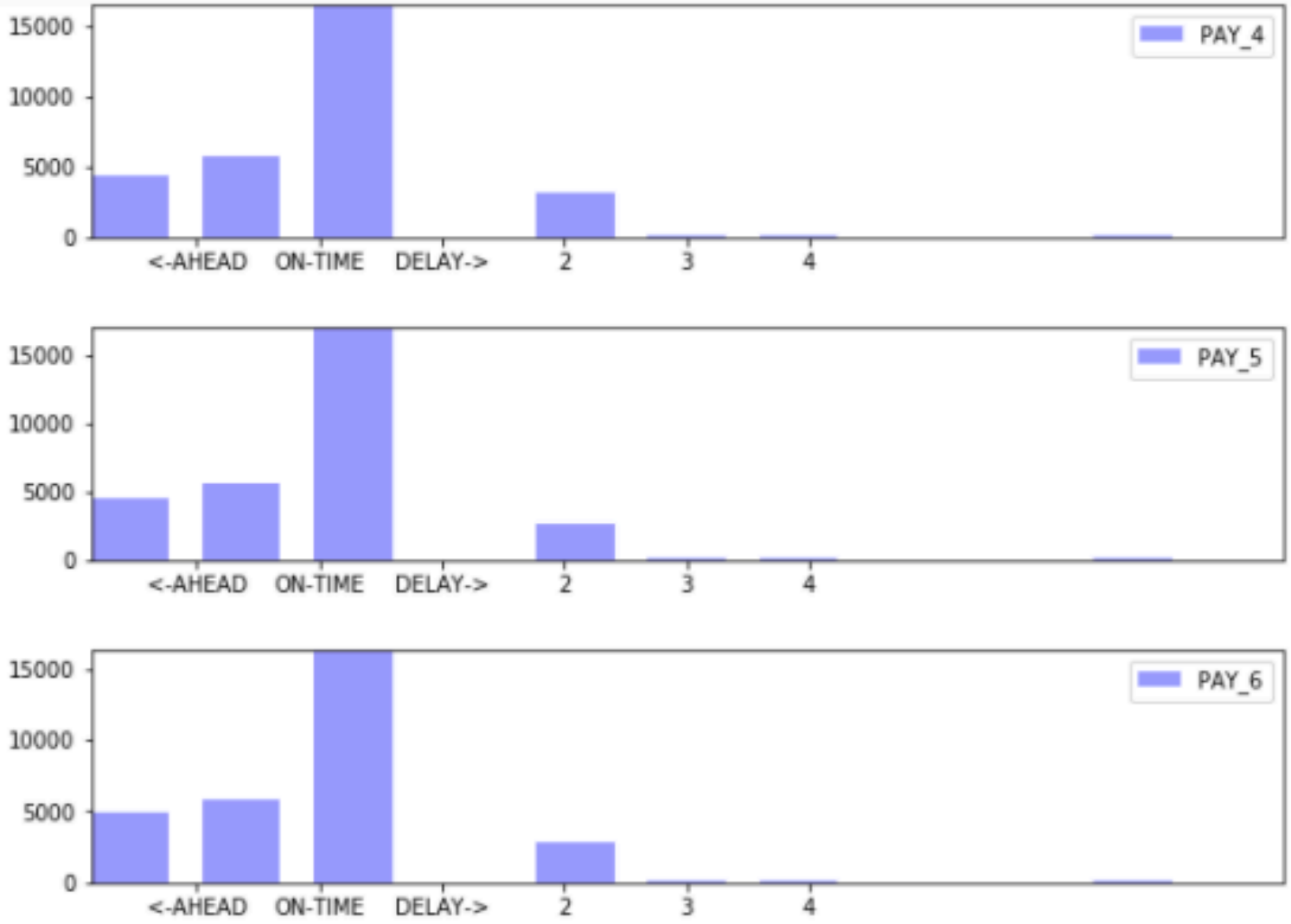
33-38 Years



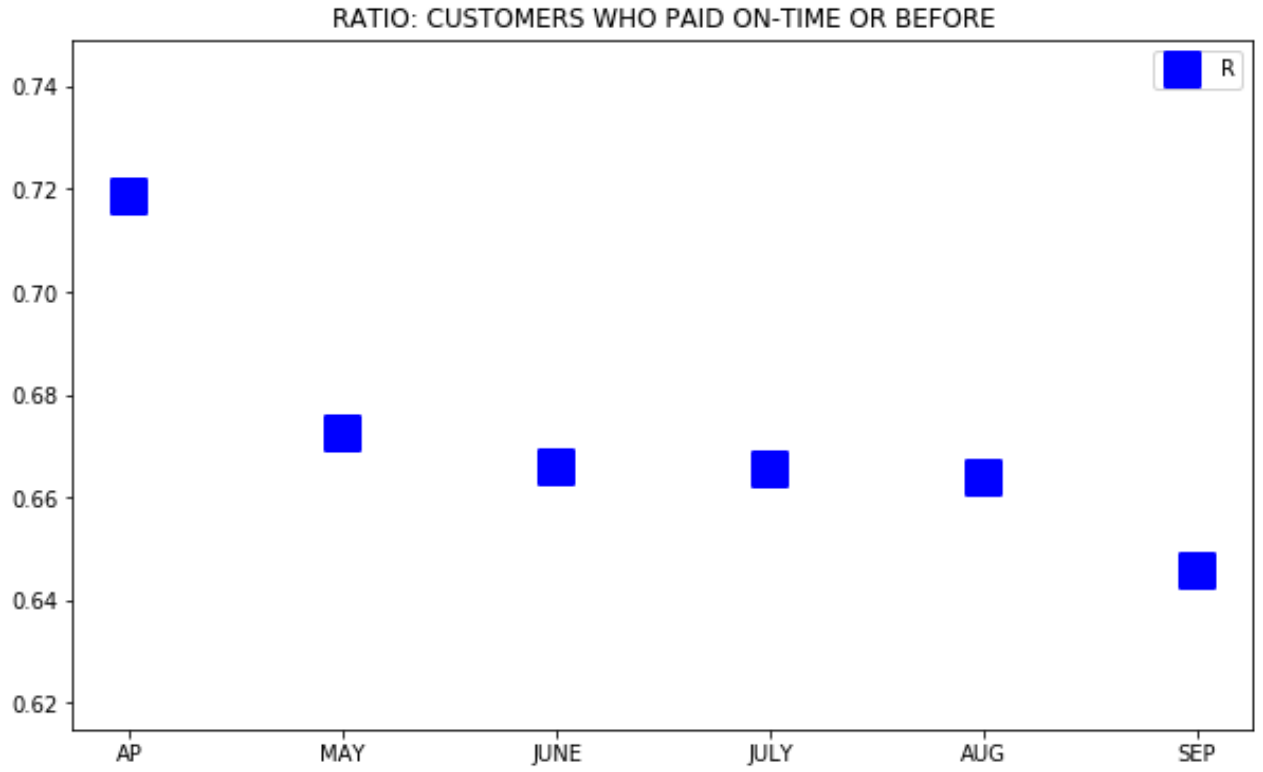
CUSTOMERS CONSECUTIVE PAY RECORDS (APRIL-JUNE, 2005) :



CUSTOMERS CONSECUTIVE PAY RECORDS (JULY-SEPT., 2005) :

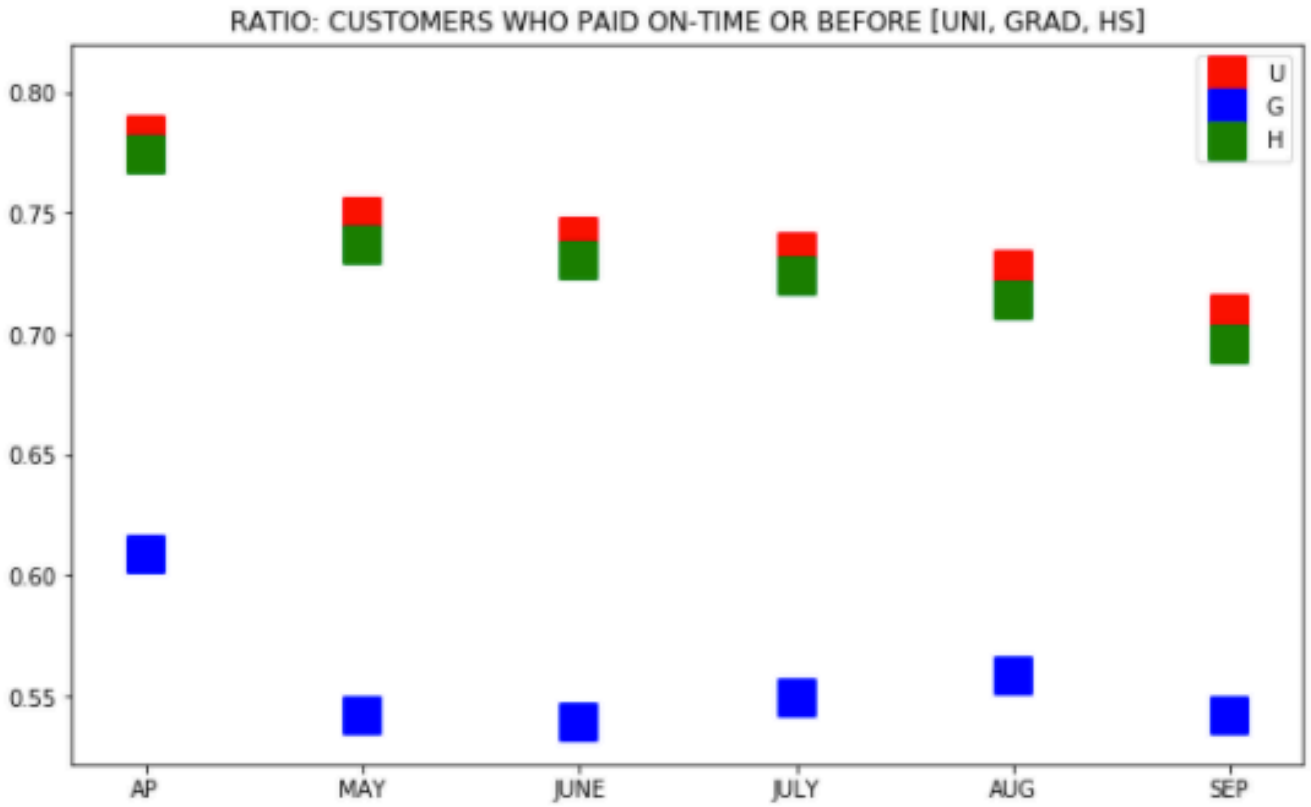


SIX CONSECUTIVE PAY RECORDS (APRIL-SEP, 2005) :



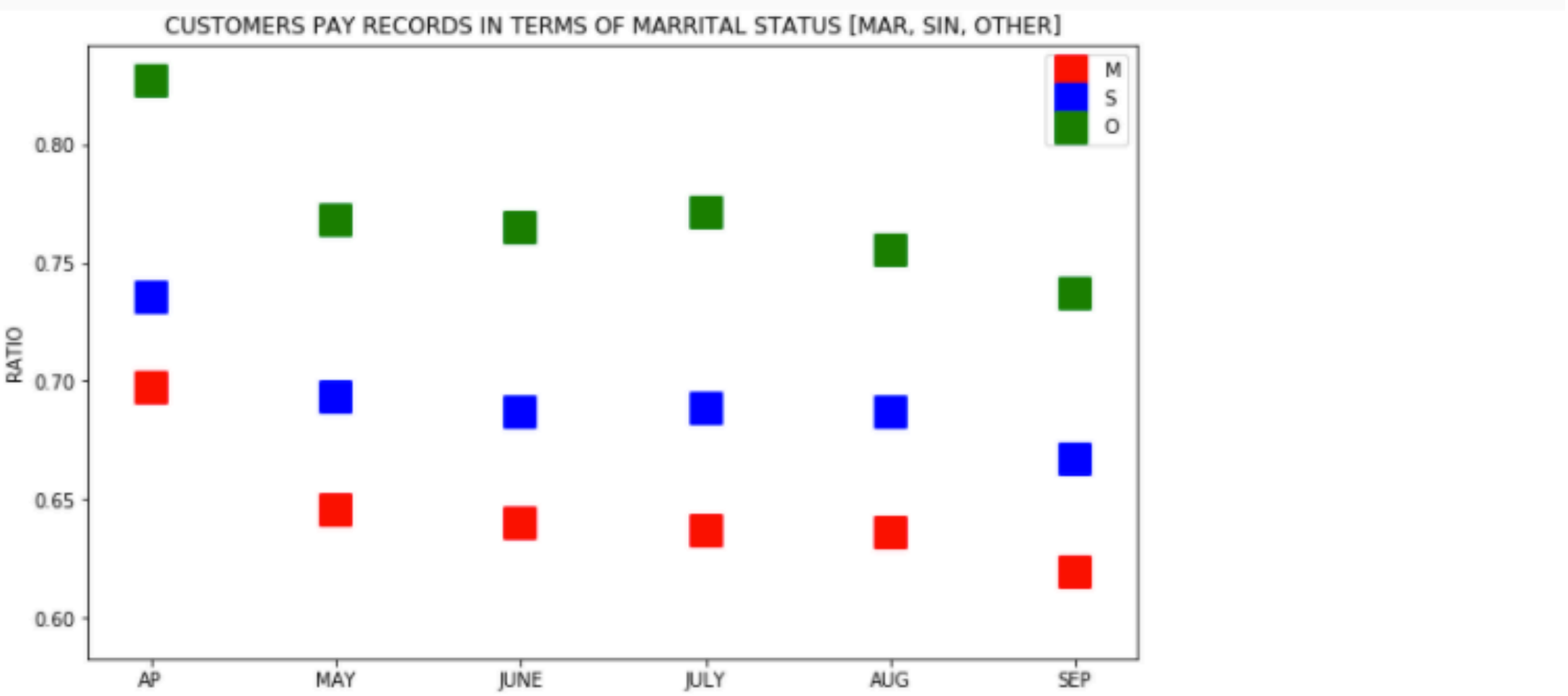
As we see roughly 72% client made the first payments on time. It drops down to less than 68% in the next pay cycle. Trend shows further drop down in terms of number of customer who made payments on time or before the time with small fluctuations towards the end of pay cycles.

CUSTOMER PAY RECORDS IN TERMS OF EDUCATION :



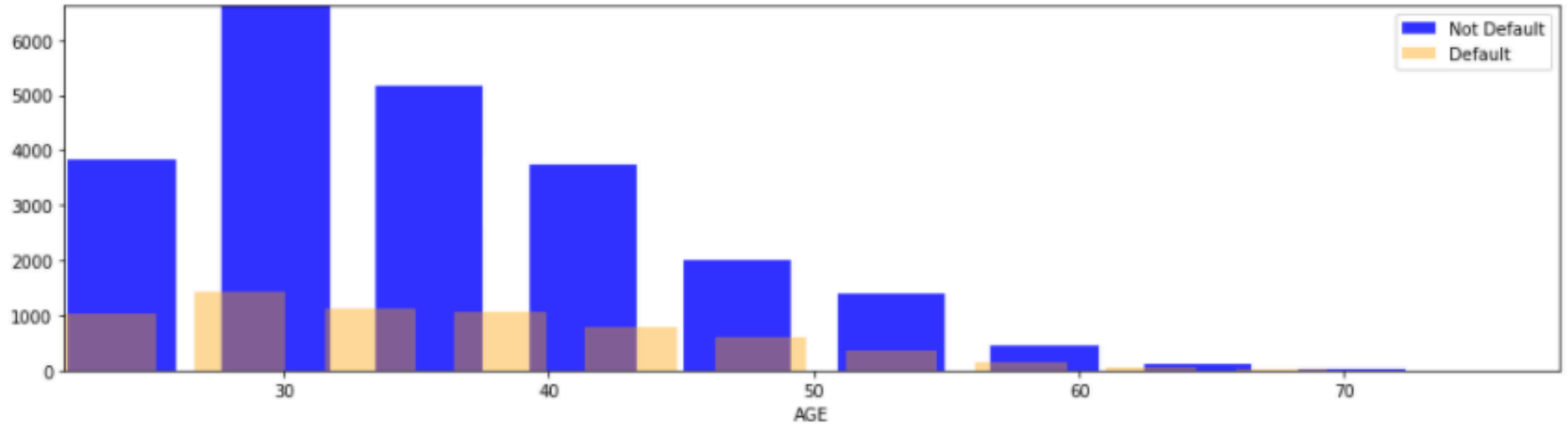
We found that in terms of pay records university and HS students show similar trends as well as grad student, however grad student make a consistent difference from the other two groups.

CUSTOMER PAY RECORDS IN TERMS OF MARRITAL STATUS :

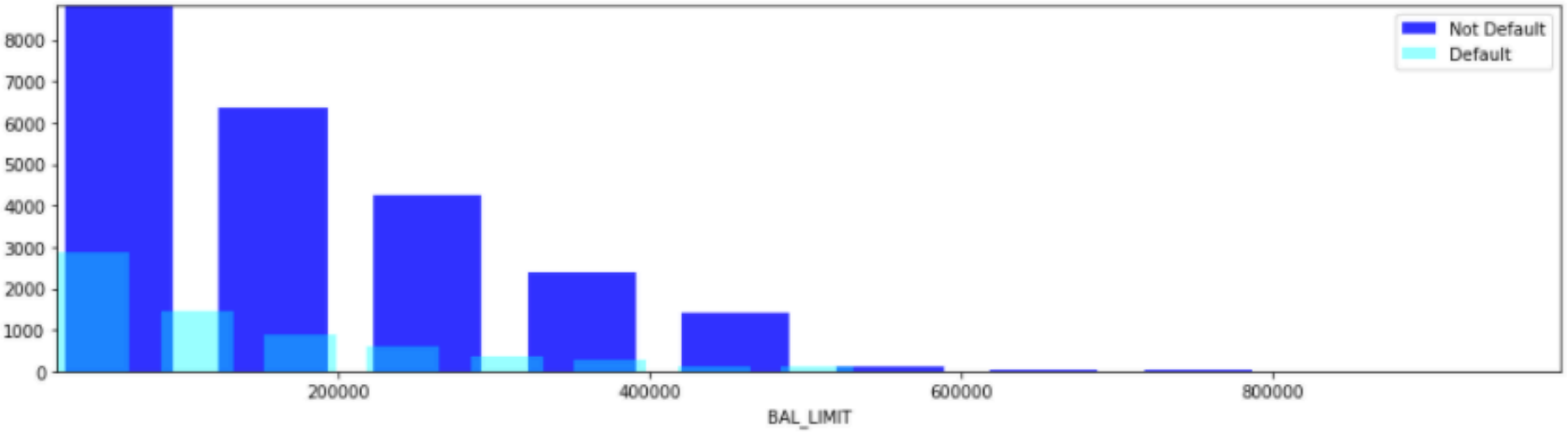


It shows comparing to Married or Single, the other group pay records are better, however this group represents only 1% of the client.

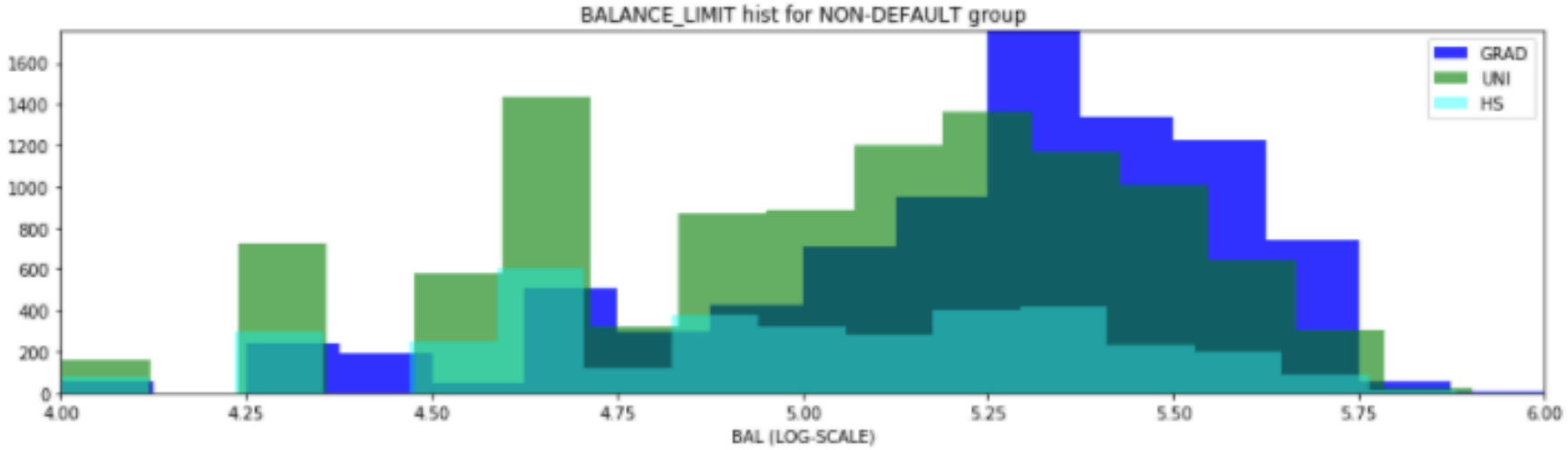
CLASSIFICATION (PAY status, default or not default in terms of AGE group)

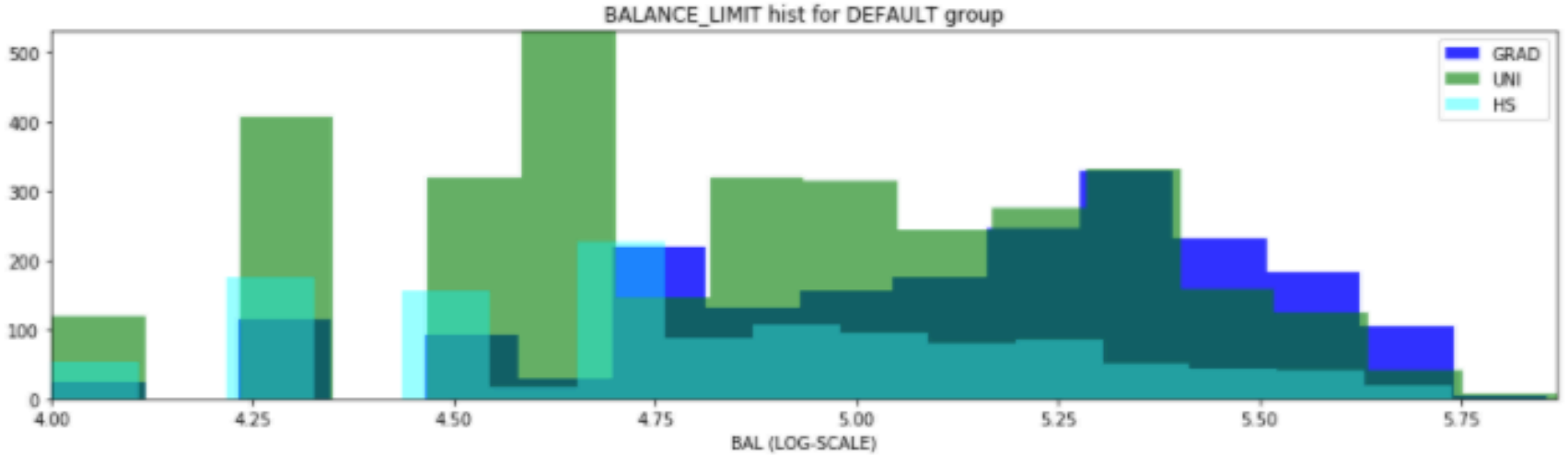


CLIENTS BALANCE (Group: default, Not default)

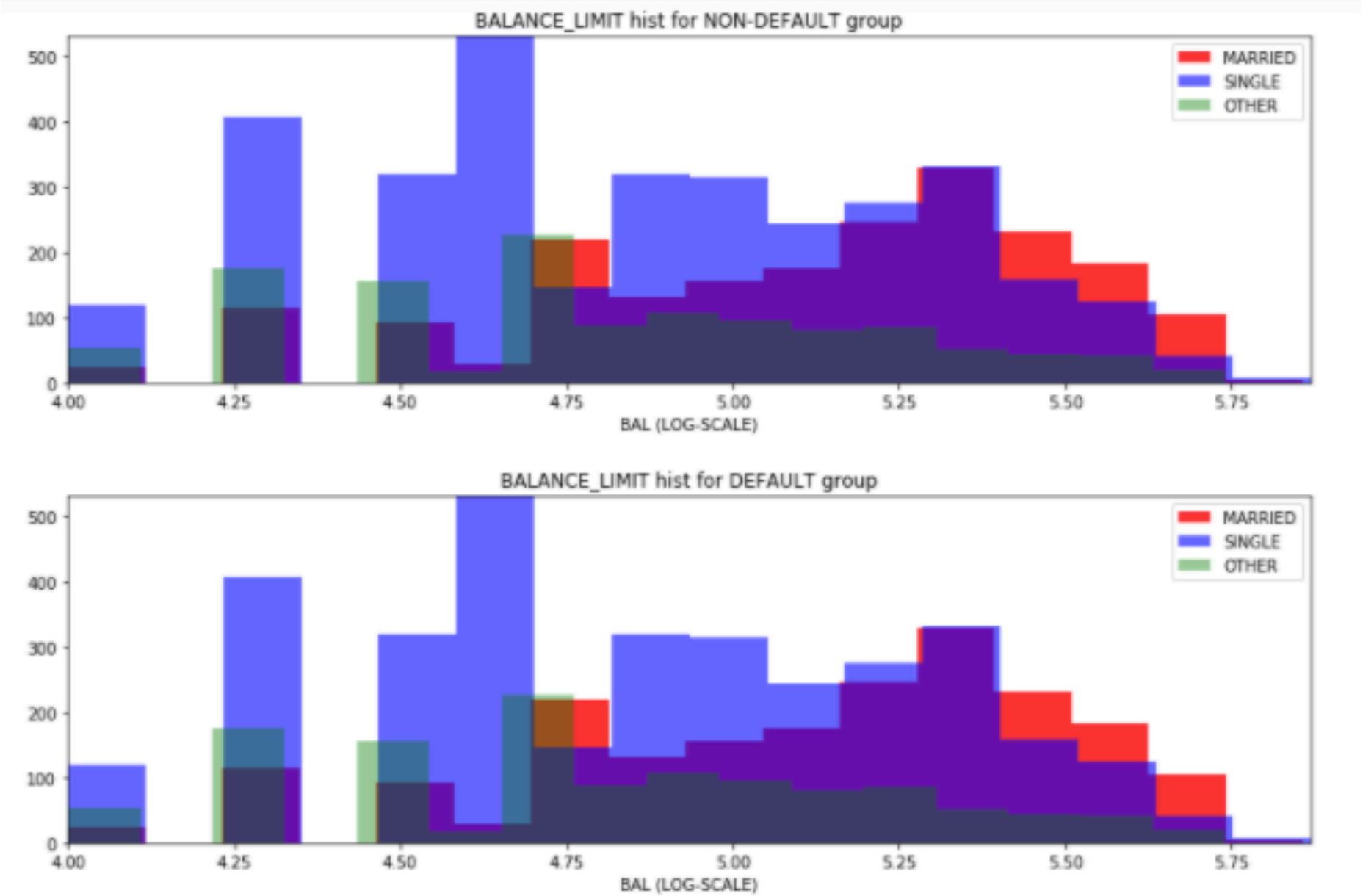


CLASSIFICATION (PAY status, default or Not default in terms of EDUCATION)

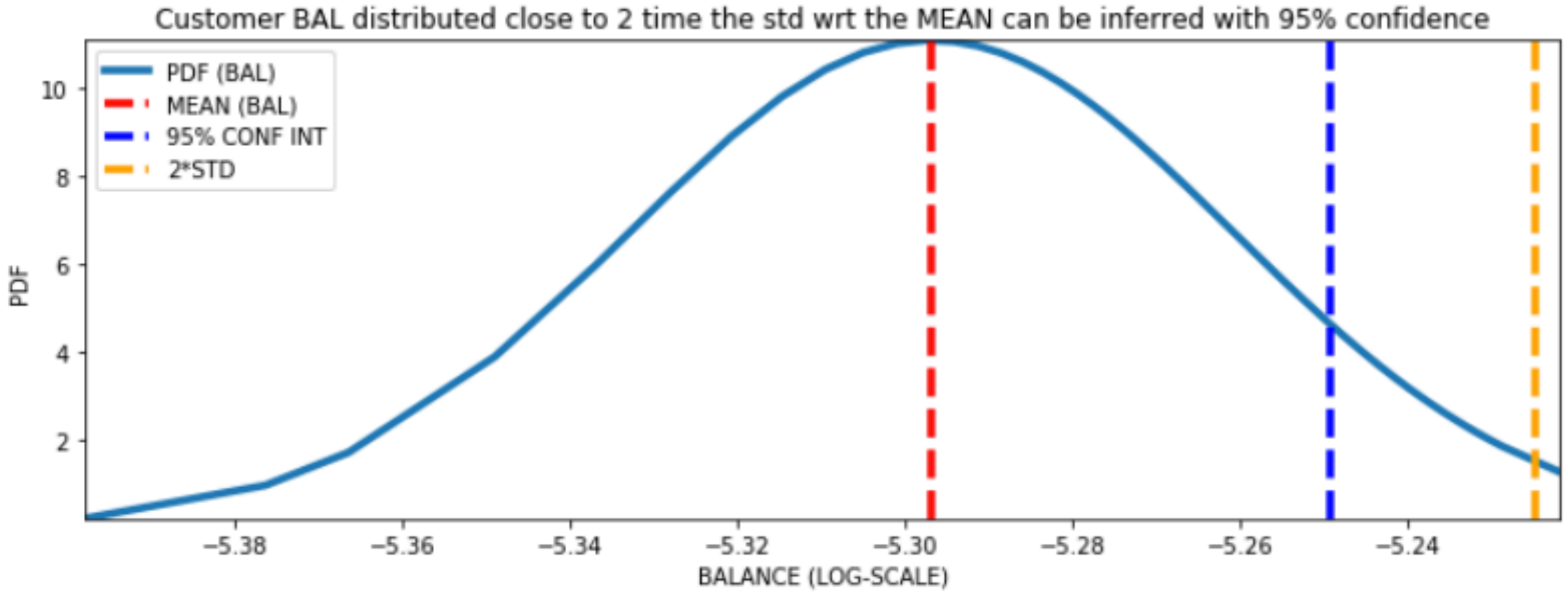




CLASSIFICATION (PAY status, default or Not default in terms of MARRIAGE)



ANALYSIS : Probability Distribution Function (PDF) of Customer BALANCE (log10 scale)

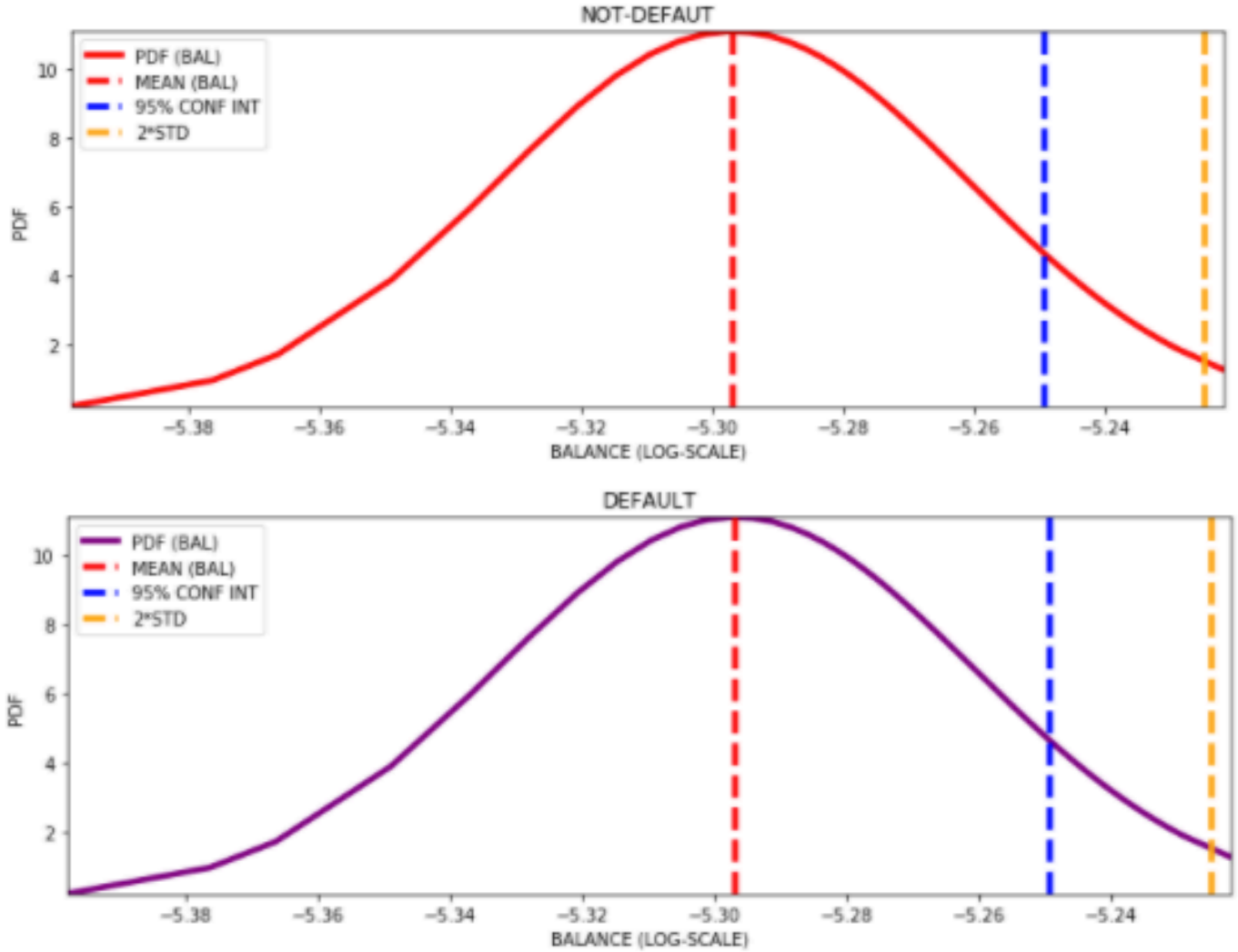


Overall customers balance distributed close to 2*STD WRT the MEAN balance can be inferred with 95% CONFIDENCE.

This TREND remains consistent even if we group customers in terms of

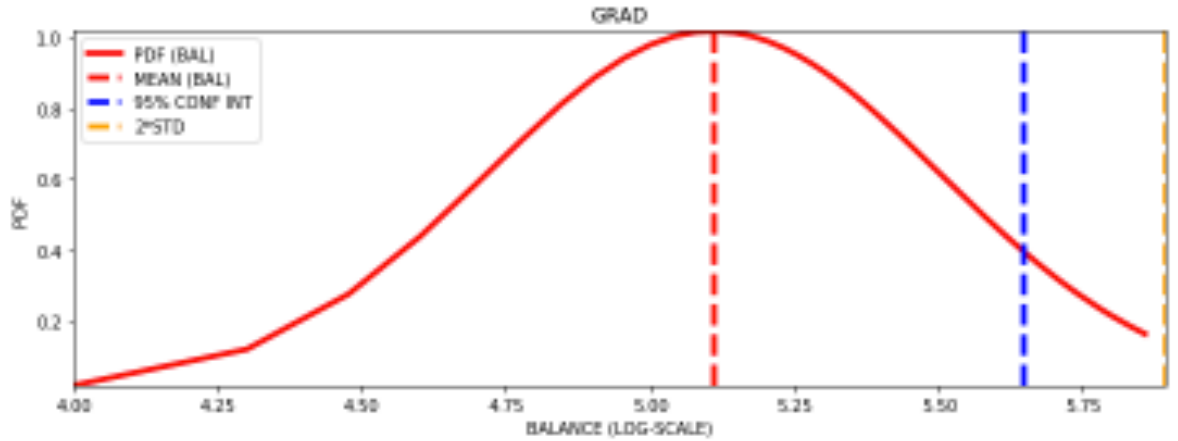
- Default, Not Default
- Education
- Marriage or
- Age group

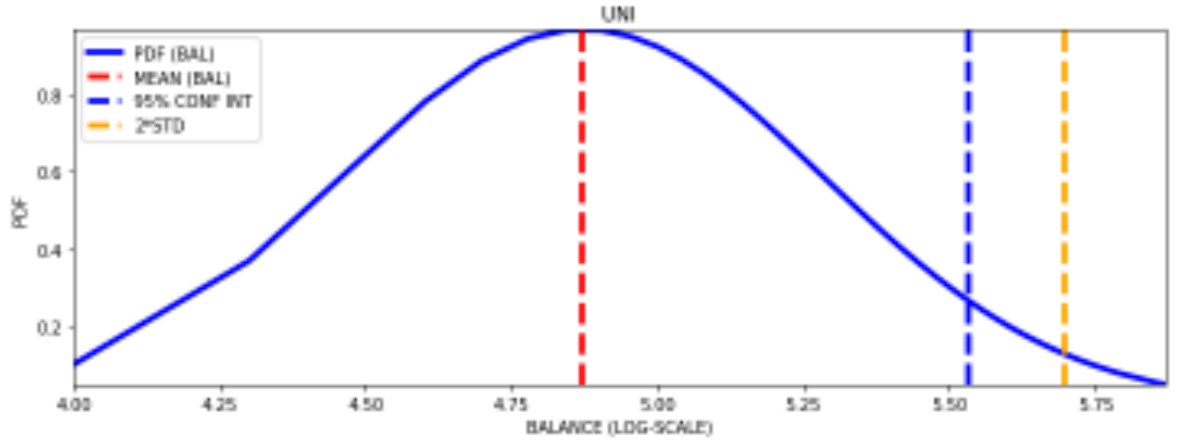
95% CONFIDENCE INTERVALE : Default, Not Default Group

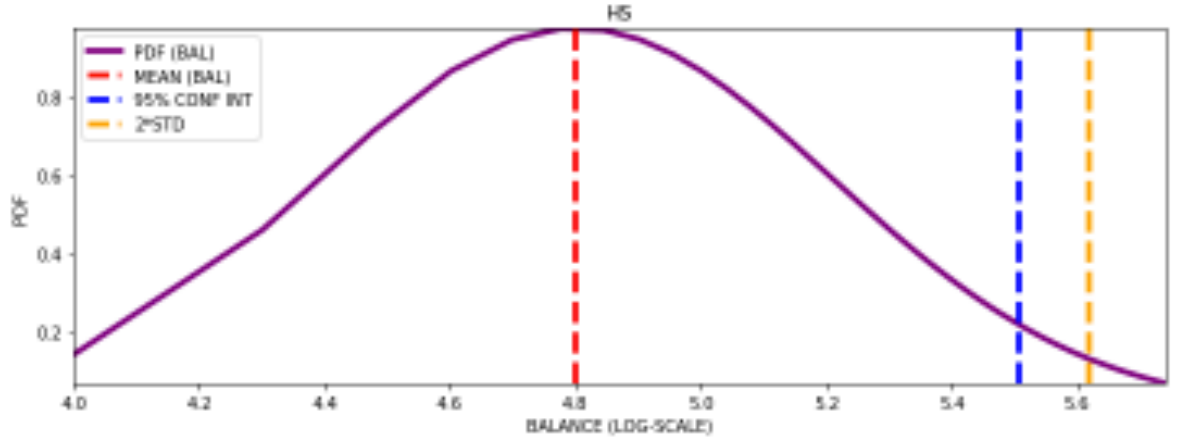


o Education

95% CONFIDENCE INTERVALE :
GRAD,
UNI STU
HS STU
Group







HYPOTHESIS TEST:

Now we would do a HYPOTHESIS with a claim that

"THE ABOVE TWO GROUPS (default, not default) ARE STATISTICALLY IDENTIAL"

```
ttest_ind(df_nd['BAL'].values, df_d['BAL'].values, equal_var=False)
```

```
Ttest_indResult(statistic=28.951587933509845, pvalue=3.3641002455114717e-178)
```

With T test we found the p-value is < 0.05, so the null hypothesis is rejected, i.e.; the two groups are statistically different.

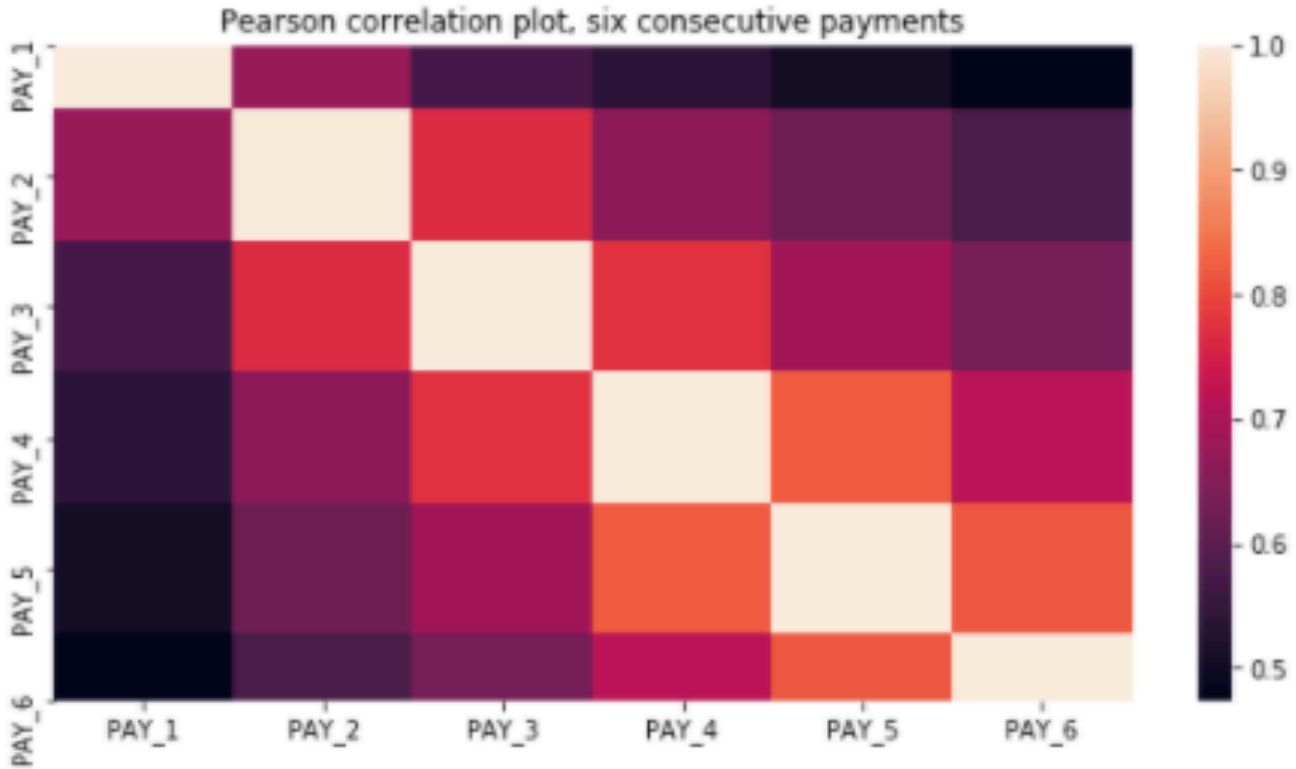
We did similar tests in terms of other attributes such as education, marital status or age group and found that null hypothesis is rejected.

FEATURE CORRELATION :

Pearson correlation among the time-based payment features

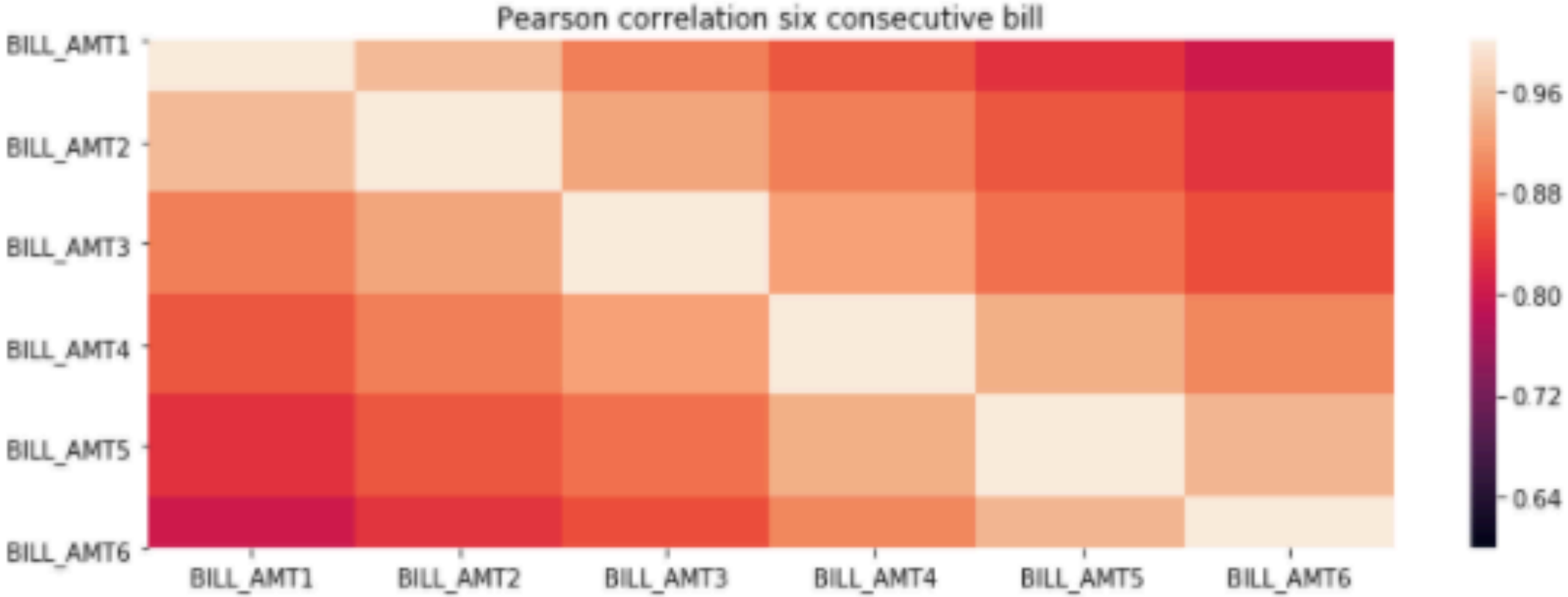
```
pay = ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']
corr_pay = df[pay].corr()
```

```
plt.figure(figsize=(10,5))
sns.heatmap(corr_pay, vmin=np.min(np.min(corr_pay)), vmax=1)
plt.title('Pearson correlation plot, six consecutive payments')
plt.show()
```



Pearson correlation bill amount

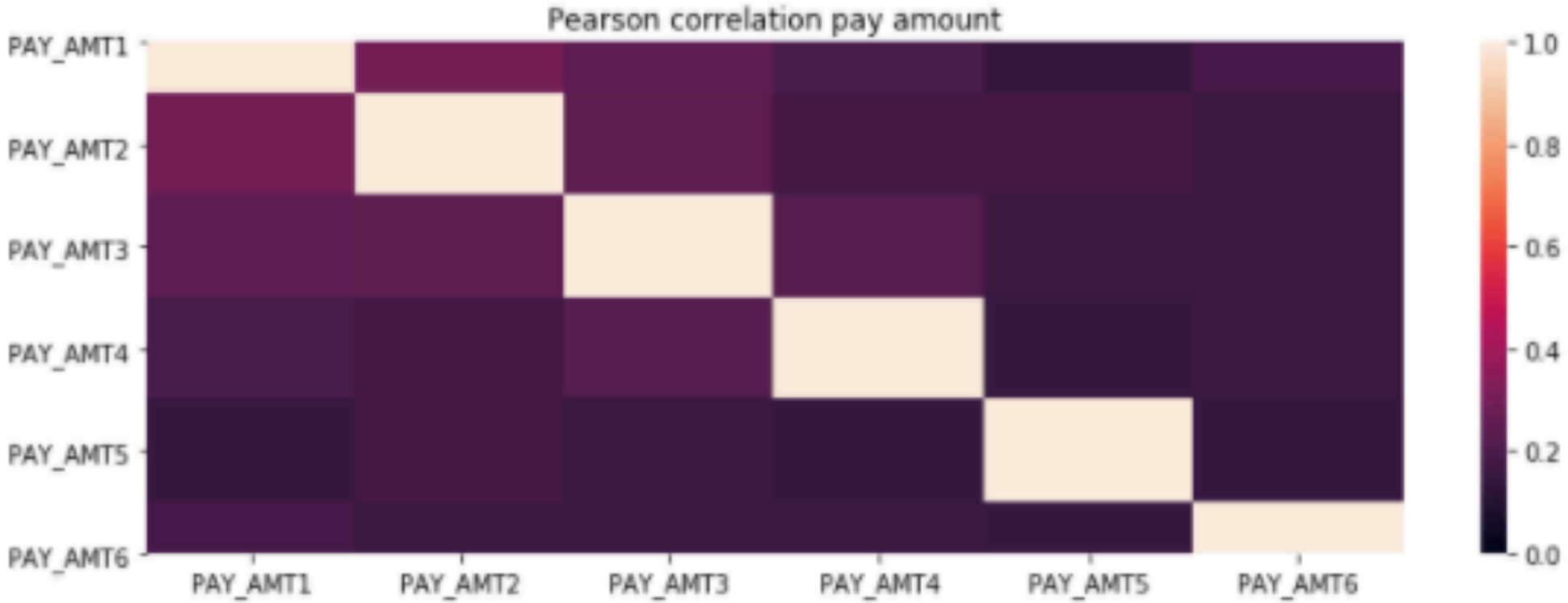
	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6
BILL_AMT1	1.000000	0.951484	0.892279	0.860272	0.829779	0.802650
BILL_AMT2	0.951484	1.000000	0.928326	0.892482	0.859778	0.831594
BILL_AMT3	0.892279	0.928326	1.000000	0.923969	0.883910	0.853320
BILL_AMT4	0.860272	0.892482	0.923969	1.000000	0.940134	0.900941
BILL_AMT5	0.829779	0.859778	0.883910	0.940134	1.000000	0.946197
BILL_AMT6	0.802650	0.831594	0.853320	0.900941	0.946197	1.000000



Pearson correlation pay amount

	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6
PAY_AMT1	1.000000	0.285576	0.252191	0.199558	0.148459	0.185735
PAY_AMT2	0.285576	1.000000	0.244770	0.180107	0.180908	0.157634
PAY_AMT3	0.252191	0.244770	1.000000	0.216325	0.159214	0.162740
PAY_AMT4	0.199558	0.180107	0.216325	1.000000	0.151830	0.157834
PAY_AMT5	0.148459	0.180908	0.159214	0.151830	1.000000	0.154896
PAY_AMT6	0.185735	0.157634	0.162740	0.157834	0.154896	1.000000

```
<function matplotlib.pyplot.show(*args, **kw)>
```



LAON DEFAULT PREDICTION :

Now we will apply various Machine Learning Algorithms to predict whether clients will be a default or not default in the next month.

PREDICTION: MACHINE LEARNING

```
df.columns
```

```
Index(['ID', 'BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_1', 'PAY_2',
       'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'default payment next month'],
      dtype='object')
```

Features :

```
PAY = []
for idx in range(6):
    for j in range(len(df)):
        if (df[str(df.columns[idx+11])].iloc[j]) <= (df[str(df.columns[idx+17])].iloc[j]):
            PAY.append(1)
        elif (df[str(df.columns[idx+11])].iloc[j]) < 0.75*(df[str(df.columns[idx+17])].iloc[j]):
            PAY.append(2)
        elif (df[str(df.columns[idx+11])].iloc[j]) < 0.50*(df[str(df.columns[idx+17])].iloc[j]):
            PAY.append(3)
        elif (df[str(df.columns[idx+11])].iloc[j]) < 0.25*(df[str(df.columns[idx+17])].iloc[j]):
            PAY.append(4)
        else:
            PAY.append(5)
```

```
payr = np.array(PAY).reshape(6, len(df))
payr[:,0:10]
```

```
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
       [5, 5, 5, 5, 5, 5, 5, 5, 5, 1],
       [5, 5, 5, 5, 1, 5, 5, 1, 5, 1],
       [5, 5, 5, 5, 5, 5, 5, 5, 5, 1],
       [1, 5, 5, 5, 5, 5, 5, 1, 5, 1],
       [1, 5, 5, 5, 5, 5, 5, 1, 5, 5]])
```

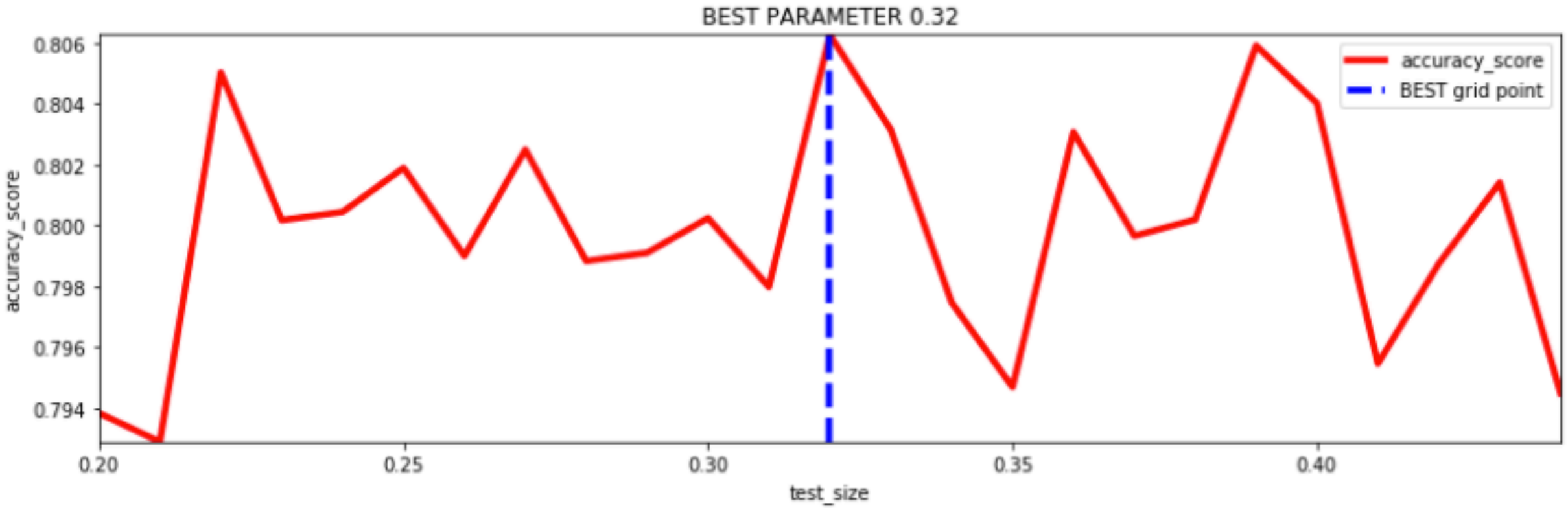
```
df[ ['PAY_1', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6',
     'PAY-FE-1', 'PAY-FE-2', 'PAY-FE-3', 'PAY-FE-4', 'PAY-FE-5', 'PAY-FE-6'] ].head()
```

	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	PAY-FE-1	PAY-FE-2	PAY-FE-3	PAY-FE-4	PAY-FE-5	PAY-FE-6
0	2	2	-1	-1	-2	-2	1	5	5	5	1	1
1	-1	2	0	0	0	2	1	5	5	5	5	5
2	0	0	0	0	0	0	1	5	5	5	5	5
3	0	0	0	0	0	0	1	5	5	5	5	5
4	-1	0	-1	0	0	0	1	5	1	5	5	5

```
df['PAY-FE-1'].value_counts()/df['PAY-FE-1'].value_counts().sum()
```

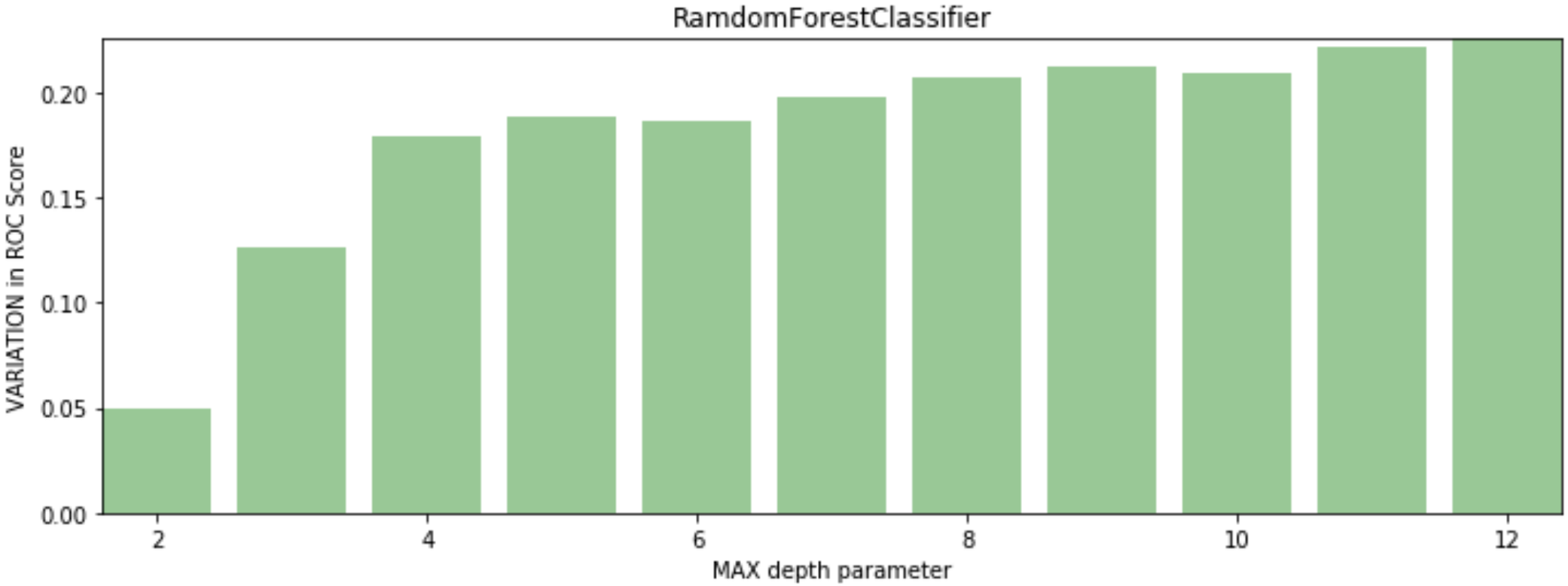
```
1    0.978000
5    0.020800
4    0.000767
```

GaussianNB:



Best parameter 0.32

Random Forest Classifier :



ROC SCORE: 0.9801, Tree depth = 12

Decision Tree Classifier

```
dtc = DecisionTreeClassifier()
dtc.fit(x_tr, y_tr)
y_p = dtc.predict(x_t)
y_tr_p = dtc.predict(x_tr)
print('Accuracy score over the test set %0.2f' %accuracy_score(y_t, y_p))
print('Accuracy score over the training set %0.2f' %accuracy_score(y_tr, y_tr_p))
print('Accuracy score over the training set %0.2f' %roc_auc_score(y_tr, y_tr_p))
print('roc_auc_score over the training set %0.2f' %roc_auc_score(y_tr, y_tr_p))
```

```
Accuracy score over the test set 0.74
Accuracy score over the training set 0.98
Accuracy score over the training set 0.95
roc_auc_score over the training set 0.95
```

Summary and final comments :

```
MLclf = [LogisticRegression(), GaussianNB(), DecisionTreeClassifier(), ExtraTreesClassifier(), Ran
domForestClassifier(), ensemble.GradientBoostingClassifier() ]
roc_list = []
for clf in MLclf:
    y_p = clf.fit(x_tr, y_tr).predict(x_t)
    roc_list.append(roc_auc_score(y_t, y_p))
    print(roc_auc_score(y_t, y_p))
```

We applied the following ML algorithms:

- **Logistic Regression**
- **GaussianNB**
- **DecisionTreeClassifier**
- **ExtratreesClassifier**
- **RandomForestClassifier**
- **Ensemble Gradient Boosting Classifier**

Among them we found DecisionTreeClassifier did the best prediction with a

roc score 0.99835

Summary and final comments :

```
best_roc = 0
best_roc_idx = 0

for idx in range(len(roc_list)):
    if (roc_list[idx]>best_roc):
        best_roc = roc_list[idx]
        best_roc_idx = idx

print('The best classifier from default parameter :', MLclf[best_roc_idx], '\n', 'with roc_auc_sco
re : ', best_roc)
```

```
The best classifier from default parameter : DecisionTreeClassifier(class_weight=None, criterion
='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
 with roc_auc_score :  0.9983542841660619
```