ML Hackathon

Sai Venkatesh, George T Abraham, Pratyush Nandi
(IMT2017012, IMT2017019, IMT2017518)

Problem Statement:

Given dataset of credit-card usage and payment history,predict the probability of credit default for the next month.

Format of data:

We have 23 features in the dataset, all of them in numerical format. No column has null values. The format of each of the feature is:

- **Column 1:** Amount of the given credit (NT dollar). It includes both the individual consumer credit and his/her family (supplementary) credit.
- **Column 2:** Gender (1 = male, 2 = female).
- **Column 3:** Education (1 = graduate school; 2 = university; 3 = high school; 4 = others,5=unknown,=unknown).
- **Column 4:** Marital status (1 = married, 2 = single, 3 = others).
- **Column 5:** Age (year).
- **Column 6:** the repayment status in September, 2005
- **Column [7:11] :** the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
- **Column 12:** amount of bill statement in September, 2005;
- **Column [13:17]:** amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
- **Column 18:** amount paid in September, 2005
- **Column 19:** amount paid in August, 2005
- **Column 20:** amount paid in July, 2005
- **Column 21:** amount paid in June, 2005
- **Column 22:** amount paid in May, 2005
- **Column 23:** amount paid in April, 2005

The classification label to be predicted is **"default.payment.next.month"**

Data Cleaning:

1.There are very few instances of **“education”** taking values 5, 6 and there are a few 0’s in the column entry which are all grouped together and replaced with value “5”(since 5 and 6 are both categorized as unknowns).

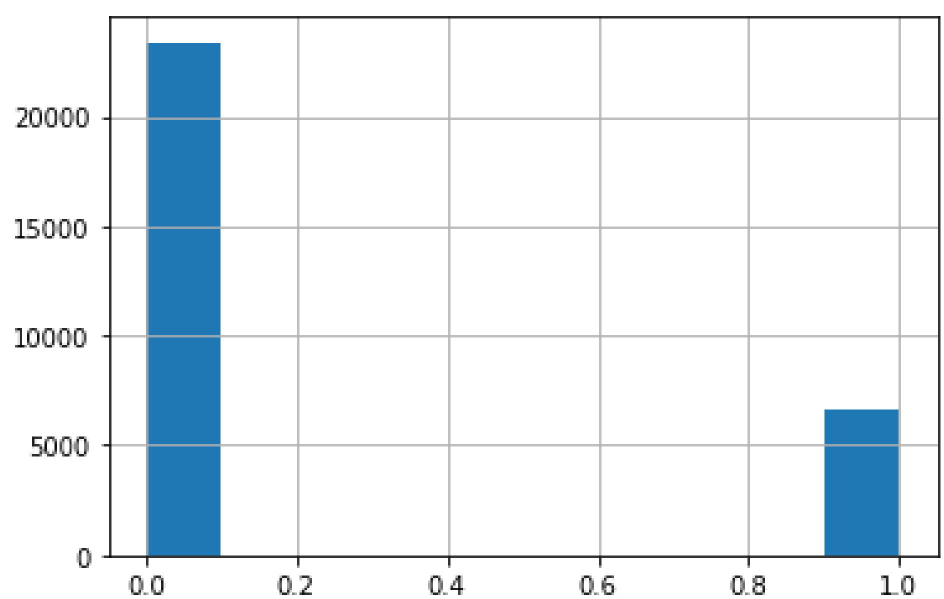
2.There are instances in column **“Marriage”** which take value as “0” even though it is not defined in format,so they are replaced with 3(unknown)

3.Column **PAY_0**(actually corresponds to month september which is “month 1”)and **default.payment.next.month** are renamed to **PAY_1** and **default** respectively.

Feature Analysis:

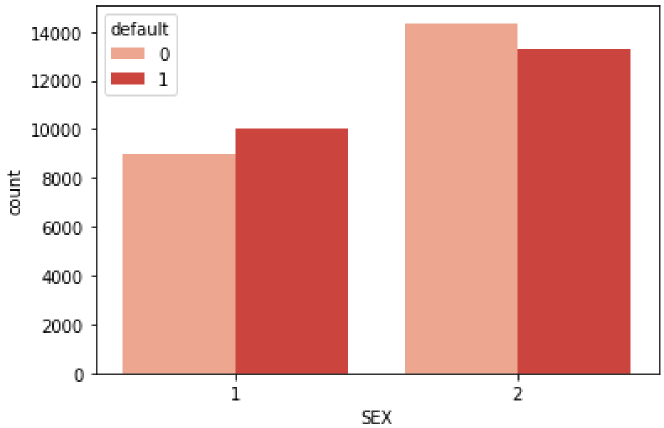
As we can see in the below histogram,the frequency of non-defaulters is much higher than that of the defaulters. Since identifying defaulters is critical to the bank, it is important to balance the dataset to accurately identify defaulters.This is done by upsampling defaulters in the given dataset.

(Properties of features will be seen from a balanced dataset)



After oversampling the dataset, Let us see the properties of the features given.

1.SEX:



From the above plot, we can see that there is not much difference in fraction of defaulters and non defaulters across the genders. So the model is not likely to differentiate between genders to decide if the person is a defaulter or not. So, this is not a very good feature for the model.

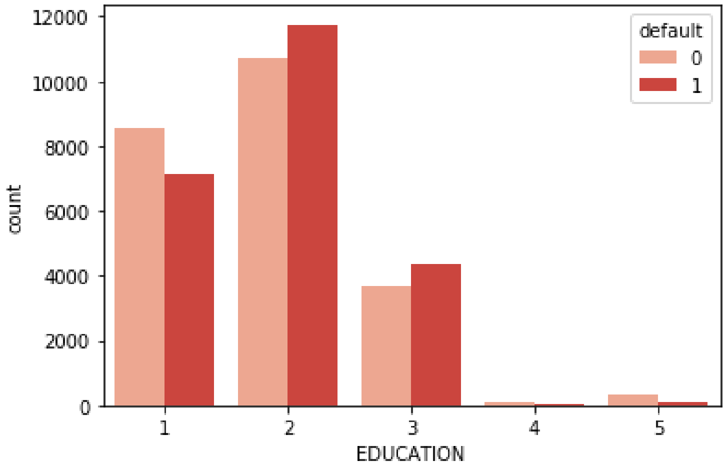
In [15]:
```
temp = upsampled_analy[upsampled_analy["SEX"]==1]
frac = temp.default.value_counts()/temp.shape[0]
frac
```

Out[15]:
```
1    0.527268
0    0.472732
Name: default, dtype: float64
```

In [16]:
```
temp = upsampled_analy[upsampled_analy["SEX"]==2]
frac = temp.default.value_counts()/temp.shape[0]
frac
```

Out[16]:
```
0    0.518801
1    0.481199
Name: default, dtype: float64
```

2.EDUCATION:



The values [1,2,3] yield very similar fraction values in terms of defaulters and non defaulters but they differ from categories [4,5] who are more likely to be non defaulter compared to categories [1,2,3].

In [18]:
```
temp = upsampled_analy[upsampled_analy["EDUCATION"]==1]
frac = temp.default.value_counts()/temp.shape[0]
frac
```

Out[18]:
```
0    0.545426
1    0.454574
Name: default, dtype: float64
```

In [19]:
```
temp = upsampled_analy[upsampled_analy["EDUCATION"]==2]
frac = temp.default.value_counts()/temp.shape[0]
frac
```

Out[19]:
```
1    0.523534
0    0.476466
Name: default, dtype: float64
```

In [20]:
```
temp = upsampled_analy[upsampled_analy["EDUCATION"]==3]
frac = temp.default.value_counts()/temp.shape[0]
frac
# temp.count()
```

Out[20]:
```
1    0.54263
0    0.45737
Name: default, dtype: float64
```

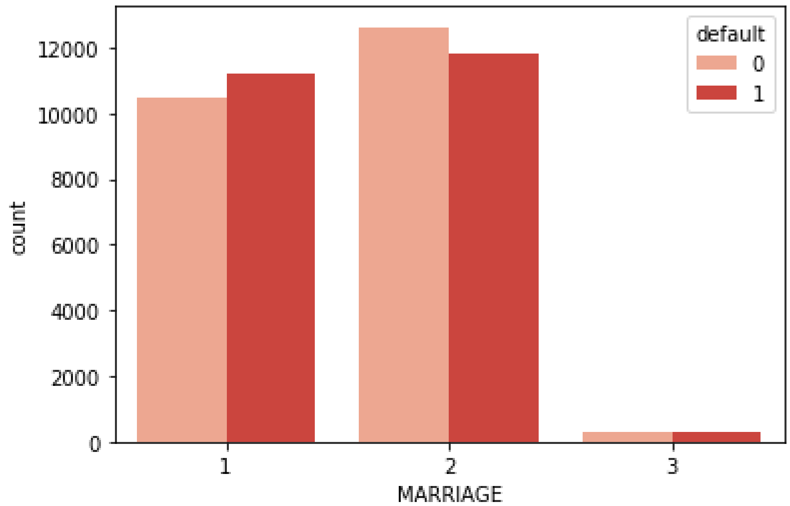
In [21]:
```
temp = upsampled_analy[upsampled_analy["EDUCATION"]==4]
frac = temp.default.value_counts()/temp.shape[0]
frac
# temp.count()
```

Out[21]:
```
0    0.778523
1    0.221477
Name: default, dtype: float64
```

In [22]:
```
temp = upsampled_analy[upsampled_analy["EDUCATION"]==5]
frac = temp.default.value_counts()/temp.shape[0]
frac
# temp.count()
```

Out[22]:
```
0    0.793532
1    0.206468
Name: default, dtype: float64
```

3.Marrital status:



Similar to the case of gender of the card holder there is not much variance in fraction of defaulters and non-defaulters across the categories [1,2,3]. We can assume that this feature does not contribute much to the classification model.

In [28]:
```
temp = upsampled_analy[upsampled_analy["MARRIAGE"]==1]
frac = temp.default.value_counts()/temp.shape[0]
frac
```

Out[28]:
```
1    0.517673
0    0.482327
Name: default, dtype: float64
```

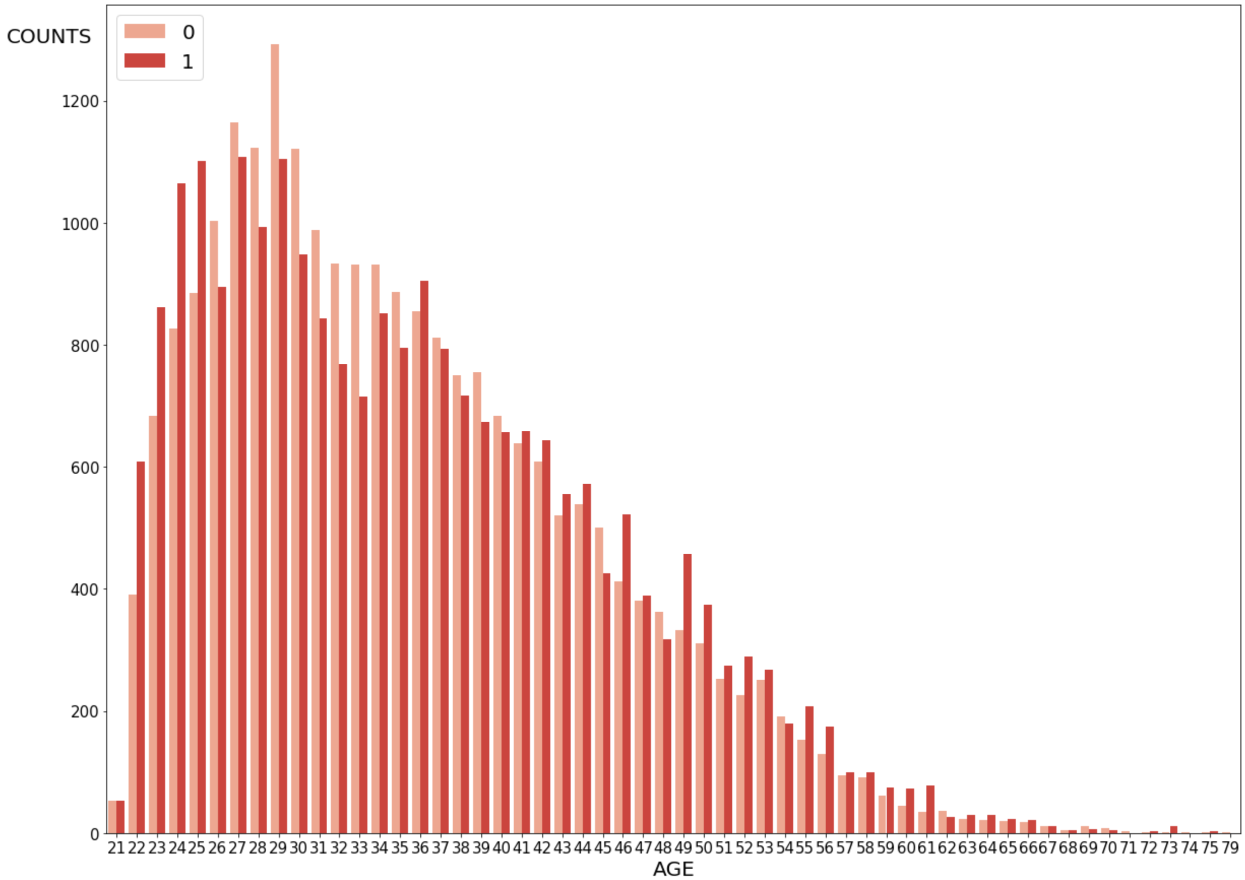
In [29]:
```
temp = upsampled_analy[upsampled_analy["MARRIAGE"]==2]
frac = temp.default.value_counts()/temp.shape[0]
frac
```

Out[29]:
```
0    0.516236
1    0.483764
Name: default, dtype: float64
```

In [30]:
```
temp = upsampled_analy[upsampled_analy["MARRIAGE"]==3]
frac = temp.default.value_counts()/temp.shape[0]
frac
```

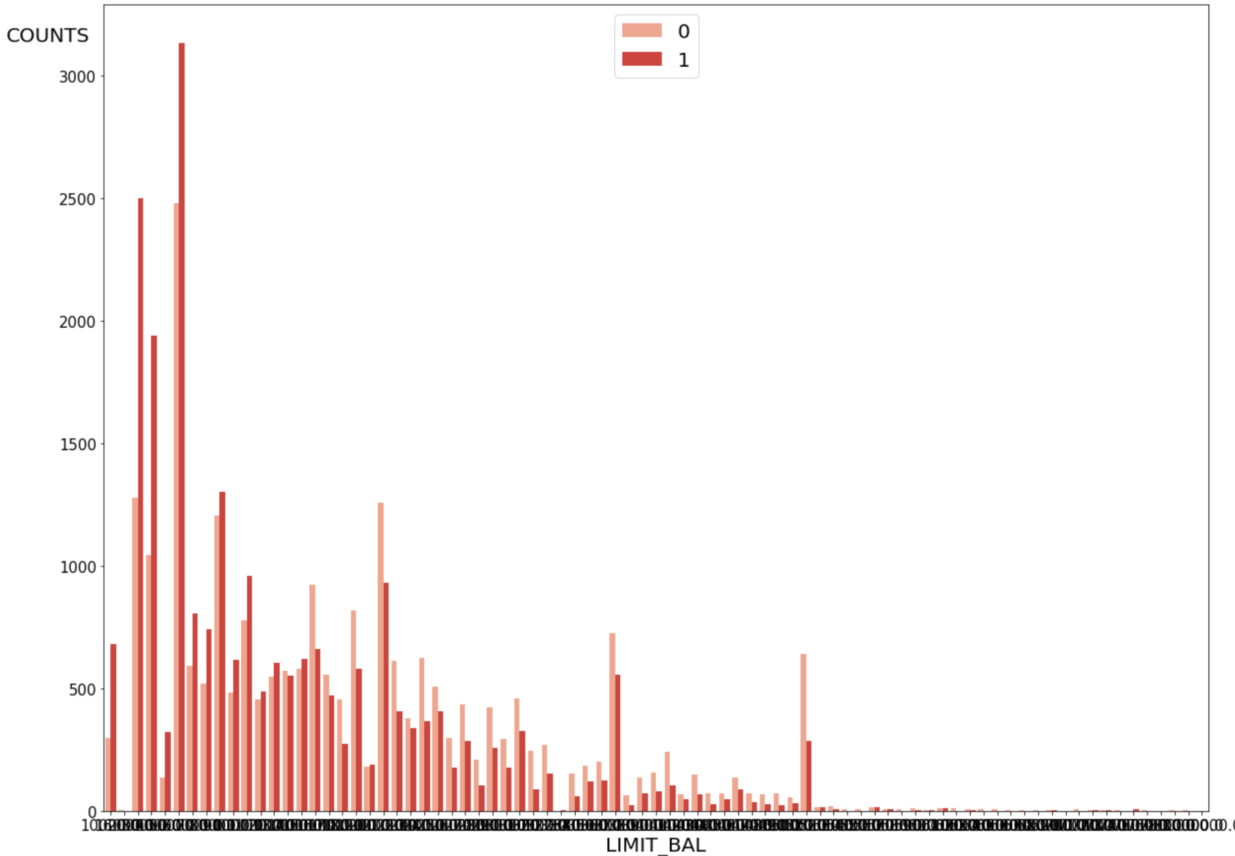
Out[30]:
```
1    0.523179
0    0.476821
Name: default, dtype: float64
```

4.Age:



The plot of frequency of “defaulters and non defaulters” across age is very similar. Using similar arguments as in case of “sex” or “marrital status”, we can say that this feature does not contribute largely to the classification model.

5.Amount of given credit(Limit Balance):



As we can see in the above image,the defaulters(default=1) are concentrated towards lower Limit Balance and frequency of “non-defaulters” dominates the “defaulters” towards the higher end of Limit Balance.Hence we say that Limit Balance is indeed a good feature.

If we have redundant features in a Random Forest, they will be given close weights and the actual useful features might be given lower weights. Hence it is important to drop redundant features.

6.BILL AMOUNTS:

In [63]:
```
new_temp = data.iloc[:,8:]
new_temp.corr()
```

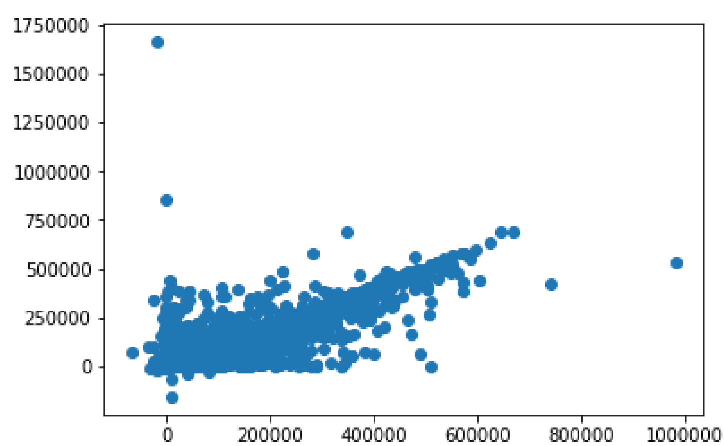
Out[63]:

	BILL_AMT1	BILL_AMT4	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6	default
BILL_AMT1	1.000000	0.860272	0.802650	0.140277	0.099355	0.156887	0.158303	0.167026	0.179341	-0.019644
BILL_AMT4	0.860272	1.000000	0.900941	0.233012	0.207564	0.300023	0.130191	0.160433	0.177637	-0.010156
BILL_AMT6	0.802650	0.900941	1.000000	0.199965	0.172663	0.233770	0.250237	0.307729	0.115494	-0.005372
PAY_AMT1	0.140277	0.233012	0.199965	1.000000	0.285576	0.252191	0.199558	0.148459	0.185735	-0.072929
PAY_AMT2	0.099355	0.207564	0.172663	0.285576	1.000000	0.244770	0.180107	0.180908	0.157634	-0.058579
PAY_AMT3	0.156887	0.300023	0.233770	0.252191	0.244770	1.000000	0.216325	0.159214	0.162740	-0.056250
PAY_AMT4	0.158303	0.130191	0.250237	0.199558	0.180107	0.216325	1.000000	0.151830	0.157834	-0.056827
PAY_AMT5	0.167026	0.160433	0.307729	0.148459	0.180908	0.159214	0.151830	1.000000	0.154896	-0.055124
PAY_AMT6	0.179341	0.177637	0.115494	0.185735	0.157634	0.162740	0.157834	0.154896	1.000000	-0.053183
default	-0.019644	-0.010156	-0.005372	-0.072929	-0.058579	-0.056250	-0.056827	-0.055124	-0.053183	1.000000

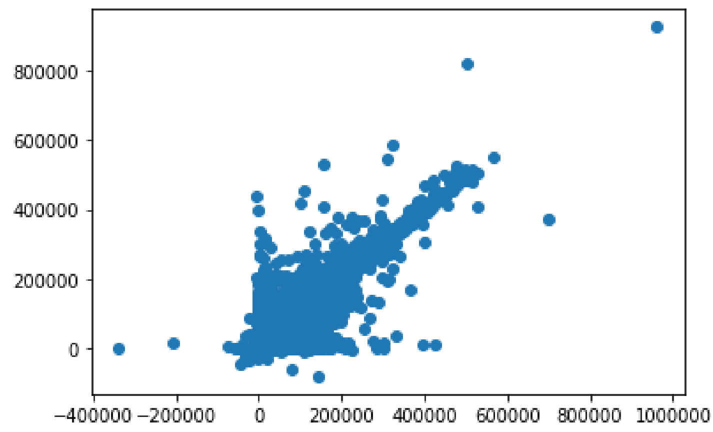
As we can see from above correlation matrix, there is high amount of correlation between columns 12:17(All of them correspond to bill statements).

From here we can conclude that :

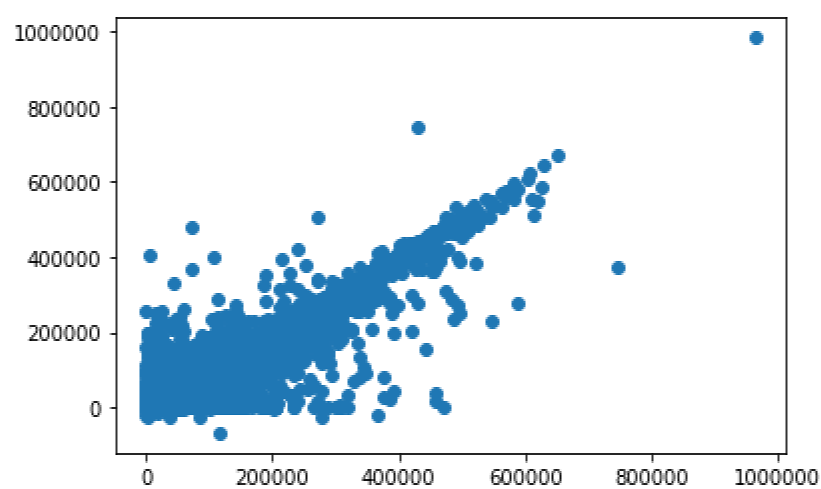
a) BILL_AMT2 and BILL_AMT3



b) BILL_AMT5 and BILL_AMT6

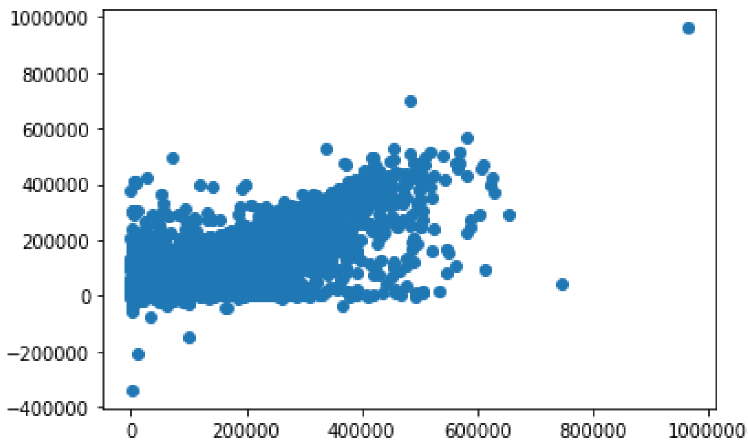


c)BILL_AMT2 and BILL_AMT1

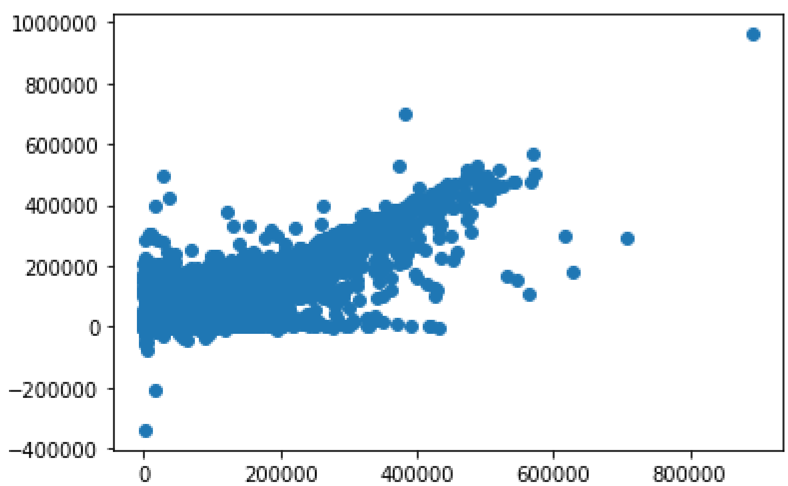


Here we see that they are highly correlated in a linear fashion. Hence they are redundant features and some of them can be dropped.

a)BILL_AMT1 and BILL_AMT6(plot 1)



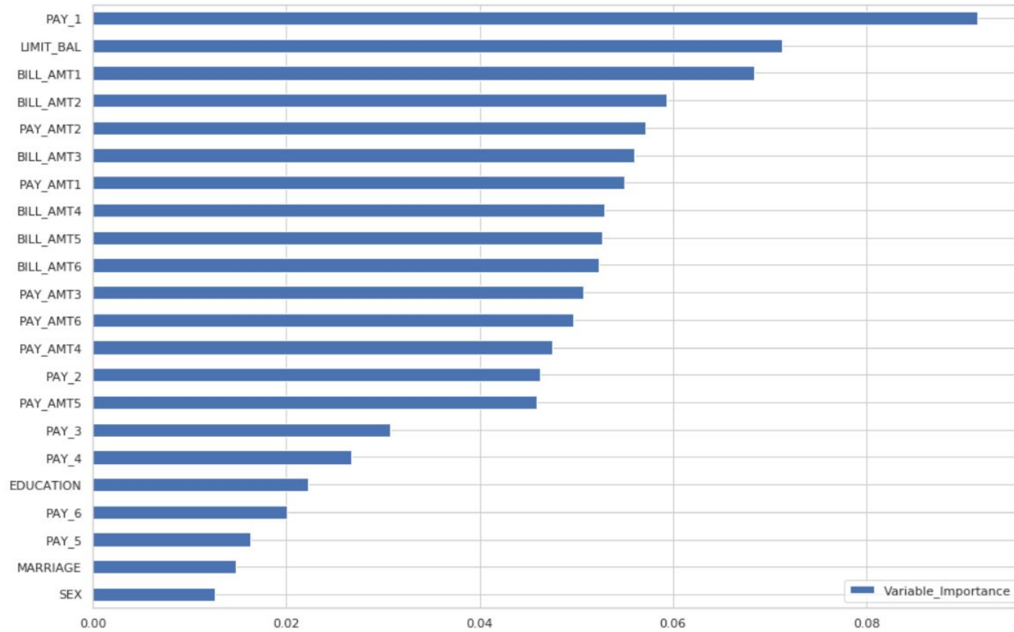
b)BILL_AMT4 and BILL_AMT6(plot 2)



We see there is no linear correlation between the above features and it is necessary to retain these features.

To illustrate the above points, we run a random forest classifier on the cleaned dataset (without dropping unnecessary features) and importance of each feature is plotted below.

In [50]:
```
feature_importance.plot.barh(figsize=(15,10))
```
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x7f57f4a31898>



In [62]:
```
feature_importance.head(30)
```
Out[62]:

	Variable_Importance
SEX	0.012626
MARRIAGE	0.014750
PAY_5	0.016270
PAY_6	0.020032
EDUCATION	0.022261
PAY_4	0.026712
PAY_3	0.030737
PAY_AMT5	0.045885
PAY_2	0.046246
PAY_AMT4	0.047487
PAY_AMT6	0.049701
PAY_AMT3	0.050720
BILL_AMT6	0.052304
BILL_AMT5	0.052650
BILL_AMT4	0.052944
PAY_AMT1	0.054940
BILL_AMT3	0.056059
PAY_AMT2	0.057190
BILL_AMT2	0.059294
BILL_AMT1	0.068437
LIMIT_BAL	0.071217
PAY_1	0.091539

Image 1

As said above,the weights of columns like Marrital status and Sex of the card holder do not contribute much and highly correlated features (correlated to important feature) have nearly same amount of weights.

So we have **dropped** the following features based on the above discussion:

a)BILL_AMT2

b)BILL_AMT3

c)BILL_AMT5

d)Marriage

e)Sex

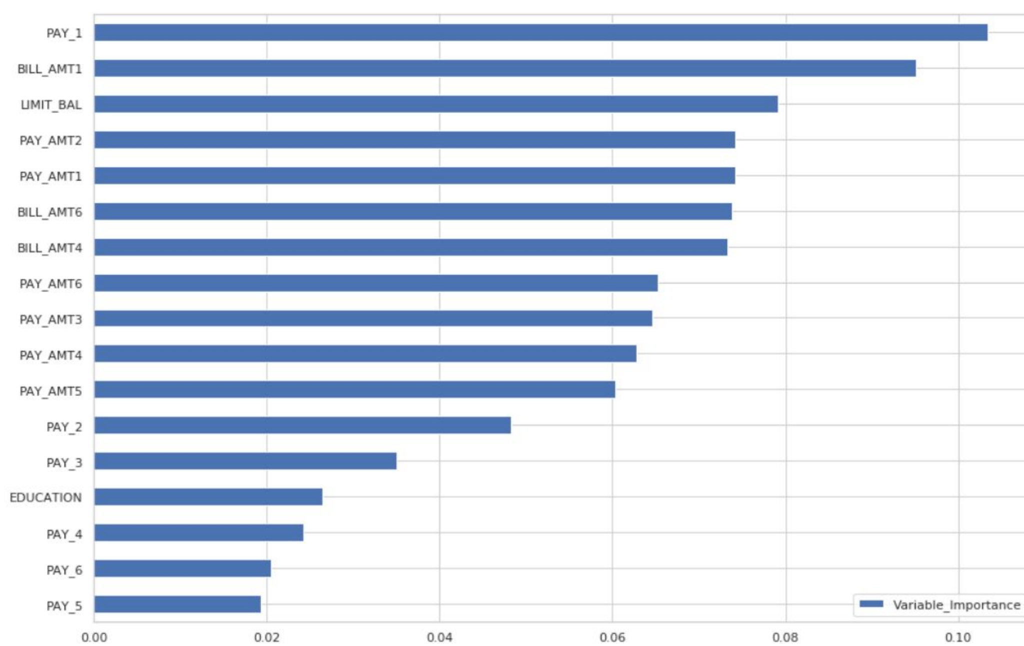
f)Age

Model Used:

As discussed above, it is important to correctly identify **“defaulters”** for which we have to optimize score_metrics like **“f1_score”** and **“recall**” rather than **“precision”**.

Models like Random Forest ,Logistic Regression and Gaussian NB were applied. **Random Forest** yielded the most accurate results.

```
In [53]: 1 feature_importance.plot.barh(figsize=(15,10))
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7f65c1e640b8>
```



```
In [65]: feature_importance.head(30)
Out[65]:
```

	Variable_Importance
PAY_5	0.019349
PAY_6	0.020480
PAY_4	0.024237
EDUCATION	0.026410
PAY_3	0.034987
PAY_2	0.048264
PAY_AMT5	0.060271
PAY_AMT4	0.062784
PAY_AMT3	0.064566
PAY_AMT6	0.065298
BILL_AMT4	0.073321
BILL_AMT6	0.073819
PAY_AMT1	0.074224
PAY_AMT2	0.074271
LIMIT_BAL	0.079120
BILL_AMT1	0.095104
PAY_1	0.103494

Image 2

The weights follow our initial observation,and allot more weights to columns like PAY_AMT

instead of redundant features like BILL_AMT. This can be seen by comparing weights for PAY_AMT features for the model where redundant columns are not dropped and now when they are dropped (Image 1 and Image 2). Therefore decisions are made on features that actually provide new information to the model.

The **accuracy, precision** and recall of the model after dropping features is shown below:

```
In [50]: print(metrics.classification_report(y_test,predictions_rf))
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      5864
           1       0.57      0.46      0.51      1636

    accuracy                           0.81      7500
   macro avg       0.71      0.68      0.69      7500
weighted avg       0.79      0.81      0.80      7500
```

```
In [51]: repo={"accuracy":metrics.accuracy_score(y_test,predictions_rf),"precision":metrics.precision_score(y_test,predi
         print(repo)
{'accuracy': 0.8070666666666667, 'precision': 0.5718631178707224, 'recall': 0.45965770171149145, 'f1_score': 0.509
6577431379193}
```

Which is an improvement over the application of model on the cleaned dataset with no dropped columns.(The main scoring metric for this problem is recall and f1_score)

In [37]:
```
print(metrics.classification_report(y_test,predictions_rf))
```
```
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      5864
           1       0.55      0.37      0.44      1636

    accuracy                           0.80      7500
   macro avg       0.69      0.64      0.66      7500
weighted avg       0.78      0.80      0.78      7500
```

In [38]:
```
repo={"accuracy":metrics.accuracy_score(y_test,predictions_rf),"precision":metrics.precision_score(y_test,predi
print(repo)
```
```
{'accuracy': 0.7964, 'precision': 0.5495004541326067, 'recall': 0.3698044009779951, 'f1_score': 0.442089879430032
9}
```

Model Tuning:

To further improve recall and f1_score,we need to modify decision threshold .The definitions of precision and recall are given below:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

It follows that there is a trade-off between precision and recall.In our case ,since it is critical to identify “defaulters”, we need a higher recall than precision.

This is done by selecting optimum value of decision_threshold from the plot of :

In [56]:

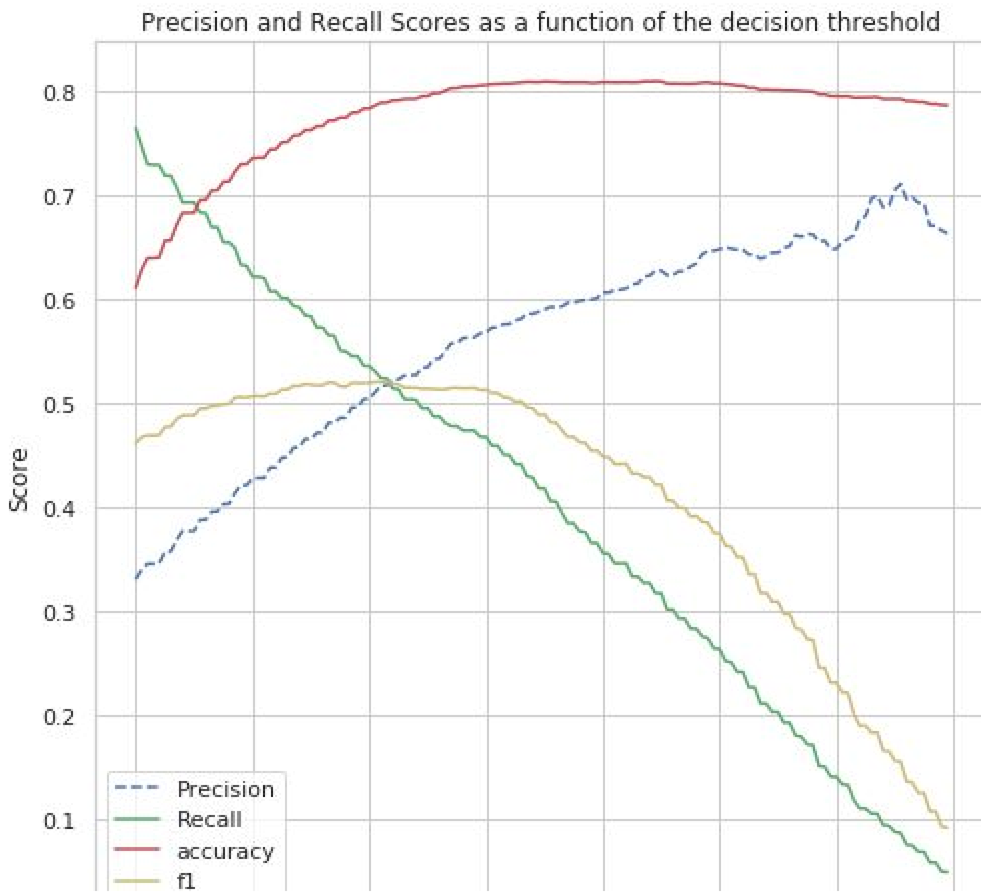
```python
predictions_rf_proba = rf.predict_proba(X_test)
```

In [57]:

```python
x1 = np.linspace(0.2,0.9,140, endpoint = False)
```

In [58]:

```python
accuracy = []
precision =[]
recall =[]
f1 =[]
for x in x1:
    predictions_rf=adjusted_classes(predictions_rf_proba[:,1],x)
    accuracy.append(metrics.accuracy_score(y_test,predictions_rf))
    precision.append(metrics.precision_score(y_test,predictions_rf))
    recall.append(metrics.recall_score(y_test,predictions_rf))
    f1.append(metrics.f1_score(y_test,predictions_rf))
```



Accuracy,precision,recall,f1_score against decision threshold .

The intersection of precision and recall is optimal point for maximizing f1_score. Since recall is a deciding factor than accuracy we select decision factor left to intersection point to increase the recall.

Final Result:

The corresponding values for Accuracy, precision, recall, f1_score are :

In [60]:
```
predictions_rf=adjusted_classes(predictions_rf_proba[:,1],0.33)
repo={"accuracy":metrics.accuracy_score(y_test,predictions_rf),"precision":metrics.precision_score(y_test,predi
print(repo)
```

```
{'accuracy': 0.7517333333333334, 'precision': 0.44849589790337285, 'recall': 0.6014669926650367, 'f1_score': 0.513
8381201044386}
```

In [61]:
```
print(metrics.classification_report(y_test,predictions_rf))
```

```
              precision    recall  f1-score   support

           0       0.88      0.79      0.83      5864
           1       0.45      0.60      0.51      1636

    accuracy                           0.75      7500
   macro avg       0.66      0.70      0.67      7500
weighted avg       0.78      0.75      0.76      7500
```