Name : Wahyu Adi P Nainggolan
Student : Del Institute of Technology

Data Science Fulltime Test

1. Description Test

The purpose of this test is to find out the right algorithm that suit with the data. This algorithm will give the best accuration of the data. To improve the acccuration there are a few processes that have to do before. The processes will be explained in this document.

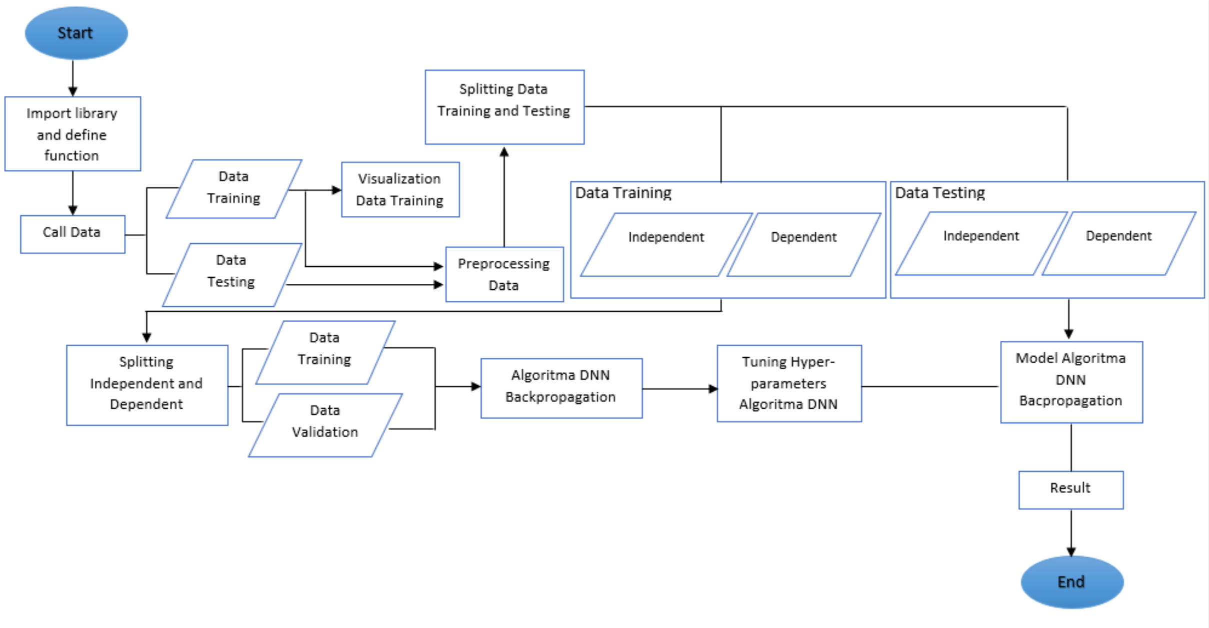
2. Description of the dataset

There are types of data, training and test. Both of the data have the same attributes but different function. Data training will train the data with the algorithm and build the model and data test will test the model with the algorithm. in this project there are 11 attributes in the dataset. There are 2 types of variables in the dataset. **SeriousDlqin2yrs** attribute is the dependent variable and the other is independent variable.
The description of the attributes can be seen in table below

Variable Name	Description	Type
SeriousDlqin2yrs	**Person experienced 90 days past due delinquency or worse**	**Y/N**
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits	percentage
age	Age of borrower in years	integer
NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days past due but no worse in the last 2 years.	integer
DebtRatio	Monthly debt payments, alimony,living costs divided by monthy gross income	percentage
MonthlyIncome	Monthly income	real
NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards)	integer
NumberOfTimes90DaysLate	Number of times borrower has been 90 days or more past due.	integer
NumberRealEstateLoansOrLines	Number of mortgage and real estate loans including home equity lines of credit	integer
NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days past due but no worse in the last 2 years.	integer
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.)	integer

3. System Design and Implementation

In this section will be explained the prediction processes :



For doing implementation, i use IDE in Anaconda is that Jupyter Notebook. The programming language that is the Python programming language.

1. Import the library and define all the functions that will be used

In [1]:
```
import pandas as pd
from fancyimpute import SoftImpute,KNN
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, Imputer,StandardScaler,Normalizer
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score, train_test_split
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
from keras.layers import Dense, Dropout
%matplotlib inline
```

```
Using TensorFlow backend.
```

In [2]:
```
def plot_accuracy_against_epoch(model):
    plt.plot(model.history['acc'])
    plt.plot(model.history['val_acc'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.show()
def plot_loss_against_epoch(model):
    plt.plot(model.history['loss'])
    plt.plot(model.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'validation'], loc='upper right')
    plt.show()
```

2. Call data training and data testing

In [3]:
```
dataset_train=pd.read_csv("C:/Users/Wahyu Nainggolan/ronde_2/Data/cs-training.csv").drop("Unnamed: 0",axis=1)
dataset_test=pd.read_csv("C:/Users/Wahyu Nainggolan/ronde_2/Data/cs-test.csv").drop("Unnamed: 0",axis=1)
```

2.1. Size of the Dataset

In [4]:
```
print("Dataset Size :")
print("Training Size : ",dataset_train.shape)
print("Test Size : ",dataset_test.shape)
```

```
Dataset Size :
Training Size :  (150000, 11)
Test Size :  (101503, 11)
```

2.2. The tenth first training data

In [5]:
```
print("Tenth first Training Data :")
dataset_train.head(10)
```

```
Tenth first Training Data :
```

Out[5]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLine
0	1	0.766127	45	2	0.802982	9120.0	
1	0	0.957151	40	0	0.121876	2600.0	
2	0	0.658180	38	1	0.085113	3042.0	
3	0	0.233810	30	0	0.036050	3300.0	
4	0	0.907239	49	1	0.024926	63588.0	
5	0	0.213179	74	0	0.375607	3500.0	
6	0	0.305682	57	0	5710.000000	NaN	
7	0	0.754464	39	0	0.209940	3500.0	
8	0	0.116951	27	0	46.000000	NaN	
9	0	0.189169	57	0	0.606291	23684.0	

2.2. The tenth first testing data

In [6]:
```
print("Tenth first Test Data :")
dataset_test.head(10)
```

```
Tenth first Test Data :
```

Out[6]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLine
0	NaN	0.885519	43	0	0.177513	5700.0	
1	NaN	0.463295	57	0	0.527237	9141.0	
2	NaN	0.043275	59	0	0.687648	5083.0	
3	NaN	0.280308	38	1	0.925961	3200.0	
4	NaN	1.000000	27	0	0.019917	3865.0	
5	NaN	0.509791	63	0	0.342429	4140.0	
6	NaN	0.587778	50	0	1048.000000	0.0	
7	NaN	0.046149	79	1	0.369170	3301.0	
8	NaN	0.013527	68	0	2024.000000	NaN	
9	NaN	1.000000	23	98	0.000000	0.0	

2.3. Describe Data Training

In [7]:
```
dataset_train.describe()
```

Out[7]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	Number(
count	150000.000000	150000.000000	150000.000000	150000.000000	150000.000000	1.202690e+05	
mean	0.066840	6.048438	52.295207	0.421033	353.005076	6.670221e+03	
std	0.249746	249.755371	14.771866	4.192781	2037.818523	1.438467e+04	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	
25%	0.000000	0.029867	41.000000	0.000000	0.175074	3.400000e+03	
50%	0.000000	0.154181	52.000000	0.000000	0.366508	5.400000e+03	
75%	0.000000	0.559046	63.000000	0.000000	0.868254	8.249000e+03	
max	1.000000	50708.000000	109.000000	98.000000	329664.000000	3.008750e+06	

3. Visualization data training

3.1. Ratio Y/N (0 and 1) in SeriousDlqin2yrs

In [107]:
```
sns.countplot(dataset_train['SeriousDlqin2yrs'],label="Count")
```

Out[107]: <matplotlib.axes._subplots.AxesSubplot at 0x1af76278550>

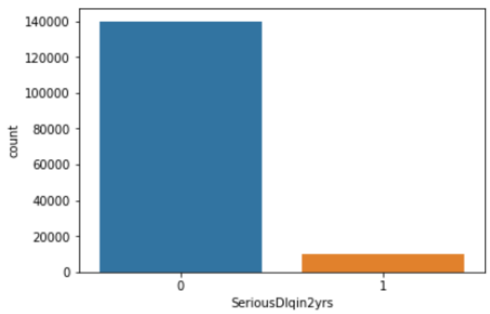


Figure 1. SeriousDlqin2yrs ratio chart

In figure 2 shows the ratio of SeriousDlqin2yrs value between 0 and 1. From the figure we can see that the result is prefer to 0 value than 1 value. The result of this prediction is not good, because the dominant value is the priority of this dataset's classifier.

1.1. Graphics ratio of SeriousDlqin2yrs and DebtRatio

In [11]:
```
# Your code goes here
plt.figure(figsize=(8,6))

plt.scatter(dataset_train['DebtRatio'],dataset_train['SeriousDlqin2yrs'],marker='o')
ax = plt.subplot(111)
plt.ylabel('SeriousDlqin2yrs')
ax.yaxis.label.set_color('green')
plt.xlabel('DebtRatio')
ax.xaxis.label.set_color('red')
plt.title('SeriousDlqin2yrs information against DebtRatio', fontsize=20);
plt.show()
```

```
C:\Users\Wahyu Nainggolan\Anaconda3\lib\site-packages\matplotlib\cbook\deprecation.py:106: MatplotlibDeprecation
Warning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance.  In a
future version, a new instance will always be created and returned.  Meanwhile, this warning can be suppressed,
and the future behavior ensured, by passing a unique label to each axes instance.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```

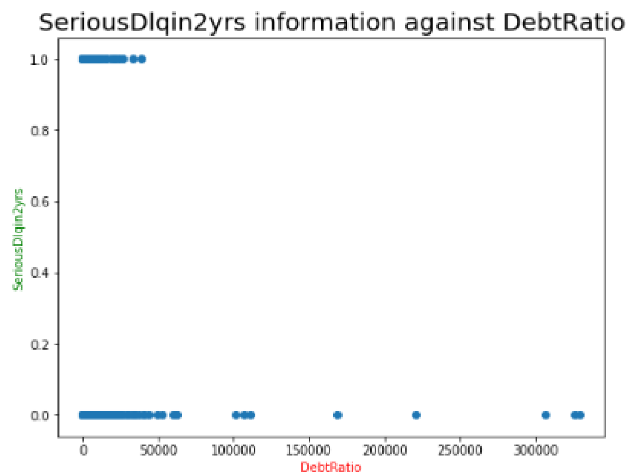
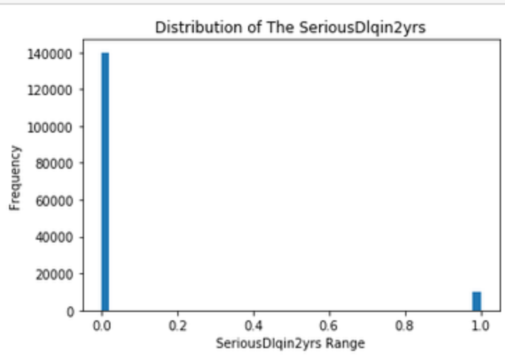


Figure 2

Figure shows that DebtRatio value can impact the SeriousDlqin2yrs value.

1.2. Visualize the distribution of the SeriousDlqin2yrs

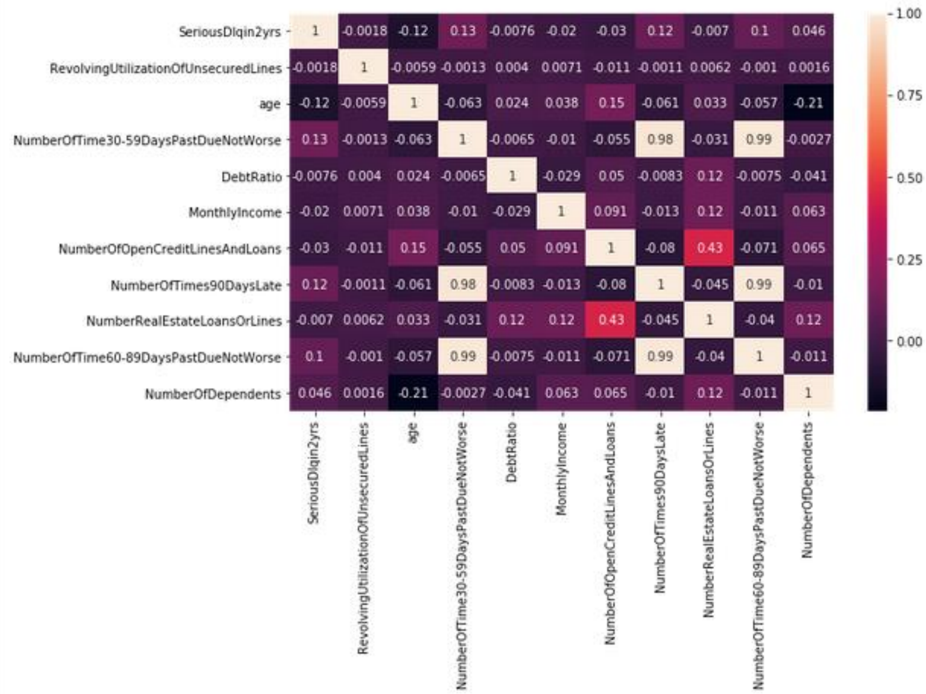
In [12]:
```
plt.hist(dataset_train['SeriousDlqin2yrs'], bins=50)
plt.title("Distribution of The SeriousDlqin2yrs")
plt.xlabel("SeriousDlqin2yrs Range")
plt.ylabel("Frequency")
plt.show()
```



1.3. Graphics of All Data Relationship

In [15]:
```
corr = dataset_train.corr()
plt.figure(figsize=(10,6))
sns.heatmap(corr, annot=True, fmt=".2g")
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1f749bdb588>



4. Preprocessing Data Training and Data Testing

4.1. Missing Value

4.1.1. Missing Value in Data Training

In [8]:
```
null_counts_data_train = dataset_train.isnull().sum()
print("Number of null values in each column in data train:\n{}".format(null_counts_data_train))
```

```
Number of null values in each column in data train:
SeriousDlqin2yrs                            0
RevolvingUtilizationOfUnsecuredLines        0
age                                         0
NumberOfTime30-59DaysPastDueNotWorse        0
DebtRatio                                   0
MonthlyIncome                           29731
NumberOfOpenCreditLinesAndLoans             0
NumberOfTimes90DaysLate                     0
NumberRealEstateLoansOrLines                0
NumberOfTime60-89DaysPastDueNotWorse        0
NumberOfDependents                       3924
dtype: int64
```

4.1.2. Solve Missing Value in Data Training

In [9]:
```
print("----------------------Solve Missing Value Data Train----------------------------------------")
dataset_train.iloc[:,1:] = SoftImpute().complete(dataset_train.iloc[:,1:])
dataset_train = dataset_train.round({'age': 0, 'NumberOfTime30-59DaysPastDueNotWorse': 0, 'NumberOfOpenCreditLinesAndLoans':0,
                             'NumberOfTimes90DaysLate':0, 'NumberRealEstateLoansOrLines':0, 'NumberOfTime60-89DaysPastDueNotWor
                             'NumberOfDependents':0})
```

```
---------------------Solve Missing Value Data Train----------------------------------------
[SoftImpute] Max Singular Value of X_init = 5498797.118077
[SoftImpute] Iter 1: observed MAE=22.144330 rank=2

C:\Users\Wahyu Nainggolan\Anaconda3\lib\site-packages\fancyimpute\soft_impute.py:100: RuntimeWarning: divide by zero enco
untered in double_scalars
  return (np.sqrt(ssd) / old_norm) < self.convergence_threshold

[SoftImpute] Iter 2: observed MAE=22.144389 rank=2
[SoftImpute] Iter 3: observed MAE=22.144398 rank=2
[SoftImpute] Iter 4: observed MAE=22.144402 rank=2
[SoftImpute] Iter 5: observed MAE=22.144406 rank=2
[SoftImpute] Iter 6: observed MAE=22.144409 rank=2
[SoftImpute] Iter 7: observed MAE=22.144413 rank=2
[SoftImpute] Iter 8: observed MAE=22.144416 rank=2
[SoftImpute] Iter 9: observed MAE=22.144420 rank=2
[SoftImpute] Iter 10: observed MAE=22.144423 rank=2
[SoftImpute] Iter 11: observed MAE=22.144426 rank=2
[SoftImpute] Iter 12: observed MAE=22.144429 rank=2
[SoftImpute] Iter 13: observed MAE=22.144432 rank=2
```

4.1.3. Tenth First Training Data Without Missing Value

In [10]:
```
print("10 Data Train pertama tanpa missing value:  ")
dataset_train.head(10)
```

```
10 Data Train pertama tanpa missing value:
```

Out[10]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLine:
0	1	0.766127	45.0	2.0	0.802982	9120.000000	
1	0	0.957151	40.0	0.0	0.121876	2600.000000	
2	0	0.658180	38.0	1.0	0.085113	3042.000000	
3	0	0.233810	30.0	0.0	0.036050	3300.000000	
4	0	0.907239	49.0	1.0	0.024926	63588.000000	
5	0	0.213179	74.0	0.0	0.375607	3500.000000	
6	0	0.305682	57.0	0.0	5710.000000	2.323121	
7	0	0.754464	39.0	0.0	0.209940	3500.000000	
8	0	0.116951	27.0	0.0	46.000000	1.611424	
9	0	0.189169	57.0	0.0	0.606291	23684.000000	

4.1.4. Missing Value in Data Testing

In [11]:
```
null_counts_data_train = dataset_test.isnull().sum()
print("Number of null values in each column in data test:\n{}".format(null_counts_data_train))
```

```
Number of null values in each column in data test:
SeriousDlqin2yrs                        101503
RevolvingUtilizationOfUnsecuredLines         0
age                                          0
NumberOfTime30-59DaysPastDueNotWorse         0
DebtRatio                                    0
MonthlyIncome                            20103
NumberOfOpenCreditLinesAndLoans              0
NumberOfTimes90DaysLate                      0
NumberRealEstateLoansOrLines                 0
NumberOfTime60-89DaysPastDueNotWorse         0
NumberOfDependents                        2626
dtype: int64
```

4.1.5. Solve Missing Value in Data Testing

```
In [12]: print("----------------------Solve Missing Value Data Test-----------------------------------------")
         dataset_test.iloc[:,1:] = SoftImpute().complete(dataset_test.iloc[:,1:])
         dataset_test = dataset_test.round({'age': 0, 'NumberOfTime30-59DaysPastDueNotWorse': 0, 'NumberOfOpenCreditLinesAndLoans':0,
                                'NumberOfTimes90DaysLate':0, 'NumberRealEstateLoansOrLines':0, 'NumberOfTime60-89DaysPastDueNotWor
                                'NumberOfDependents':0})
```

```
----------------------Solve Missing Value Data Test-----------------------------------------
[SoftImpute] Max Singular Value of X_init = 10598115.191638
[SoftImpute] Iter 1: observed MAE=32.035881 rank=2
[SoftImpute] Iter 2: observed MAE=32.035890 rank=2

C:\Users\Wahyu Nainggolan\Anaconda3\lib\site-packages\fancyimpute\soft_impute.py:100: RuntimeWarning: divide by zero enco
untered in double_scalars
  return (np.sqrt(ssd) / old_norm) < self.convergence_threshold

[SoftImpute] Iter 3: observed MAE=32.035891 rank=2
[SoftImpute] Iter 4: observed MAE=32.035891 rank=2
[SoftImpute] Iter 5: observed MAE=32.035891 rank=2
[SoftImpute] Iter 6: observed MAE=32.035891 rank=2
[SoftImpute] Iter 7: observed MAE=32.035892 rank=2
[SoftImpute] Iter 8: observed MAE=32.035892 rank=2
[SoftImpute] Iter 9: observed MAE=32.035892 rank=2
[SoftImpute] Iter 10: observed MAE=32.035893 rank=2
[SoftImpute] Iter 11: observed MAE=32.035893 rank=2
[SoftImpute] Iter 12: observed MAE=32.035893 rank=2
[SoftImpute] Iter 13: observed MAE=32.035893 rank=2
```

4.1.6. Tenth First Training Data Without Missing Value

```
In [13]: print("10 Data Test pertama tanpa missing value:  ")
         dataset_test.head(10)
```

```
10 Data Test pertama tanpa missing value:
```

Out[13]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLine:
0	NaN	0.885519	43.0	0.0	0.177513	5700.000000	
1	NaN	0.463295	57.0	0.0	0.527237	9141.000000	
2	NaN	0.043275	59.0	0.0	0.687648	5083.000000	
3	NaN	0.280308	38.0	1.0	0.925961	3200.000000	
4	NaN	1.000000	27.0	0.0	0.019917	3865.000000	
5	NaN	0.509791	63.0	0.0	0.342429	4140.000000	
6	NaN	0.587778	50.0	0.0	1048.000000	0.000000	
7	NaN	0.046149	79.0	1.0	0.369170	3301.000000	
8	NaN	0.013527	68.0	0.0	2024.000000	0.725155	
9	NaN	1.000000	23.0	98.0	0.000000	0.000000	

In this data testing, SeriousDlqin2yrs will be filled with the result of the prediction.

4.2. Normalization Data

MonthlyIncome and DebtRatio attributes are normalized, because both of them have high dimension value. StandartScaler normalization will be used to handle this problem.

4.2.1. Data Training Before Normalization

In [36]:
```
print("Tenth Data Training Before Normalization")
dataset_train.head(10)
```

```
Tenth Data Training Before Normalization
```

Out[36]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenC
0	1	0.766127	45.0	2.0	0.802982	9120.000000	
1	0	0.957151	40.0	0.0	0.121876	2600.000000	
2	0	0.658180	38.0	1.0	0.085113	3042.000000	
3	0	0.233810	30.0	0.0	0.036050	3300.000000	
4	0	0.907239	49.0	1.0	0.024926	63588.000000	
5	0	0.213179	74.0	0.0	0.375607	3500.000000	
6	0	0.305682	57.0	0.0	5710.000000	2.323121	
7	0	0.754464	39.0	0.0	0.209940	3500.000000	
8	0	0.116951	27.0	0.0	46.000000	1.611424	
9	0	0.189169	57.0	0.0	0.606291	23684.000000	

4.2.2. Data Testing Before Normalization

In [37]:
```
print("Tenth Data Testing Before Normalization")
dataset_test.head(10)
```

```
Tenth Data Testing Before Normalization
```

Out[37]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenC
0	NaN	0.885519	43.0	0.0	0.177513	5700.000000	
1	NaN	0.463295	57.0	0.0	0.527237	9141.000000	
2	NaN	0.043275	59.0	0.0	0.687648	5083.000000	
3	NaN	0.280308	38.0	1.0	0.925961	3200.000000	
4	NaN	1.000000	27.0	0.0	0.019917	3865.000000	
5	NaN	0.509791	63.0	0.0	0.342429	4140.000000	
6	NaN	0.587778	50.0	0.0	1048.000000	0.000000	
7	NaN	0.046149	79.0	1.0	0.369170	3301.000000	
8	NaN	0.013527	68.0	0.0	2024.000000	0.725155	
9	NaN	1.000000	23.0	98.0	0.000000	0.000000	

4.2.3. Normalization Data Training and Data Testing

In [14]:
```
scaler = StandardScaler()
train_size = dataset_train.values.shape[0]
test_size = dataset_test.values.shape[0]
dataset_train['MonthlyIncome'] = scaler.fit_transform(dataset_train['MonthlyIncome'].values.reshape([train_size,-1]))
dataset_test['MonthlyIncome'] = scaler.transform(dataset_test['MonthlyIncome'].values.reshape([test_size,-1]))
dataset_train['DebtRatio'] = scaler.fit_transform(dataset_train['DebtRatio'].values.reshape([train_size,-1]))
dataset_test['DebtRatio'] = scaler.transform(dataset_test['DebtRatio'].values.reshape([test_size,-1]))
```

4.2.4. Data Training After Normalization

```
In [11]: dataset_train.head(5)
```

Out[11]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCre
0	1	0.766127	45.0	2.0	-0.172833	0.286748	
1	0	0.957151	40.0	0.0	-0.173168	-0.209004	
2	0	0.658180	38.0	1.0	-0.173186	-0.175396	
3	0	0.233810	30.0	0.0	-0.173210	-0.155779	
4	0	0.907239	49.0	1.0	-0.173215	4.428247	

4.2.5. Data Testing After Normalization

```
In [12]: print("5 data test pertama setelah di normalisasi")
         dataset_test.head(5)
```

```
5 data test pertama setelah di normalisasi
```

Out[12]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCre
0	NaN	0.885519	43.0	0.0	-0.173140	0.026706	
1	NaN	0.463295	57.0	0.0	-0.172969	0.288344	
2	NaN	0.043275	59.0	0.0	-0.172890	-0.020207	
3	NaN	0.280308	38.0	1.0	-0.172773	-0.163382	
4	NaN	1.000000	27.0	0.0	-0.173218	-0.112819	

5. Splitting Data Training and Data Testing to Independent and dependent variable

```
In [18]: independent_variabel_data_train = ['RevolvingUtilizationOfUnsecuredLines', 'age','NumberOfTime30-59DaysPastDueNotWorse', 'Debt
         independent_variabel_data_train = dataset_train[independent_variabel_data_train].values
         dependent_variabel_data_train = dataset_train[['SeriousDlqin2yrs']].values
         |
         independent_variabel_data_test = ['RevolvingUtilizationOfUnsecuredLines', 'age','NumberOfTime30-59DaysPastDueNotWorse', 'DebtR
         independent_variabel_data_test = dataset_test[independent_variabel_data_test].values
         dependent_variabel_data_test = dataset_test[['SeriousDlqin2yrs']].values
```

RevolvingUtilizationOfUnsecuredLines, age, NumberOfTime30-59DaysPastDueNotWorse, DebtRatio, MonthlyIncome, NumberOfOpenCreditLinesAndLoans, NumberOfTimes90DaysLate, NumberRealEstateLoansOrLines, NumberOfTime60-89DaysPastDueNotWorse and NumberOfDependents are the independent variable, whereas SeriousDlqin2yrs is the dependent variable.

6. Splitting Independent Variable Data Training to Data Training and Data Validation

```
In [19]: X_tr, X_val, Y_tr, Y_val = train_test_split(independent_variabel_data_train,dependent_variabel_data_train, test_size = 0.2, ra
print("Ukuran X_tr:",X_tr.shape)
print("Ukuran X_val:",X_val.shape)
print("Ukuran Y_tr:",Y_tr.shape)
print("Ukuran Y_val:",Y_val.shape)
```

```
Ukuran X_tr: (120000, 10)
Ukuran X_val: (30000, 10)
Ukuran Y_tr: (120000, 1)
Ukuran Y_val: (30000, 1)
```

The independent variable of data training will be splitted to data training dan data validation. Data training will be used to train the data with the algorithm, the data validation will be used to evaluate the model. Held out cross validation will be used to split the data with 80:20 (80 for training dan 20 for test).

7. Modeling Algorithm DNN Backpropagation with Data Training

DNN Backpropagation is one of neural network's algorithm. This algorithm is suit for binary data. There are parameter dan hyper-parameters in DNN algorithm. Parameter influence the model and the accuration while hyper-parameters influence the parameter. Parameter is weight and bias, while hyper-parameters is hidden layer, activation function, epoch, learning rate, etc.

In first experiment the hidden layer will be tuned:

```
In [23]: layer_1 = 500
layer_2 = 400
layer_3 = 300
layer_4 = 200
layer_5 = 100
nilai_activation = 'relu'
opt=Adam(lr=0.1)
epoch = 50
```

There are a few steps to train the DNN model:

7.1. Create Layer and Add Layer to Model DNN

```
In [31]: model_DNN = Sequential()
model_DNN.add(Dense(units=layer_1, input_dim=X_tr.shape[1], activation=nilai_activation))
model_DNN.add(Dense(units=layer_2, activation=nilai_activation))
model_DNN.add(Dense(units=layer_3, activation=nilai_activation))
model_DNN.add(Dense(units=layer_4, activation=nilai_activation))
model_DNN.add(Dense(units=layer_5, activation=nilai_activation))
model_DNN.add(Dense(Y_tr.shape[1], activation=nilai_activation))
```

7.2. Compile Model DNN

```
In [32]: # Compile model
model_DNN.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
model_DNN_compile=model_DNN.fit(X_tr, Y_tr,batch_size= 1000, epochs=epoch, verbose=1, validation_data=(X_val, Y_val))
```

```
Train on 120000 samples, validate on 30000 samples
Epoch 1/50
120000/120000 [==============================] - 12s 101us/step - loss: 0.2506 - acc: 0.9335 - val_loss: 0.2484 - val_acc
: 0.9319
Epoch 2/50
120000/120000 [==============================] - 11s 95us/step - loss: 0.2101 - acc: 0.9334 - val_loss: 0.1893 - val_acc:
0.9331
Epoch 3/50
120000/120000 [==============================] - 11s 90us/step - loss: 0.1823 - acc: 0.9341 - val_loss: 0.1866 - val_acc:
0.9333
Epoch 4/50
120000/120000 [==============================] - 11s 91us/step - loss: 0.1829 - acc: 0.9350 - val_loss: 0.1865 - val_acc:
0.9342
Epoch 5/50
120000/120000 [==============================] - 11s 91us/step - loss: 0.1810 - acc: 0.9359 - val_loss: 0.1851 - val_acc:
0.9358
Epoch 6/50
120000/120000 [==============================] - 11s 94us/step - loss: 0.1796 - acc: 0.9366 - val_loss: 0.1845 - val_acc:
0.9366
Epoch 7/50
```

8. Evaluated Model Algorithm with data validation

```
In [56]: scores = model_DNN.evaluate(X_val, Y_val)
print("\n%s: %.2f%%" % (model_DNN.metrics_names[1], scores[1]*100))
```

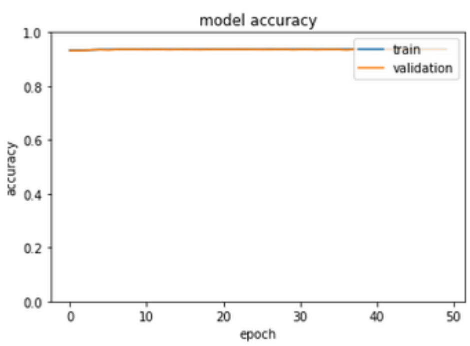
```
30000/30000 [==============================] - 3s 93us/step

acc: 93.19%
```

The accuration for the first experiment is 93,19%

9. Analysis Model Algorithm

9.1. Learning Curve Accuracy Against Epoch

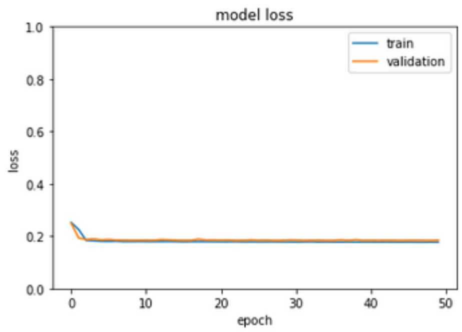
```
In [23]: plot_accuracy_against_epoch(model_DNN_compile)
```



The graphic shows that the training accuration and the validation accuration is fit. When the accuration is fit, the prediction of the model in experiment one can handle well.

9.2. Learning Curve Loss Against Epoch

```
In [24]: plot_loss_against_epoch(model_DNN_compile)
```



Th graphc shows that the training loss and the validation loss is fit. When the loss is fit, the prediction of the model in experiment one can handle well too.

9.3. Clasification Report

```
In [37]: print(classification_report(Y_val, [np.round(pred) for pred in deep_neural_network_predict], labels=[0, 1]))

             precision    recall  f1-score   support

          0       0.94      0.99      0.97     27957
          1       0.64      0.16      0.25      2043

avg / total       0.92      0.94      0.92     30000
```

10. Tuning Hyper-parameters algoritma DNN Bacpropagation

Improve the prediction result can be done by tuned the hyper-parameters.

10.1. Tuning Activation Function

In this section Relu, Sigmoid, Tanh and Softmax activation functions are tuned.

```
In [42]: activation_functions = ['relu','sigmoid','tanh','softmax']
cvscores = []
counter = 0

for a in range(len(activation_functions)):
            #create Hyperparameters
            rand_layer_1 = 500
            rand_layer_2 = 400
            rand_layer_3 = 300
            rand_layer_4 = 200
            rand_layer_5 = 100
            value_activation = activation_functions[a]
            # create model
            model = Sequential()
            model.add(Dense(units=rand_layer_1, input_dim=X_tr.shape[1], activation=value_activation))
            model.add(Dense(units=rand_layer_2, activation=value_activation))
            model.add(Dense(units=rand_layer_3, activation=value_activation))
            model.add(Dense(units=rand_layer_4, activation=value_activation))
            model.add(Dense(units=rand_layer_5, activation=value_activation))
            model.add(Dense(Y_tr.shape[1], activation='sigmoid'))
            # Compile model
            model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
            # Fit the model
            model.fit(X_tr, Y_tr, epochs=50, verbose=0)
            # evaluate the model
            scores = model.evaluate(X_tr, Y_tr, verbose=0)
            print("Hidden Unit 1: {0};\nHidden Unit 2: {1};\nHidden Unit 3: {2};\nHidden Unit 4: {3};\nHidden Unit
                    .format(rand_layer_1,
                            rand_layer_2,
                            rand_layer_3,
                            rand_layer_4,
                            rand_layer_5,
                            value_activation,"%s: %.2f%%" % (model.metrics_names[1], scores[1]*100)))
            cvscores.append(scores[1] * 100)
            counter += 1
            print("{0}-attempt(s) with acurracy approx.: {1}"
                .format(counter, "%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores))))
```

```
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: relu;
==> acc: 93.76%
1-attempt(s) with acurracy approx.: 93.76% (+/- 0.00%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
==> acc: 93.77%
2-attempt(s) with acurracy approx.: 93.77% (+/- 0.01%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: tanh;
==> acc: 93.35%
3-attempt(s) with acurracy approx.: 93.63% (+/- 0.20%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: softmax;
==> acc: 93.35%
4-attempt(s) with acurracy approx.: 93.56% (+/- 0.21%)
```

The tuning process shows that the accuration of Sigmoid is 93,77%.

10.2. Tuning Learning Rate

In this section will be tuned 4 kinds of learning rate (1, 0.1,0.01, and 0.001)

```
In [45]: learning_rate = [1,0.1,0.01,0.001]
cvscores = []
counter = 0

for b in range(len(learning_rate)):
            #create Hyperparameters
            rand_layer_1 = 500
            rand_layer_2 = 400
            rand_layer_3 = 300
            rand_layer_4 = 200
            rand_layer_5 = 100
            value_activation = 'sigmoid'
            value_learning_rate = learning_rate[b]
            opt=Adam(lr=value_learning_rate)

            # create model
            model = Sequential()
            model.add(Dense(units=rand_layer_1, input_dim=X_tr.shape[1], activation=value_activation))
            model.add(Dense(units=rand_layer_2, activation=value_activation))
            model.add(Dense(units=rand_layer_3, activation=value_activation))
            model.add(Dense(units=rand_layer_4, activation=value_activation))
            model.add(Dense(units=rand_layer_5, activation=value_activation))
            model.add(Dense(Y_tr.shape[1], activation='sigmoid'))
            # Compile model
            model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
            # Fit the model
            model.fit(X_tr, Y_tr, epochs=50, verbose=0)
            # evaluate the model
            scores = model.evaluate(X_tr, Y_tr, verbose=0)
            print("Hidden Unit 1: {0};\nHidden Unit 2: {1};\nHidden Unit 3: {2};\nHidden Unit 4: {3};\nHidden Unit
                    .format(rand_layer_1,
                            rand_layer_2,
                            rand_layer_3,
                            rand_layer_4,
                            rand_layer_5,
                            value_activation,
                            value_learning_rate,"%s: %.2f%%" % (model.metrics_names[1], scores[1]*100)))
            cvscores.append(scores[1] * 100)
            counter += 1
            print("{0}-attempt(s) with acurracy approx.: {1}"
                .format(counter, "%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores))))
```

```
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
Learning Rate: 1;
==> acc: 93.35%
1-attempt(s) with acurracy approx.: 93.35% (+/- 0.00%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
Learning Rate: 0.1;
==> acc: 93.35%
2-attempt(s) with acurracy approx.: 93.35% (+/- 0.00%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
Learning Rate: 0.01;
==> acc: 93.35%
3-attempt(s) with acurracy approx.: 93.35% (+/- 0.00%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
Learning Rate: 0.001;
==> acc: 93.80%
4-attempt(s) with acurracy approx.: 93.46% (+/- 0.20%)
```

The tuning process shows that the best learning rate is 0,001 with accuration 93,80%.

10.3. Tuning Epoch

In this section will be tuned 4 kinds of epoch (50, 100, 150, and 200).

```
In [18]: epoch = [50,100,150,200]
cvscores = []
counter = 0

for c in range(len(epoch)):
            #create Hyperparameters
            rand_layer_1 = 500
            rand_layer_2 = 400
            rand_layer_3 = 300
            rand_layer_4 = 200
            rand_layer_5 = 100
            value_activation = 'sigmoid'
            value_learning_rate = 0.001
            value_epoch = epoch[c]
            opt=Adam(lr=value_learning_rate)
            # create model
            model = Sequential()
            model.add(Dense(units=rand_layer_1, input_dim=X_tr.shape[1], activation=value_activation))
            model.add(Dense(units=rand_layer_2, activation=value_activation))
            model.add(Dense(units=rand_layer_3, activation=value_activation))
            model.add(Dense(units=rand_layer_4, activation=value_activation))
            model.add(Dense(units=rand_layer_5, activation=value_activation))
            model.add(Dense(Y_tr.shape[1], activation='sigmoid'))
            # Compile model
            model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
            # Fit the model
            model.fit(X_tr, Y_tr, epochs=value_epoch, verbose=0)
            # evaluate the model
            scores = model.evaluate(X_tr, Y_tr, verbose=0)
            print("Hidden Unit 1: {0};\nHidden Unit 2: {1};\nHidden Unit 3: {2};\nHidden Unit 4: {3};\nHidden Unit
                    .format(rand_layer_1,
                            rand_layer_2,
                            rand_layer_3,
                            rand_layer_4,
                            rand_layer_5,
                            value_activation,
                            value_learning_rate,
                            value_epoch,"%s: %.2f%%" % (model.metrics_names[1], scores[1]*100)))
            cvscores.append(scores[1] * 100)
            counter += 1
            print("{0}-attempt(s) with acurracy approx.: {1}"
                .format(counter, "%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores))))
```

```
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
Learning Rate: 0.001;
Epoch: 50;
==> acc: 93.77%
1-attempt(s) with acurracy approx.: 93.77% (+/- 0.00%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
Learning Rate: 0.001;
Epoch: 100;
==> acc: 94.06%
2-attempt(s) with acurracy approx.: 93.91% (+/- 0.15%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
Learning Rate: 0.001;
Epoch: 150;
==> acc: 94.95%
3-attempt(s) with acurracy approx.: 94.26% (+/- 0.50%)
Hidden Unit 1: 500;
Hidden Unit 2: 400;
Hidden Unit 3: 300;
Hidden Unit 4: 200;
Hidden Unit 5: 100;
Activation Function: sigmoid;
Learning Rate: 0.001;
Epoch: 200;
==> acc: 95.02%
4-attempt(s) with acurracy approx.: 94.45% (+/- 0.55%)
```

The tuning process shows that the best epoch is 200 with accuration 95.02%.

11. Prediction after tuned the Hyper-parameters

```
In [*]: layer_1 = 500
        layer_2 = 400
        layer_3 = 300
        layer_4 = 200
        layer_5 = 100
        nilai_activation = 'sigmoid'
        opt=Adam(lr=0.001)
        epoch = 200
        #Create layer
        model_DNN = Sequential()
        model_DNN.add(Dense(units=layer_1, input_dim=X_tr.shape[1], activation=nilai_activation))
        model_DNN.add(Dense(units=layer_2, activation=nilai_activation))
        model_DNN.add(Dense(units=layer_3, activation=nilai_activation))
        model_DNN.add(Dense(units=layer_4, activation=nilai_activation))
        model_DNN.add(Dense(units=layer_5, activation=nilai_activation))
        model_DNN.add(Dense(Y_tr.shape[1], activation=nilai_activation))
        # Compile model
        model_DNN.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
        model_DNN_compile=model_DNN.fit(X_tr, Y_tr,batch_size= 1000, epochs=epoch, verbose=1, validation_data=(X_val, Y_val
        #Evaluated Model
        scores = model_DNN.evaluate(X_val, Y_val)
        print("\n%s: %.2f%%" % (model_DNN.metrics_names[1], scores[1]*100))
```

```
- val_acc: 0.9353
Epoch 196/200
120000/120000 [==============================] - 11s 88us/step - loss: 0.1719 - acc: 0.9396 - val_loss: 0.1860
- val_acc: 0.9355
Epoch 197/200
120000/120000 [==============================] - 12s 97us/step - loss: 0.1715 - acc: 0.9398 - val_loss: 0.1864
- val_acc: 0.9358
Epoch 198/200
120000/120000 [==============================] - 12s 96us/step - loss: 0.1719 - acc: 0.9392 - val_loss: 0.1864
- val_acc: 0.9357
Epoch 199/200
120000/120000 [==============================] - 11s 94us/step - loss: 0.1717 - acc: 0.9392 - val_loss: 0.1875
- val_acc: 0.9353
Epoch 200/200
120000/120000 [==============================] - 11s 88us/step - loss: 0.1717 - acc: 0.9395 - val_loss: 0.1878
- val_acc: 0.9356
30000/30000 [==============================] - 3s 87us/step

acc: 93.56%
```

```
In [47]: dnn_predict_with_tuning_hyperparameters = model_DNN.predict(X_val)
         predict_validation = pd.DataFrame(dnn_predict_with_tuning_hyperparameters)
         print("Tenth predict validation after tuning hyper-parameters : ")
         predict_validation.head(10)
```

Tenth predict validation after tuning hyper-parameters :

Out[47]:

	0
0	0.012570
1	0.025313
2	0.017281
3	0.028360
4	0.068669
5	0.044931
6	0.024690
7	0.014795
8	0.008886
9	0.068951

The accuration for the first experiment is 93,56%

11.1. Analysis Result

```
In [42]: print(classification_report(Y_val, [np.round(pred) for pred in dnn_predict_with_tuning_hyperparameters], labels=[0,
```

```
             precision    recall  f1-score   support

          0       0.94      0.99      0.97     27957
          1       0.60      0.16      0.25      2043

avg / total       0.92      0.94      0.92     30000
```

The result of the prediction shows that by tuning the hyper-parameter of DNN Backpropagation algorithm, the accuration can be improved.

12. Predict Dataset Testing to Model Algoritma DNN Backpropagation After Tuning

```
In [49]: predict = pd.DataFrame(model_DNN.predict(independent_variabel_data_test))
         print('Tenth data predict :')
         predict.head(10)
```

```
Tenth data predict :
```

Out[49]:

	0
0	0.056217
1	0.048616
2	0.010041
3	0.048885
4	0.075639
5	0.023909
6	0.025062
7	0.029128
8	0.003262
9	0.385999

12.1. Convert result predict to Comma (CSV)

	A	B
1	Id	probability
2	1	0.056217
3	2	0.048616
4	3	0.010041
5	4	0.048885
6	5	0.075639
7	6	0.023909
8	7	0.025062
9	8	0.029128
10	9	0.003262
11	10	0.385999
12	11	0.012556
13	12	0.010024
14	13	0.008212
15	14	0.056871
16	15	0.037994
17	16	0.014774
18	17	0.020875
19	18	0.012211
20	19	0.134198
21	20	0.07953
22	21	0.00919
23	22	0.010054
24	23	0.00341