Machine Learning to predict Credit Card holder's payment behavior

ABSTRACT

Using various machine learning methodologies to predict the borrower's default status using borrower's information and payment behaviors. Used Clustering analysis to segment the portfolio and check if model prediction accuracy improves or not.

Deepak Kumar Tiwari

Introduction to Projects:

The goal of this project is to:

1. Check credit card holder's behaviors, trends and factors to identify default risks
2. Predicting if borrower is going to default in next month or not given all the borrower and payment information
3. Using cluster analysis to segment the portfolio so that we can improve the model prediction accuracy for each segment
4. Checking techniques to make a prediction
5. Assessing if clustering helps us to improve the model performance or not

Data:

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.Original Data can be found here:

Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Predictors in data:

```
ID                          int64
LIMIT_BAL                 float64
SEX                         int64
EDUCATION                   int64
MARRIAGE                    int64
AGE                         int64
PAY_0                       int64
PAY_2                       int64
PAY_3                       int64
PAY_4                       int64
PAY_5                       int64
PAY_6                       int64
BILL_AMT1                 float64
BILL_AMT2                 float64
BILL_AMT3                 float64
BILL_AMT4                 float64
BILL_AMT5                 float64
BILL_AMT6                 float64
PAY_AMT1                  float64
PAY_AMT2                  float64
PAY_AMT3                  float64
PAY_AMT4                  float64
PAY_AMT5                  float64
PAY_AMT6                  float64
```

Target variable:

```
default.payment.next.month      int64
```

Our target variable is binary type, it's 1 if borrower is going to default in next month and 0 if not.

Variable Definition:

Here is information about all the variables we have
There are 25 variables:

1. ID: ID of each client
2. LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit
3. SEX: Gender (1=male, 2=female)
4. EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
5. MARRIAGE: Marital status (1=married, 2=single, 3=others)
6. AGE: Age in years
7. PAY_0: Repayment status in September, 2005 (-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)
8. PAY_2: Repayment status in August, 2005 (scale same as above)
9. PAY_3: Repayment status in July, 2005 (scale same as above)
10. PAY_4: Repayment status in June, 2005 (scale same as above)
11. PAY_5: Repayment status in May, 2005 (scale same as above)
12. PAY_6: Repayment status in April, 2005 (scale same as above)
13. BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
14. BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
15. BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
16. BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
17. BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
18. BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
19. PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
20. PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
21. PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
22. PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
23. PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
24. PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
25. default.payment.next.month: Default payment (1=yes, 0=no)

Data Preparation:

1. We don't have any missing values or duplicates values but if we had we might have to :
2. Remove entirely duplicate values after making sure no information is lost
3. Imputations for missing values after investigating reasons for missing values and what statistical method can imitate actual values.
4. In some cases, separate models are made for missing data because of similar characteristics.

Data Explorations:

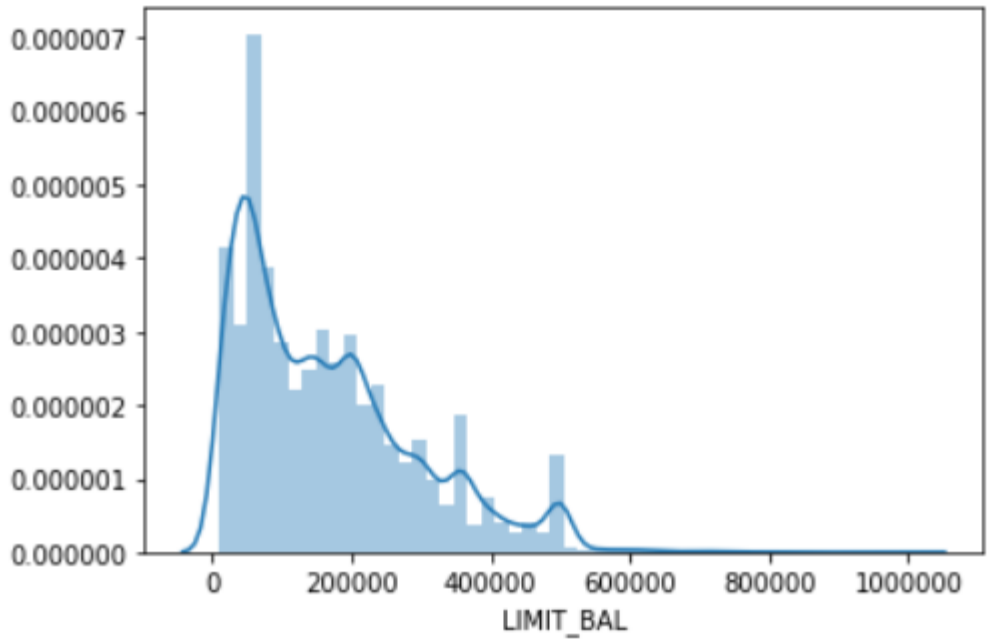
Let's explore one by one different set of information:

Limit Balance:

	LIMIT_BAL							
	count	mean	std	min	25%	50%	75%	max
SEX								
1	11888.0	163519.825034	136250.481355	10000.0	50000.0	130000.0	240000.0	800000.0
2	18112.0	170086.462014	125231.777960	10000.0	70000.0	150000.0	240000.0	1000000.0

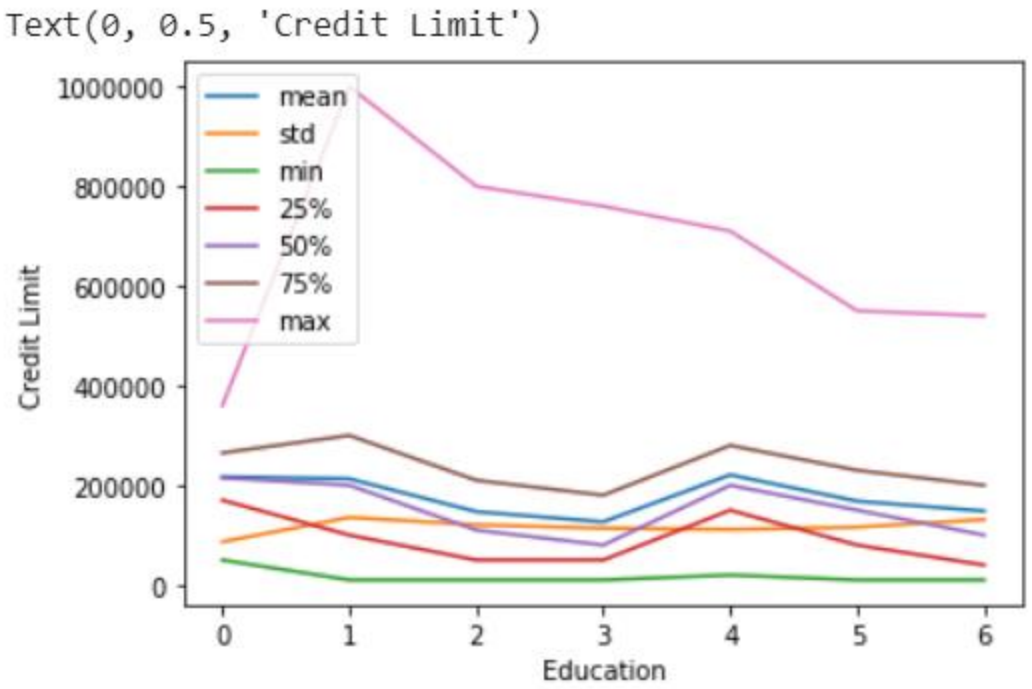
So, we have 11,888 accounts of male and 18,112 Female accounts. Credit limit maximum, average, first quartile, median is more for Females than males. This gives good information about the market there. Females are either applying for more credit cards or getting approved, getting more maximum limits and average limits than their male counterparts

<matplotlib.axes._subplots.AxesSubplot at 0x7f0acd7e2908>



Education :

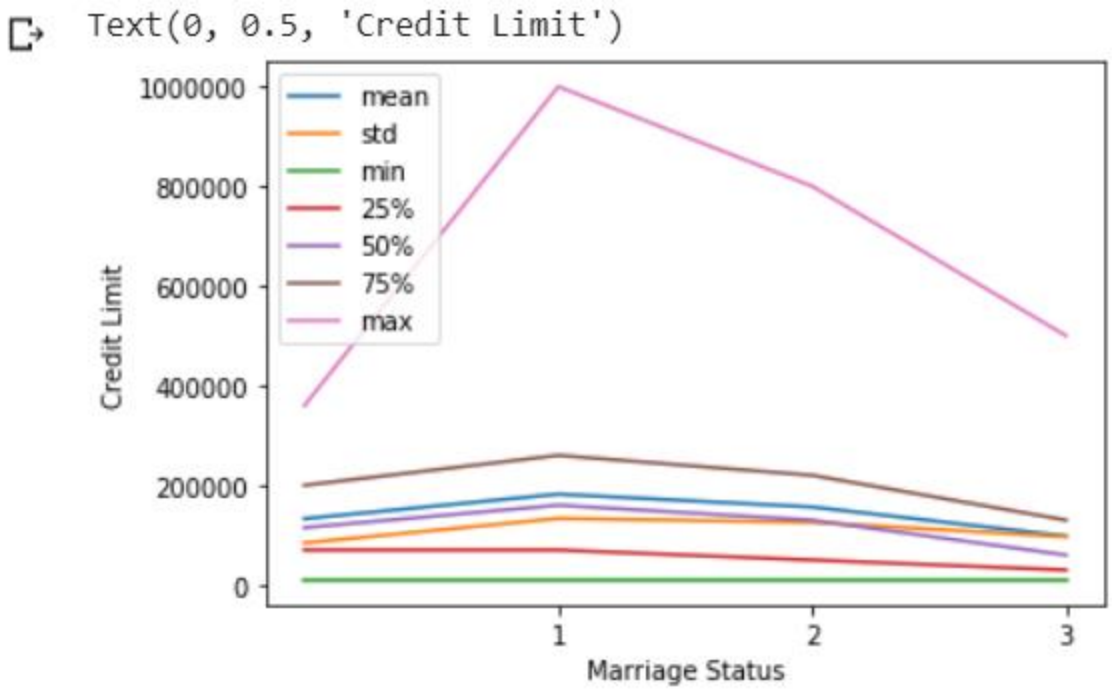
	LIMIT_BAL							
	count	mean	std	min	25%	50%	75%	max
EDUCATION								
0	14.0	217142.857143	86596.195638	50000.0	170000.0	215000.0	265000.0	360000.0
1	10585.0	212956.069910	135474.936730	10000.0	100000.0	200000.0	300000.0	1000000.0
2	14030.0	147062.437634	120672.282033	10000.0	50000.0	110000.0	210000.0	800000.0
3	4917.0	126550.270490	113979.322678	10000.0	50000.0	80000.0	180000.0	760000.0
4	123.0	220894.308943	111531.868048	20000.0	150000.0	200000.0	280000.0	710000.0
5	280.0	168164.285714	116036.276043	10000.0	80000.0	150000.0	230000.0	550000.0
6	51.0	148235.294118	131037.489023	10000.0	40000.0	100000.0	200000.0	540000.0



We have following definitions for EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown). We can see graduate school education level has more mean, first quartile, 2nd quartile, median and maximum credit limit than the university or high school education. Credit Limit certainly has direct correlation with education. More is the education, more is the credit limit.
Here, 0,5,6 are unknown education accounts. While education 0 has the highest minimum credi t limit and maximun

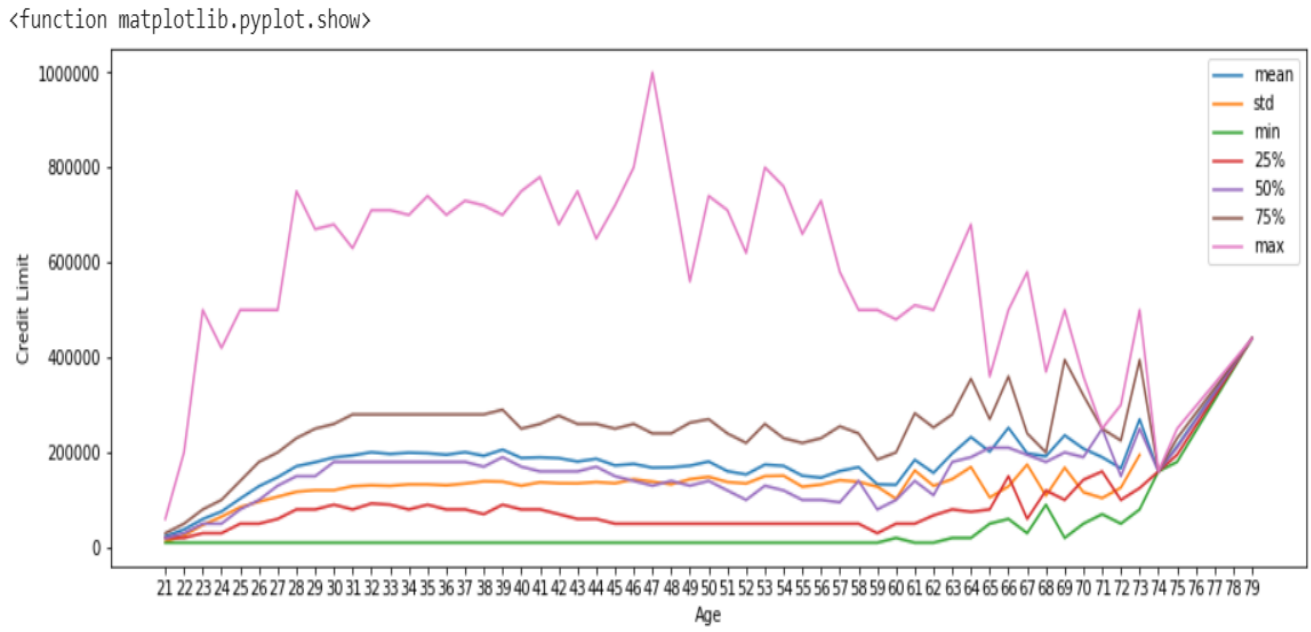
Marriage Status:

	LIMIT_BAL							
	count	mean	std	min	25%	50%	75%	max
MARRIAGE								
0	54.0	132962.962963	84331.547615	10000.0	70000.0	115000.0	200000.0	360000.0
1	13659.0	182200.893184	133382.262668	10000.0	70000.0	160000.0	260000.0	1000000.0
2	15964.0	156413.660737	125673.426316	10000.0	50000.0	130000.0	220000.0	800000.0
3	323.0	98080.495356	96542.879913	10000.0	30000.0	60000.0	130000.0	500000.0

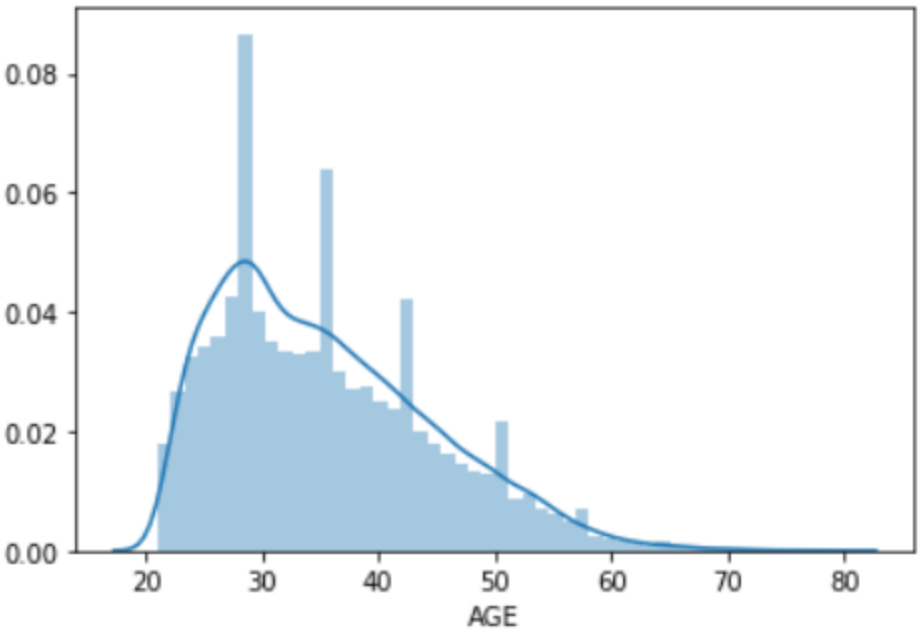


We have this information for MARRIAGE: Marital status (1=married, 2=single, 3=others). Clearly Singles either apply for more credit cards or they get more approval. However, a greater number of accounts doesn't mean more credit limit. Married people are offered more credit limit both in terms of average, quartiles or maximum credit limit. We again have 0 status as marriage column. Firm would want to find out why marriage status is 1 despite of having single, married and others. these values can be missing values which has just been imputed

AGE:



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0acbde8eb8>
```

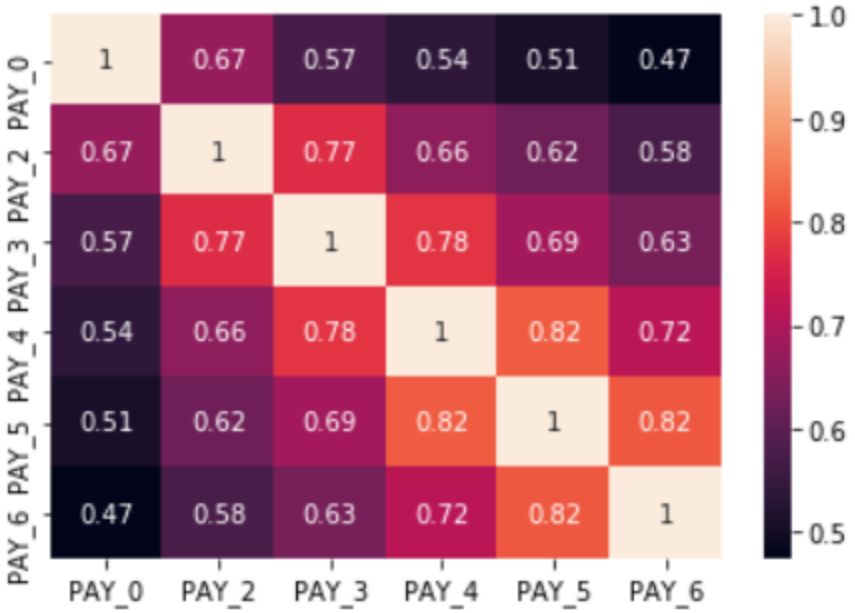


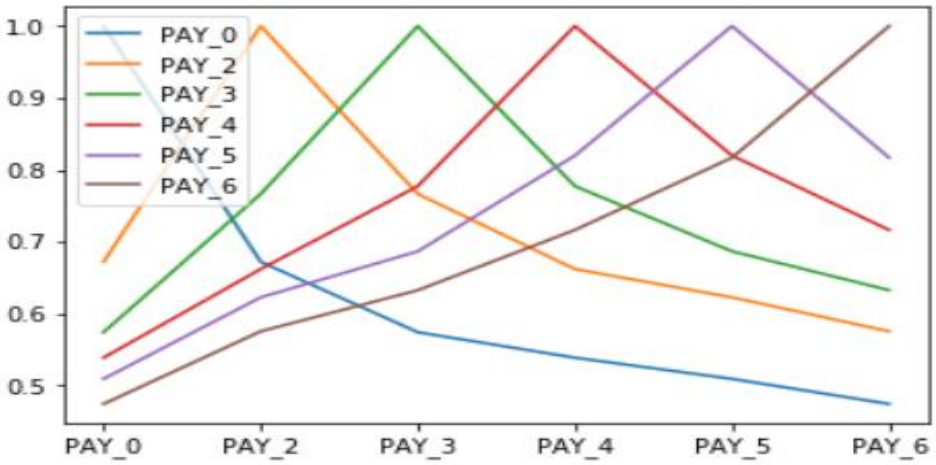
We can see how age group from 28 to 57 has almost all mean, max and min credit limit higher than the other age group.

PAYMENT STATUS:

This is the definition of this variable PAY_0: Repayment status in September, 2005 (-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). Let's see payment status for the month of April first. This is the first-time payment status recorded for above borrowers. Then we would try to find how many defaulters from April default in May following June, July, August and September.

This is the correlation matrix between payment status:





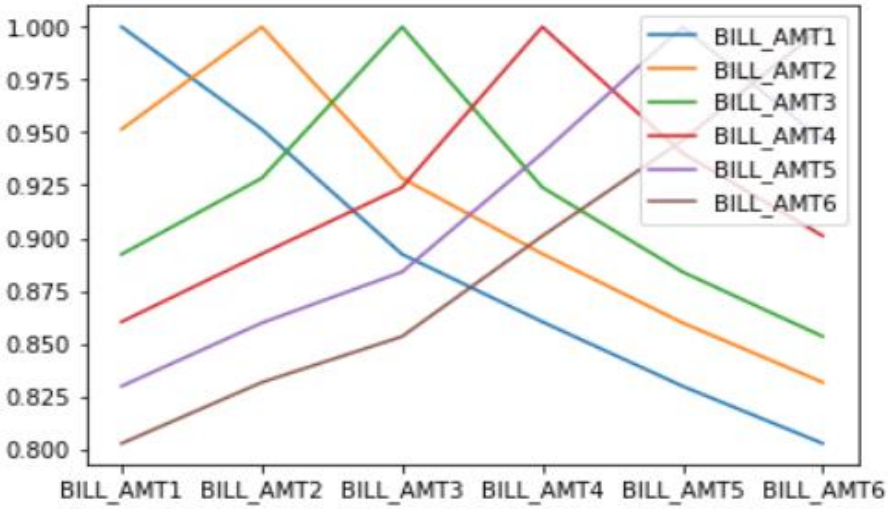
In above heat map we can see that there is strong correlation between Payment status in April and May. Similarly, little less correlation with June, July, August and minimum for September. Correlation between payment of previous month and next months are high but as months goes pass by, correlations decreases. Hence, for the month of September, payment status of August, July, June , May and April matters subsequently in decreasing order.

These are the payment status:

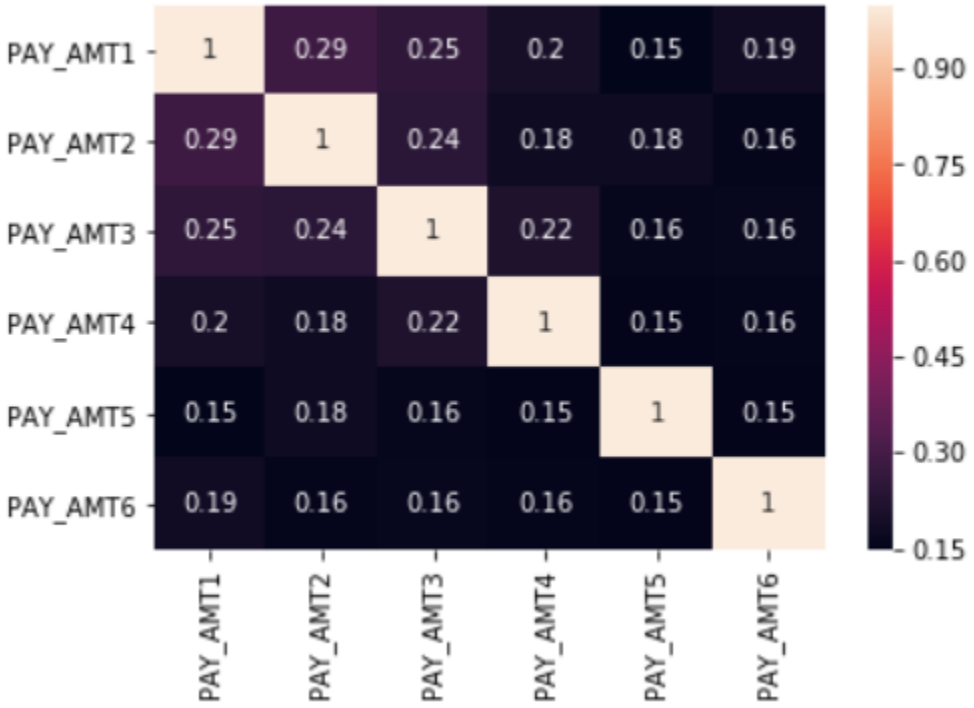
	PAY_0
PAY_2	
-2	3782
-1	6050
0	15730
1	28
2	3927
3	326
4	99
5	25
6	12
7	20
8	1

So, upon research and reading discussions, I found out that -2,-1,0 all three-payment status owes nothing to the bank and hence, they have been paying their dues on time. There are 25562 accounts who paid their dues in August paid their bills on time in September as well. There are people who didn't paid for 1 months but paid in August but again didn't paid in September. Thus here 28 accounts who didn't paid in July, paid in August and then again didn't paid in September. These are indeterminate customers and many banks make separate models to predict payment behavior of these customers.

There are 3927 accounts who didn't paid in August as well as September. So, it is quite evident that if card holder doesn’t pay in 1st month then there are more chances of defaulting in 2nd month as well and so on.

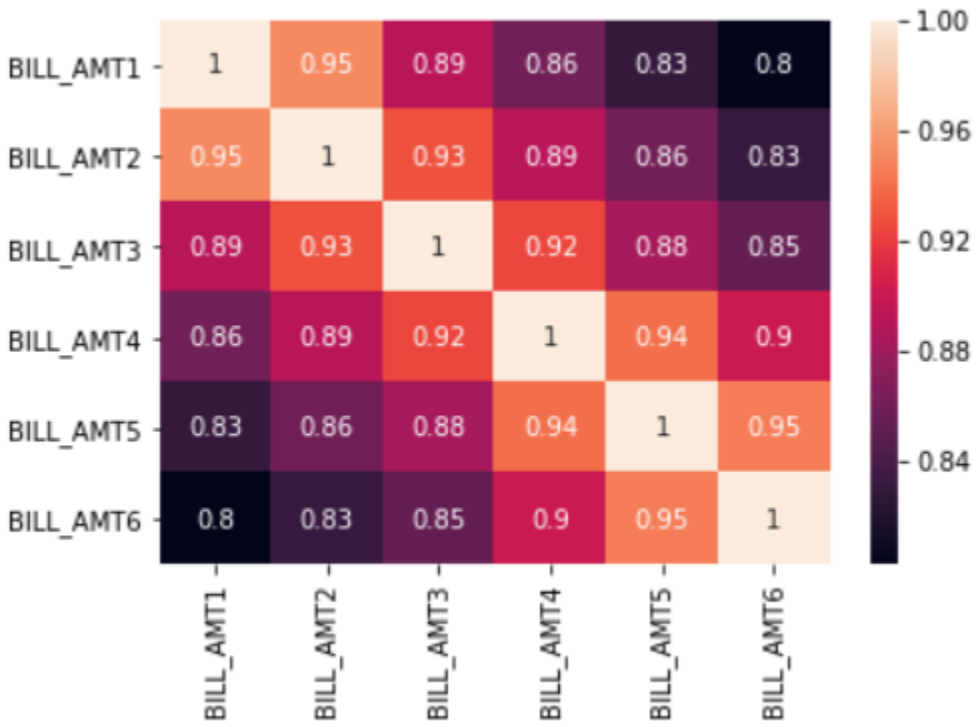


PAYMENT AMOUNT:



We can see that; we have less correlation with payment amount between months because of unexpected income or bill amount in months. However, there are still some correlations between months but not that much as it is between bill amount and payment status.

BILL AMOUNT:



Behaviors in card holders can be inferred clearly through above plot and heatmap. Bill amount in September is 95% correlated with the bill amount in August, 89% correlated with July amount, 86% with June and as months passes by the correlation decreases. So, it indicates that bill amount goes on increasing because of nonpayment. In other terms if bill amount is high in one month then there is more chance that borrower will not pay in full and leading to higher bill amount in subsequent months

METHODOLOGIES:

Standardization:

As each variable are of different scale this can be a problem and might not give expected result. Standardization makes mean of variable as 0 and standard deviation 1.

Logistic Regression:

Sigmoid function which separates log of odds linearly. It also results into probability of odd and if odd is greater than threshold then it's the target variable(In case of Binary, 0 or 1). At this step we performed logistic regression on overall data without segmenting.

This was the performance on 10-fold cross validation, we have maximum accuracy of 82.8%.

Random Forest: Random Forest tries to make various number of decision trees by picking variables randomly, one tree should be not correlated with another and checking entropy and information value to classify. There are some hyperparameters like number of tress and depth. Let's see how random forest did on original data according to number of trees:

```
[ ]  tree_lst=[5,10,15,20,25,30,35,40,45,50]
     for i in range(len(tree_lst)):
         classifier_rf=RandomForestClassifier(n_estimators=tree_lst[i], criterion='entropy')
         classifier_rf.fit(X,Y)
         accuracies_rf=cross_val_score(estimator=classifier_rf,X=X,y=Y,cv=10)
         print(accuracies_rf)
```

```
[0.78573809 0.77874042 0.77974009 0.77940686 0.79       0.78666667
 0.80460153 0.7975992  0.79726576 0.79193064]
[0.80139953 0.78940353 0.80106631 0.79473509 0.80666667 0.81
 0.81660554 0.81160387 0.81160387 0.8066022 ]
[0.79840053 0.80373209 0.80006664 0.79840053 0.81       0.80966667
 0.82727576 0.81960654 0.81627209 0.8096032 ]
[0.80006664 0.80706431 0.80739753 0.79773409 0.81766667 0.81766667
 0.82594198 0.81927309 0.82694231 0.81427142]
[0.80373209 0.79773409 0.80173276 0.79706764 0.80866667 0.814
 0.8276092  0.82327442 0.82194065 0.81727242]
[0.80306564 0.80639787 0.80939687 0.80073309 0.81666667 0.82333333
 0.8336112  0.82360787 0.81893965 0.8156052 ]
[0.80373209 0.80139953 0.80873042 0.80006664 0.81733333 0.81333333
 0.83561187 0.82227409 0.82327442 0.8186062 ]
[0.80073309 0.80573142 0.81006331 0.80373209 0.81333333 0.81733333
 0.83027676 0.82694231 0.81893965 0.81427142]
[0.80039987 0.80606465 0.81106298 0.80206598 0.81766667 0.82233333
 0.83227743 0.82494165 0.81960654 0.81360453]
[0.80339887 0.80573142 0.81072976 0.80473176 0.81366667 0.826
 0.82627543 0.82394131 0.82060687 0.81693898]
```

We can see we are getting maximum accuracy of 83.56% when number of trees are 40. This is even better than Logistic Regression.

Neural Network:

This is Sequential Feed Forward Neural Network model with input layer as number of variables of 23, hidden layer of 10 and output layer containing 2 neurons since we have only two output, 0 or 1.

Activation function used is Rectified Linear unit which gives 0 for negative and 0 values and 1 for positive values.

Epoch size is 50 and batch size is 256, that means all data will go through the model 50 times in the size of 256. Model will learn each time batch go through the input, hidden layer and learn from it. It will try to reduce the loss or say inaccuracies by learning from each batch. Here loss is optimized using Stochastic Gradient Descent, which is basically tries to achieve global minimum loss by applying learning rate.

```
number_of_neurons_layer1 = 23
number_of_neurons_layer2 = 10
number_of_neurons_layer3 = 2
number_of_epochs = 50
```

```
dim =  23
samples = 30000
```

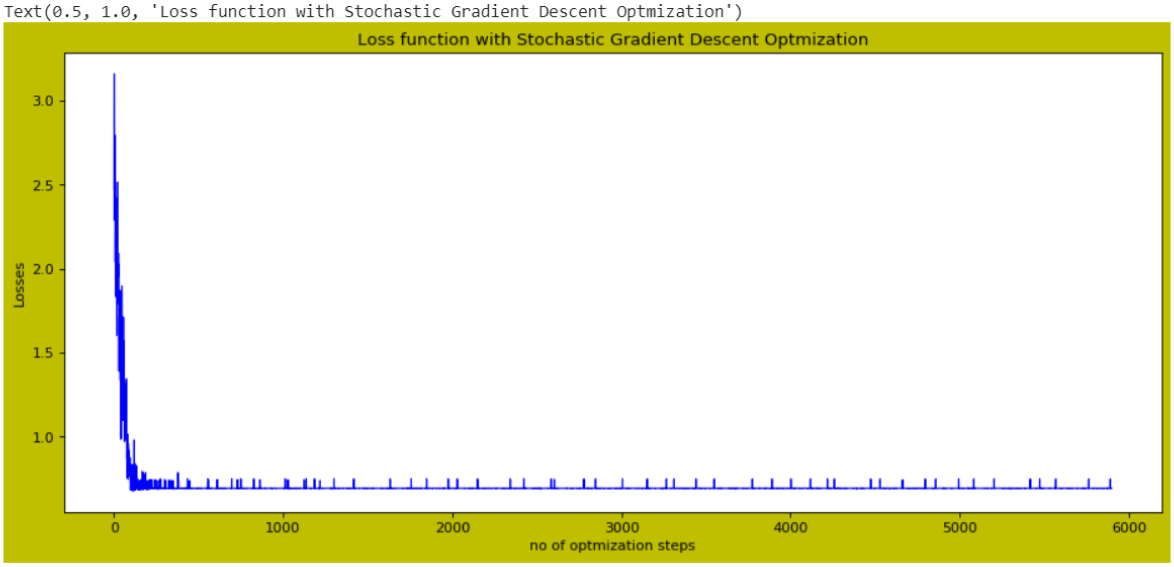
```
# design network
from keras import optimizers
sgd = optimizers.SGD(lr=0.01, clipnorm=1.)

model = Sequential()
model.add(Dense(number_of_neurons_layer1,input_shape=(dim, ), activation='relu'))
model.add(Dense(number_of_neurons_layer2, activation='relu'))
model.add(Dense(number_of_neurons_layer3, activation='relu'))
model.compile(loss='sparse_categorical_crossentropy', optimizer=sgd)

def train(data,label):
    model.fit(data, label, epochs=number_of_epochs, batch_size=256, validation_data=(data, label), verbose=0, shuffle=True,callbacks=[lr])

def score(data):
    return model.predict(data)
```

```
Text(0.5, 1.0, 'Loss function with Stochastic Gradient Descent Optmization')
```



We can see how loss started from over 3 and then decreased to less than 1 and then remained constant that means it has achieved its global minimum loss and can not be reduced further unless something is changed in data or methodologies. It gives accuracy of 99.32% which is great when compared to Random Forest and Logistic Regression

```
[ ] min(lr.losses)
```

```
0.6769031
```

```
[ ] accuracy=100-min(lr.losses)
```

```
[ ] accuracy
```

```
99.32309687137604
```

Clustering:

K-means Clustering(K-Means Clustering) and Density Based Scanning Clustering(DBSCAN):

K-Means Clustering:

Specified No. of clusters as 4 because of size of the data as 30000 only. If the size would have been more, we could have considered more than 4 given it improves the accuracy in each of them.

1**. n_clusters** : int, optional, default: 8 but here it's 4
The number of clusters to form as well as the number of centroids to generate.
2. i**nit** : {'k-means++', 'random' or an ndarray}
Method for initialization, defaults to 'k-means++': 'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.'random': choose k observations (rows) at random from data for the initial centroids. If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.
3. **n_init** : int, default: 12
Number of time the k-means algorithm will be run with different centroid seeds. The results will be the best output of n_init consecutive runs in terms of inertia.

Usually n_clusters and n_init are tuned to form clusters so, that it improves the model performance in our clusters.

This gives 4 clusters having 2888, 11206,12338 and 3568 points. We can see for cluster 0 we have mean credit limit of 83452.21 and so on for cluster 1,2,3. We have similar statistics for each of the 23 predictors.

	LIMIT_BAL		
	count	mean	std
Clus_km			
0	2888.0	83452.216066	73432.544244
1	11206.0	173687.488845	129598.755230
2	12338.0	148015.885881	119797.195925
3	3568.0	283340.156951	119345.902653

DBSCAN:

This is density-based clustering and try to make clusters based on distance and number of points. I have defined distance as 0.15 and minimum number of points as 10. DBSCAN tries to form clusters within that distance and requires to have at least 10 points in it.

	LIMIT_BAL		
	count	mean	std
db_clus_label			
-1	29835.0	167955.075582	129816.849740
0	16.0	51250.000000	3415.650255
1	14.0	200714.285714	4746.311465
2	30.0	21333.333333	3457.459036
3	12.0	81666.666667	3892.494721
4	13.0	20000.000000	0.000000
5	27.0	204444.444444	5773.502692
6	19.0	20526.315789	2294.157339
7	14.0	22142.857143	6992.932068
8	10.0	11000.000000	3162.277660
9	10.0	175000.000000	8498.365856

Here, we have 11 clusters with almost 99.5% data in cluster -1, as such we can consider other accounts as outliers and can be removed from model consideration.

Model Performance after Clustering:

After clustering I have performed Logistic Regression, Random Forest and Feed Forward Neural Network model again to see if clustering improves model accuracy on any of the segmented portfolio or not.

After DBSCAN Clustering:

We checked the model accuracy in only -1 cluster since it contained 99.5% of our data. As expected, model performance was almost same as it contained almost all data and thus had same information and variation as over all data

```
array([0.80428954, 0.80495979, 0.80864611, 0.80663539, 0.81903485,
       0.82774799, 0.82908847, 0.82433791, 0.82595573, 0.82327297])
```

After K-means Clustering:

We can see performance decreased for cluster 0 to 65% almost, increased in cluster 1 as 83.75%. Performance increased on cluster 2 to 85%, performance increased on cluster 3 to 88.5% almost.

Logistic Regression Performance:

Cluster 0:

```
accuracies_kmeans
```

```
array([0.63793103, 0.63793103, 0.66089965, 0.62283737, 0.62975779,
       0.6366782 , 0.6875    , 0.63194444, 0.64930556, 0.62847222])
```

Cluster 1-3:

```
[ ] LR_kmeans_1 = LogisticRegression(C=0.01, solver='liblinear').fit(X_kmeans_1,Y_kmeans_1)
    accuracies_kmeans_1=cross_val_score(estimator=LR_kmeans_1,X=X_kmeans_1,y=Y_kmeans_1,cv=10)
    accuracies_kmeans_1
```

```
array([0.81729055, 0.81712756, 0.8073149 , 0.81445138, 0.82426405,
       0.8375    , 0.83214286, 0.82589286, 0.82410714, 0.81428571])
```

```
[ ] LR_kmeans_2 = LogisticRegression(C=0.01, solver='liblinear').fit(X_kmeans_2,Y_kmeans_2)
    accuracies_kmeans_2=cross_val_score(estimator=LR_kmeans_2,X=X_kmeans_2,y=Y_kmeans_2,cv=10)
    accuracies_kmeans_2
```

```
array([0.83319838, 0.83238866, 0.83319838, 0.83144246, 0.8452188 ,
       0.84428224, 0.84428224, 0.85077048, 0.8459043 , 0.83941606])
```

```
[ ] LR_kmeans_3 = LogisticRegression(C=0.01, solver='liblinear').fit(X_kmeans_3,Y_kmeans_3)
    accuracies_kmeans_3=cross_val_score(estimator=LR_kmeans_3,X=X_kmeans_3,y=Y_kmeans_3,cv=10)
    accuracies_kmeans_3
```

```
array([0.8547486 , 0.8603352 , 0.85994398, 0.83753501, 0.8767507 ,
       0.87394958, 0.8511236 , 0.86516854, 0.88483146, 0.87640449])
```

Random Forest Performance on Clusters:

Cluster 0:

```
[0.62068966 0.63103448 0.65051903 0.62629758 0.63321799 0.65051903
 0.68402778 0.63194444 0.66319444 0.62152778]
```

Cluster 1:

```
[0.81996435 0.79928635 0.80196253 0.8117752  0.82426405 0.82410714
 0.81428571 0.81964286 0.81428571 0.8125    ]
```

Cluster 2:

```
[0.8291498  0.82834008 0.83319838 0.82982172 0.83792545 0.83941606
 0.84833739 0.84995945 0.84022709 0.84184915]
```

Cluster 3:

```
[0.84916201 0.84916201 0.85154062 0.83753501 0.86554622 0.85434174
 0.87078652 0.85674157 0.86235955 0.87359551]
```

We can see results are quite same as logistic regression with maximum of 87% on cluster 4 , little lower than 88.5% in case of logistic regression.

Neural Network Model Performance:

Cluster 0:

```
[ ] accuracy_0=100-min(lr.losses)
    accuracy_0
```

```
99.31647902727127
```

Cluster 1:

```
[ ] accuracy_1=100-min(lr.losses)
    accuracy_1
```

```
99.31226754188538
```

Cluster 2:

```
[ ] accuracy_2=100-min(lr.losses)
    accuracy_2
```

```
99.3207157254219
```

Cluster 3:

```
[ ] accuracy_3=100-min(lr.losses)
    accuracy_3
```

```
99.30685251951218
```

We can see how Neural Network performance has been almost constant and same as ther unclustered data performance. This is because model has learnt almost everything reached the global optimum. But yes if our data size would have been much larger then might be we could not have been able to run 50 epoch size on very large datasets in that case, model might not have reached global optimum and clustering could have helped in increasing performance.

Conclusion:

1. For credit card portfolio, payment status or payment probabilities are highly correlated with previous few months payment status, say 2 or 3 months.

2. In our project gender can not be successful criteria to segment the portfolio because both showed both payment behaviors.

3. Younger generation uses credit card more than the elder people and that's why age distribution is left shifted

4. Random Forest and Logistic Regression both are equally good in predicted binary classified model and there can be a chance to improve it further by properly optimizing the methodologies. It can be a good option if some one their model to be interpretable and little simple.

5. Neural Network beats almost all the model in terms giving prediction accuracy but is complex and thus may not be good choice from regulators and guidelines point of view.

6. K-Means clustering can segment the data so, that model accuracy increases. In one of our cluster, model accuracy declined, and we may need to transform that cluster to improve the accuracy. Please note that K-means clustering can be optimized to achieve optimal number of clusters.

7. DBSCAN was not suitable for this data as almost all the data were closely related but it can be used to remove outliers. DBSCAN can also be optimized using tuning distance and minimum data points hyperparameter

8. Segmenting portfolio is redundant if data size is small or not very large because Neural Network learn everything given number of hidden layers, activation function, epoch and batch size are chosen wisely.

9. Cross Validation is very good technique to know optimum accuracy of any model given appropriate number of folds are chosen.

Reference:

Link to the project:

https://colab.research.google.com/drive/19S5RvzR4BoJhSQa8C_s_9rujVLkxyADc