Lending Club Loan Data Analysis

DESCRIPTION

Create a model that predicts whether or not a loan will be default using the historical data.

Problem Statement:

For companies like Lending Club correctly predicting whether or not a loan will be a default is very important. In this project, using the historical data from 2007 to 2015, you have to build a deep learning model to predict the chance of default for future loans. As you will see later this dataset is highly imbalanced and includes a lot of features that makes this problem more challenging.

Domain: Finance

Analysis to be done: Perform data preprocessing and build a deep learning prediction model.

Steps to perform:

Perform exploratory data analysis and feature engineering and then apply feature engineering. Follow up with a deep learning model to predict whether or not the loan will be default using the historical data.

Import Libraries

```
#!pip install tensorflow
```

```
# Importing Important Library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model

%matplotlib inline
```

Loading Dataset

```
dataset = pd.read_csv('loan_data.csv')
dataset.head(10)
```

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revo
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	
5	1	credit_card	0.0788	125.13	11.904968	16.98	727	6120.041667	50807	
6	1	debt_consolidation	0.1496	194.02	10.714418	4.00	667	3180.041667	3839	
7	1	all_other	0.1114	131.22	11.002100	11.08	722	5116.000000	24220	
8	1	home_improvement	0.1134	87.19	11.407565	17.25	682	3989.000000	69909	
9	1	debt_consolidation	0.1221	84.12	10.203592	10.00	707	2730.041667	5630	

1. Feature Transformation

Transforming categorical values into numerical values (discrete)

```
dataset.dtypes
```

```
credit.policy          int64
purpose               object
int.rate             float64
installment          float64
log.annual.inc       float64
dti                  float64
fico                   int64
days.with.cr.line    float64
revol.bal              int64
revol.util           float64
inq.last.6mths         int64
delinq.2yrs            int64
pub.rec                int64
not.fully.paid         int64
dtype: object
```

```
dataset['purpose']
```

```
0       debt_consolidation
1              credit_card
2       debt_consolidation
3       debt_consolidation
4              credit_card
               ...
9573             all_other
9574             all_other
9575    debt_consolidation
9576      home_improvement
9577    debt_consolidation
Name: purpose, Length: 9578, dtype: object
```

```
# dataset_trans = dataset.copy()
# dataset_trans['purpose'] = pd.factorize(dataset['purpose'])[0]
dataset_trans = pd.get_dummies(dataset, columns = ['purpose'])
dataset_trans.head(10)
```

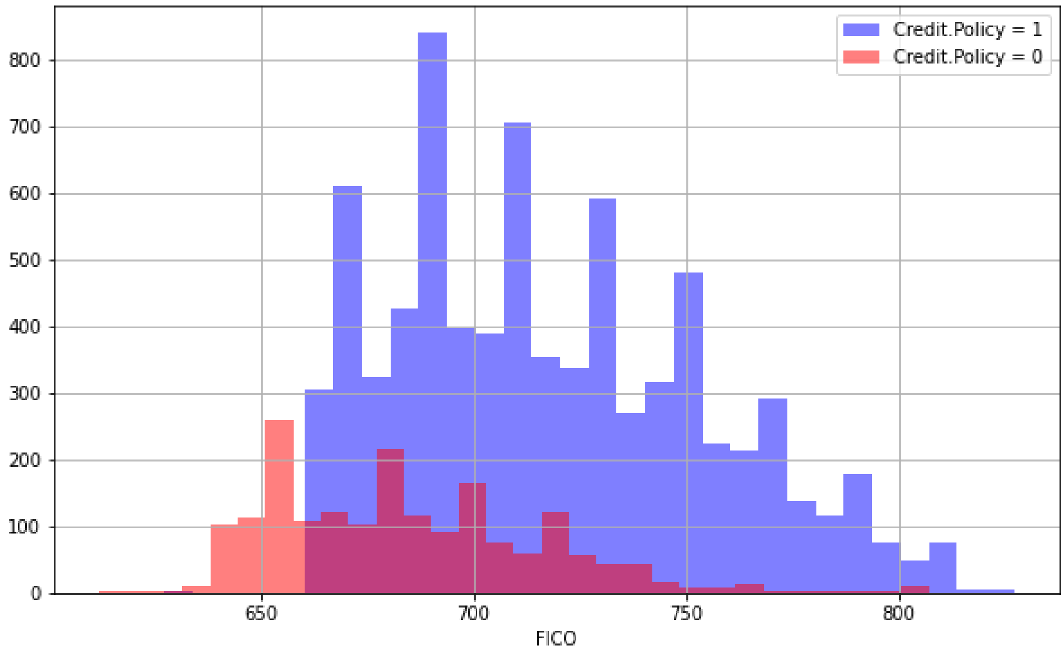
	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mth
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	
5	1	0.0788	125.13	11.904968	16.98	727	6120.041667	50807	51.0	
6	1	0.1496	194.02	10.714418	4.00	667	3180.041667	3839	76.8	
7	1	0.1114	131.22	11.002100	11.08	722	5116.000000	24220	68.6	
8	1	0.1134	87.19	11.407565	17.25	682	3989.000000	69909	51.1	
9	1	0.1221	84.12	10.203592	10.00	707	2730.041667	5630	23.0	

2. Exploratory data analysis of different factors of the dataset.

Let’s see some data visualization with seaborn and plotting. Create a histogram of two FICO distributions on top of each other, one for each credit.policy outcome.

```
plt.figure(figsize=(10,6))
dataset[dataset['credit.policy'] == 1]['fico'].hist(alpha=0.5,color="blue",bins=30,label="Credit
dataset[dataset['credit.policy'] == 0]['fico'].hist(alpha=0.5,color="red",bins=30,label="Credit.
plt.legend()
plt.xlabel('FICO')
```

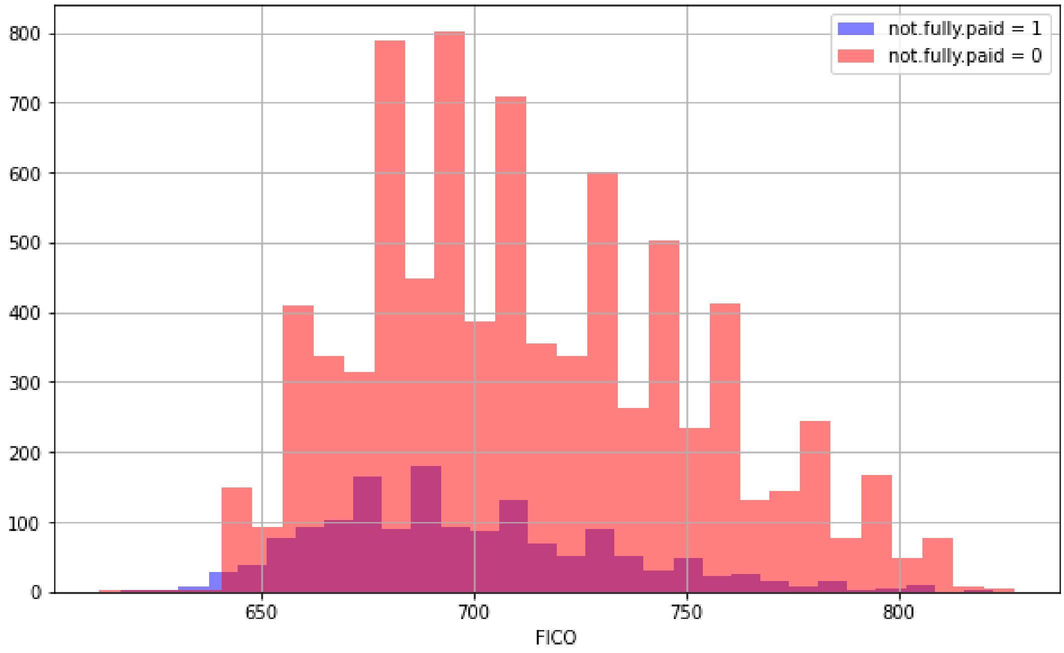
```
Text(0.5, 0, 'FICO')
```



Let's see a similar chart for “not.fully.paid” column.

```
plt.figure(figsize=(10,6))
dataset[dataset['not.fully.paid'] == 1]['fico'].hist(alpha=0.5,color="blue",bins=30,label="not.f
dataset[dataset['not.fully.paid'] == 0]['fico'].hist(alpha=0.5,color="red",bins=30,label="not.fu
plt.legend()
plt.xlabel('FICO')
```

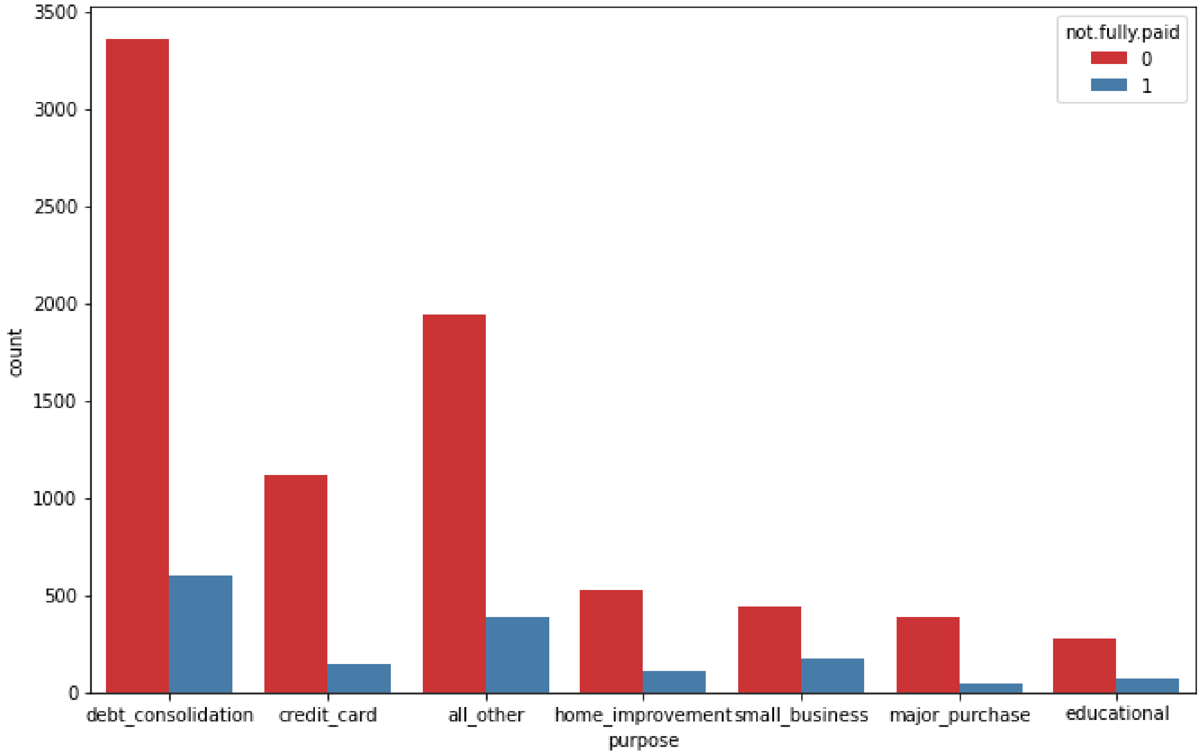
```
Text(0.5, 0, 'FICO')
```



Graph based on grouby of loan purpose

```
plt.figure(figsize=(11,7))
sns.countplot(x='purpose',hue='not.fully.paid',data=dataset,palette='Set1')
```

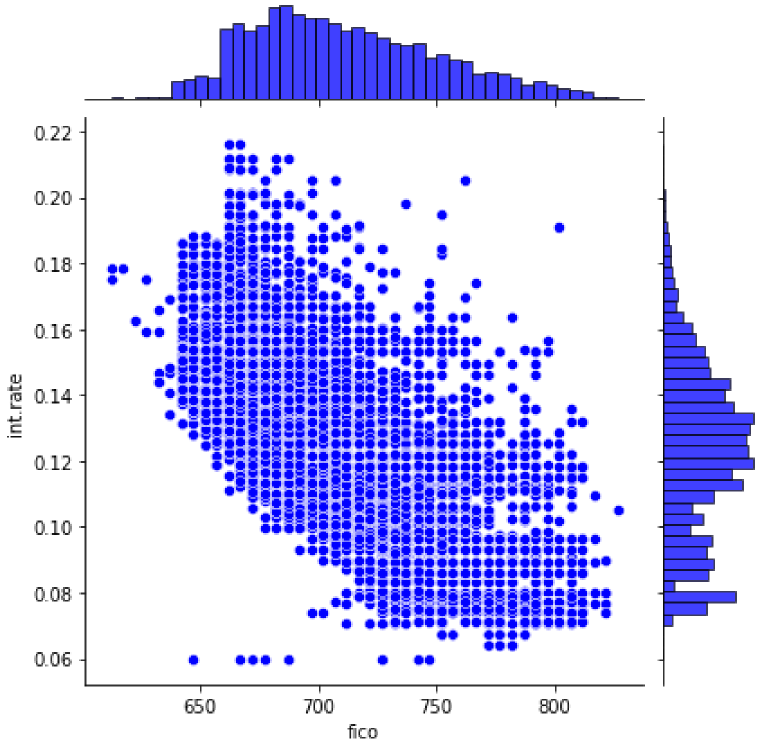
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fca1692a790>
```



Trends of FICO and Interest Rate by jointplot

```
sns.jointplot(x='fico',y='int.rate',data=dataset,color='blue')
```

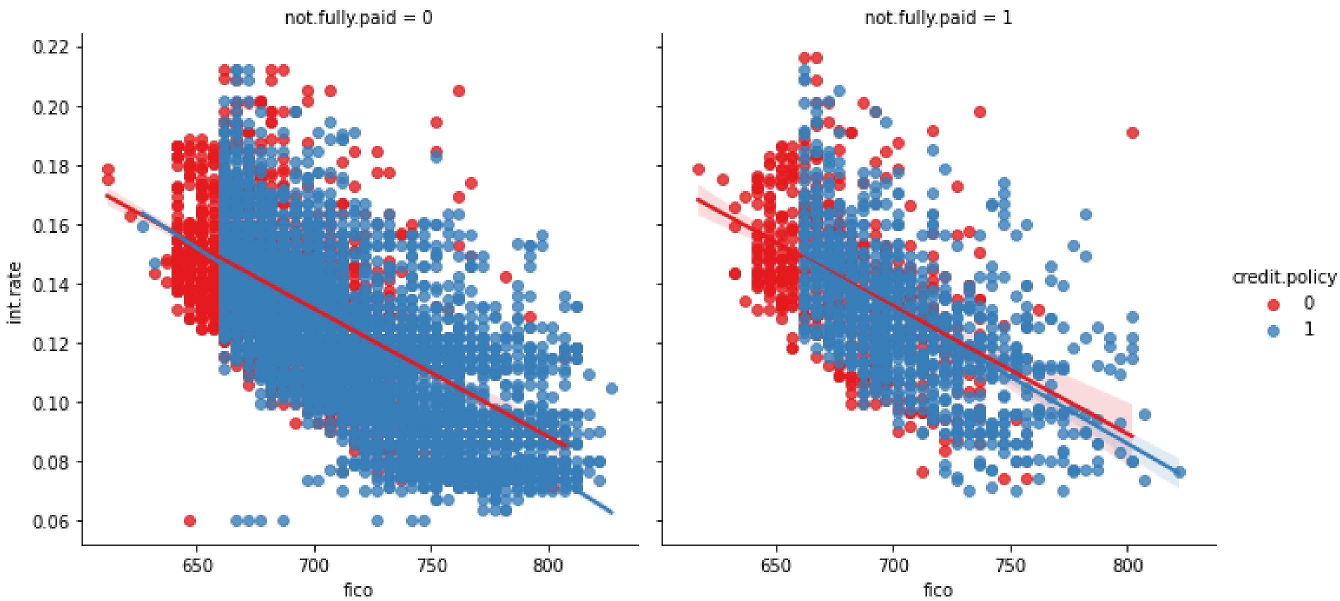
```
<seaborn.axisgrid.JointGrid at 0x7fca13698e50>
```



Comparing trend between not.fully.paid and credit.policy

```
plt.figure(figsize=(11,7))
sns.lmplot(y='int.rate',x='fico',data=dataset,hue='credit.policy', col='not.fully.paid',palette=
```

```
<seaborn.axisgrid.FacetGrid at 0x7fca13556950>
<Figure size 792x504 with 0 Axes>
```



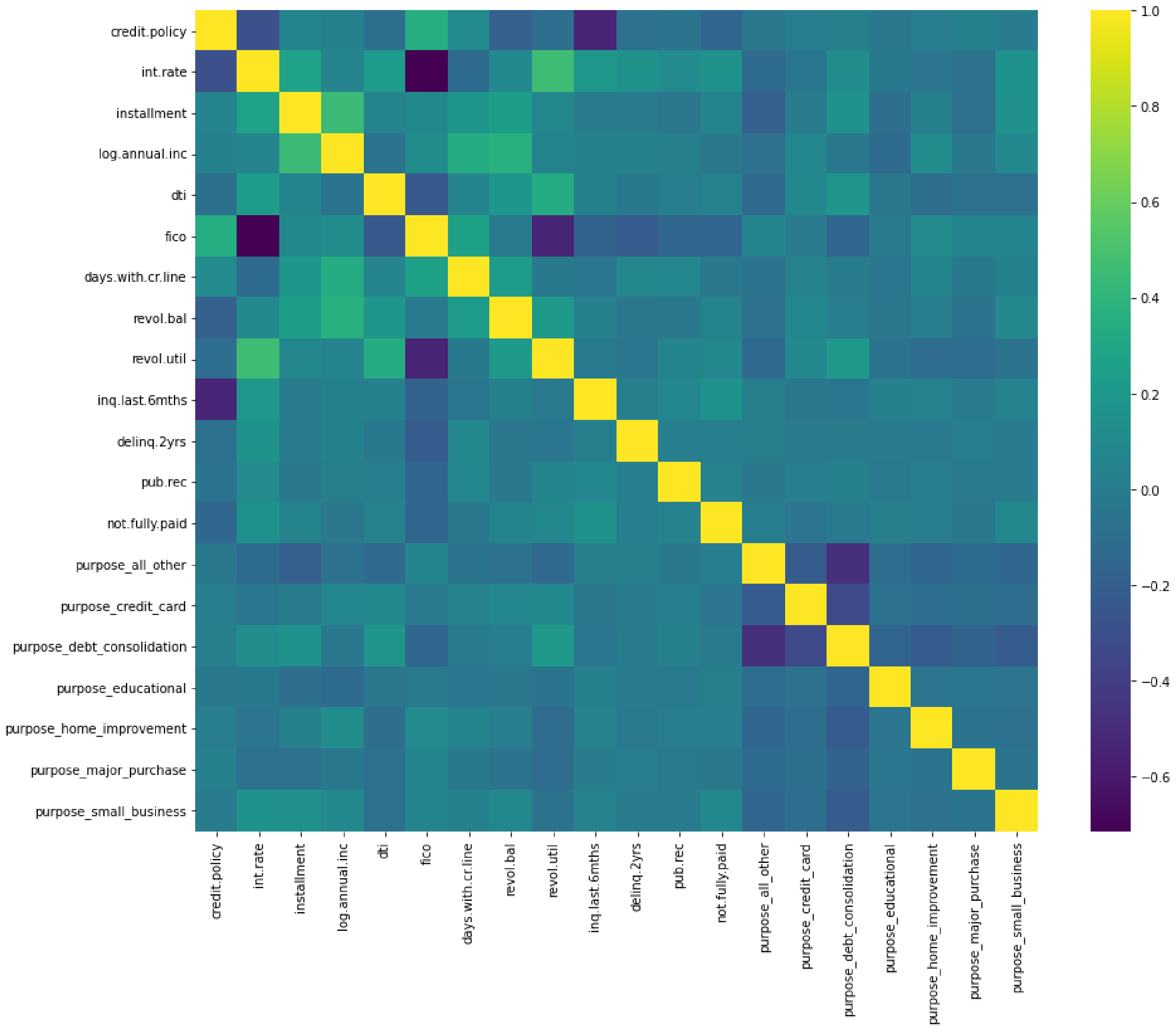
3. Additional Feature Engineering

Correlation Matrix

```
corr = dataset_trans.corr()
corr
```

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.c
credit.policy	1.000000	-0.294089	0.058770	0.034906	-0.090901	0.348319	0.09
int.rate	-0.294089	1.000000	0.276140	0.056383	0.220006	-0.714821	-0.12
installment	0.058770	0.276140	1.000000	0.448102	0.050202	0.086039	0.18
log.annual.inc	0.034906	0.056383	0.448102	1.000000	-0.054065	0.114576	0.33
dti	-0.090901	0.220006	0.050202	-0.054065	1.000000	-0.241191	0.06
fico	0.348319	-0.714821	0.086039	0.114576	-0.241191	1.000000	0.26
days.with.cr.line	0.099026	-0.124022	0.183297	0.336896	0.060101	0.263880	1.00
revol.bal	-0.187518	0.092527	0.233625	0.372140	0.188748	-0.015553	0.22
revol.util	-0.104095	0.464837	0.081356	0.054881	0.337109	-0.541289	-0.02
inq.last.6mths	-0.535511	0.202780	-0.010419	0.029171	0.029189	-0.185293	-0.04
delinq.2yrs	-0.076318	0.156079	-0.004368	0.029203	-0.021792	-0.216340	0.08
pub.rec	-0.054243	0.098162	-0.032760	0.016506	0.006209	-0.147592	0.07
not.fully.paid	-0.158119	0.159552	0.049955	-0.033439	0.037362	-0.149666	-0.02
purpose_all_other	-0.025412	-0.124000	-0.203103	-0.080077	-0.125825	0.067184	-0.05
purpose_credit_card	0.003216	-0.042109	0.000774	0.072942	0.084476	-0.012512	0.04
purpose_debt_consolidation	0.020193	0.123607	0.161658	-0.026214	0.179149	-0.154132	-0.00
purpose_educational	-0.031346	-0.019618	-0.094510	-0.119799	-0.035325	-0.013012	-0.04
purpose_home_improvement	0.006036	-0.050697	0.023024	0.116375	-0.092788	0.097474	0.06
purpose_major_purchase	0.024281	-0.068978	-0.079836	-0.031020	-0.077719	0.067129	-0.02
purpose_small_business	-0.003511	0.151247	0.145654	0.091540	-0.069245	0.063292	0.03

```
# Plot the heatmap
plt.figure(figsize=(15,12))
sns.heatmap(corr, cmap='viridis')
plt.show()
```



There is a strong correlation between **installment** and **revol.util** with **log.annual.inc** and **int.rate** respectively. This multicollinearity should be removed in the following model because these two values explain the data in the same manner. We would be overfitting the model if both of these features are contained in the final model. Most machine learning models carry assumptions which calls for little multicollinearity.

```
# Drop the installment and remov.util column to reduce multi correlations
dataset_final = dataset_trans.drop(['installment','revol.util'], axis=1)
dataset_final.head()
```

	credit.policy	int.rate	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	inq.last.6mths	delinq.2yrs	pub.rec
0	1	0.1189	11.350407	19.48	737	5639.958333	28854	0	0	0
1	1	0.1071	11.082143	14.29	707	2760.000000	33623	0	0	0
2	1	0.1357	10.373491	11.63	682	4710.000000	3511	1	0	0
3	1	0.1008	11.350407	8.10	712	2699.958333	33667	1	0	0
4	1	0.1426	11.299732	14.97	667	4066.000000	4740	0	1	0

4. Modeling

```
to_train = dataset_final[dataset_final['not.fully.paid'].isin([0,1])]
to_pred = dataset_final[dataset_final['not.fully.paid'] == 2]
```

```
X = to_train.drop('not.fully.paid', axis=1).values
y = to_train['not.fully.paid'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 8)

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential()

model.add(
        Dense(19, activation='relu')
)

model.add(
        Dense(10, activation='relu')
)

model.add(
        Dense(5, activation='relu')
)


model.add(
        Dense(1, activation='sigmoid')
)
```

```
model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy']
)
```

```
early_stop = EarlyStopping(
        monitor='val_loss',
        mode='min',
        verbose=1,
        patience=25
)

history = model.fit(
        X_train,
        y_train,
        epochs=200,
        batch_size=256,
        validation_data=(X_test, y_test),
         callbacks=[early_stop]
)
```

```
Epoch 1/200
27/27 [==============================] - 1s 9ms/step - loss: 0.6076 - accuracy: 0.8367 - val_los
s: 0.5426 - val_accuracy: 0.8459
Epoch 2/200
27/27 [==============================] - 0s 3ms/step - loss: 0.5020 - accuracy: 0.8374 - val_los
s: 0.4550 - val_accuracy: 0.8459
Epoch 3/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4523 - accuracy: 0.8374 - val_los
s: 0.4353 - val_accuracy: 0.8459
Epoch 4/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4446 - accuracy: 0.8374 - val_los
s: 0.4300 - val_accuracy: 0.8459
Epoch 5/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4408 - accuracy: 0.8374 - val_los
s: 0.4257 - val_accuracy: 0.8459
Epoch 6/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4376 - accuracy: 0.8374 - val_los
s: 0.4222 - val_accuracy: 0.8459
Epoch 7/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4348 - accuracy: 0.8374 - val_los
s: 0.4187 - val_accuracy: 0.8459
Epoch 8/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4321 - accuracy: 0.8374 - val_los
s: 0.4154 - val_accuracy: 0.8459
Epoch 9/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4294 - accuracy: 0.8374 - val_los
s: 0.4123 - val_accuracy: 0.8459
Epoch 10/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4272 - accuracy: 0.8374 - val_los
s: 0.4096 - val_accuracy: 0.8459
Epoch 11/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4251 - accuracy: 0.8374 - val_los
s: 0.4077 - val_accuracy: 0.8459
Epoch 12/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4238 - accuracy: 0.8374 - val_los
s: 0.4065 - val_accuracy: 0.8459
Epoch 13/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4231 - accuracy: 0.8374 - val_los
s: 0.4054 - val_accuracy: 0.8459
Epoch 14/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4222 - accuracy: 0.8374 - val_los
s: 0.4048 - val_accuracy: 0.8459
Epoch 15/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4217 - accuracy: 0.8374 - val_los
s: 0.4047 - val_accuracy: 0.8459
Epoch 16/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4212 - accuracy: 0.8374 - val_los
s: 0.4037 - val_accuracy: 0.8459
Epoch 17/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4207 - accuracy: 0.8374 - val_los
s: 0.4037 - val_accuracy: 0.8459
Epoch 18/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4204 - accuracy: 0.8374 - val_los
s: 0.4033 - val_accuracy: 0.8459
Epoch 19/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4201 - accuracy: 0.8374 - val_los
s: 0.4029 - val_accuracy: 0.8459
Epoch 20/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4201 - accuracy: 0.8374 - val_los
s: 0.4029 - val_accuracy: 0.8459
Epoch 21/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4194 - accuracy: 0.8374 - val_los
s: 0.4026 - val_accuracy: 0.8459
Epoch 22/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4195 - accuracy: 0.8374 - val_los
s: 0.4026 - val_accuracy: 0.8459
Epoch 23/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4191 - accuracy: 0.8374 - val_los
s: 0.4023 - val_accuracy: 0.8459
Epoch 24/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4188 - accuracy: 0.8374 - val_los
```

```
s: 0.4023 - val_accuracy: 0.8459
Epoch 25/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4186 - accuracy: 0.8374 - val_los
s: 0.4025 - val_accuracy: 0.8459
Epoch 26/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4185 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8459
Epoch 27/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4184 - accuracy: 0.8374 - val_los
s: 0.4021 - val_accuracy: 0.8459
Epoch 28/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4182 - accuracy: 0.8374 - val_los
s: 0.4023 - val_accuracy: 0.8459
Epoch 29/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4180 - accuracy: 0.8374 - val_los
s: 0.4021 - val_accuracy: 0.8459
Epoch 30/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4178 - accuracy: 0.8374 - val_los
s: 0.4019 - val_accuracy: 0.8459
Epoch 31/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4177 - accuracy: 0.8374 - val_los
s: 0.4020 - val_accuracy: 0.8459
Epoch 32/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4173 - accuracy: 0.8374 - val_los
s: 0.4020 - val_accuracy: 0.8459
Epoch 33/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4173 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8459
Epoch 34/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4171 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8459
Epoch 35/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4170 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8459
Epoch 36/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4170 - accuracy: 0.8374 - val_los
s: 0.4018 - val_accuracy: 0.8459
Epoch 37/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4173 - accuracy: 0.8374 - val_los
s: 0.4017 - val_accuracy: 0.8459
Epoch 38/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4167 - accuracy: 0.8374 - val_los
s: 0.4020 - val_accuracy: 0.8459
Epoch 39/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4168 - accuracy: 0.8374 - val_los
s: 0.4026 - val_accuracy: 0.8459
Epoch 40/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4171 - accuracy: 0.8374 - val_los
s: 0.4018 - val_accuracy: 0.8459
Epoch 41/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4163 - accuracy: 0.8374 - val_los
s: 0.4016 - val_accuracy: 0.8459
Epoch 42/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4167 - accuracy: 0.8374 - val_los
s: 0.4027 - val_accuracy: 0.8459
Epoch 43/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4163 - accuracy: 0.8374 - val_los
s: 0.4016 - val_accuracy: 0.8459
Epoch 44/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4162 - accuracy: 0.8374 - val_los
s: 0.4017 - val_accuracy: 0.8459
Epoch 45/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4162 - accuracy: 0.8374 - val_los
s: 0.4023 - val_accuracy: 0.8459
Epoch 46/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4159 - accuracy: 0.8374 - val_los
s: 0.4015 - val_accuracy: 0.8459
Epoch 47/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4159 - accuracy: 0.8374 - val_los
s: 0.4023 - val_accuracy: 0.8459
Epoch 48/200
```

```
27/27 [==============================] - 0s 3ms/step - loss: 0.4165 - accuracy: 0.8374 - val_los
s: 0.4017 - val_accuracy: 0.8459
Epoch 49/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4154 - accuracy: 0.8374 - val_los
s: 0.4017 - val_accuracy: 0.8459
Epoch 50/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4155 - accuracy: 0.8374 - val_los
s: 0.4018 - val_accuracy: 0.8459
Epoch 51/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4160 - accuracy: 0.8374 - val_los
s: 0.4018 - val_accuracy: 0.8459
Epoch 52/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4152 - accuracy: 0.8374 - val_los
s: 0.4016 - val_accuracy: 0.8459
Epoch 53/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4152 - accuracy: 0.8374 - val_los
s: 0.4015 - val_accuracy: 0.8459
Epoch 54/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4150 - accuracy: 0.8374 - val_los
s: 0.4017 - val_accuracy: 0.8459
Epoch 55/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4156 - accuracy: 0.8374 - val_los
s: 0.4015 - val_accuracy: 0.8459
Epoch 56/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4149 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8459
Epoch 57/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4149 - accuracy: 0.8374 - val_los
s: 0.4017 - val_accuracy: 0.8459
Epoch 58/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4148 - accuracy: 0.8374 - val_los
s: 0.4018 - val_accuracy: 0.8459
Epoch 59/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4146 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8459
Epoch 60/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4143 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8459
Epoch 61/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4149 - accuracy: 0.8374 - val_los
s: 0.4021 - val_accuracy: 0.8459
Epoch 62/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4145 - accuracy: 0.8374 - val_los
s: 0.4021 - val_accuracy: 0.8459
Epoch 63/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4146 - accuracy: 0.8376 - val_los
s: 0.4029 - val_accuracy: 0.8459
Epoch 64/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4144 - accuracy: 0.8376 - val_los
s: 0.4023 - val_accuracy: 0.8459
Epoch 65/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4139 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8459
Epoch 66/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4140 - accuracy: 0.8374 - val_los
s: 0.4021 - val_accuracy: 0.8455
Epoch 67/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4140 - accuracy: 0.8374 - val_los
s: 0.4029 - val_accuracy: 0.8459
Epoch 68/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4143 - accuracy: 0.8373 - val_los
s: 0.4019 - val_accuracy: 0.8452
Epoch 69/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4140 - accuracy: 0.8374 - val_los
s: 0.4022 - val_accuracy: 0.8452
Epoch 70/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4140 - accuracy: 0.8377 - val_los
s: 0.4019 - val_accuracy: 0.8452
Epoch 71/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4137 - accuracy: 0.8374 - val_los
s: 0.4021 - val_accuracy: 0.8452
```

```
Epoch 72/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4134 - accuracy: 0.8373 - val_los
s: 0.4027 - val_accuracy: 0.8459
Epoch 73/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4139 - accuracy: 0.8374 - val_los
s: 0.4021 - val_accuracy: 0.8452
Epoch 74/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4132 - accuracy: 0.8379 - val_los
s: 0.4023 - val_accuracy: 0.8441
Epoch 75/200
27/27 [==============================] - 0s 4ms/step - loss: 0.4134 - accuracy: 0.8382 - val_los
s: 0.4025 - val_accuracy: 0.8459
Epoch 76/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4129 - accuracy: 0.8379 - val_los
s: 0.4020 - val_accuracy: 0.8462
Epoch 77/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4131 - accuracy: 0.8376 - val_los
s: 0.4031 - val_accuracy: 0.8462
Epoch 78/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4126 - accuracy: 0.8380 - val_los
s: 0.4026 - val_accuracy: 0.8462
Epoch 79/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4127 - accuracy: 0.8379 - val_los
s: 0.4029 - val_accuracy: 0.8452
Epoch 80/200
27/27 [==============================] - 0s 3ms/step - loss: 0.4132 - accuracy: 0.8379 - val_los
s: 0.4023 - val_accuracy: 0.8448
Epoch 80: early stopping
```

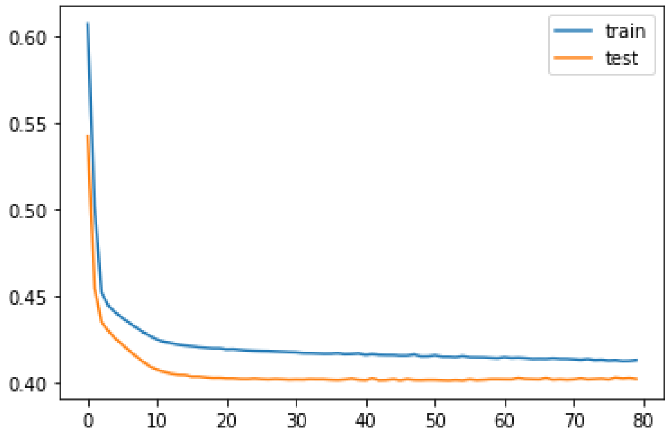
Evaluating the Model

```
_, train_acc = model.evaluate(X_train,y_train, verbose=1)
_, test_acc = model.evaluate(X_test, y_test, verbose=1)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

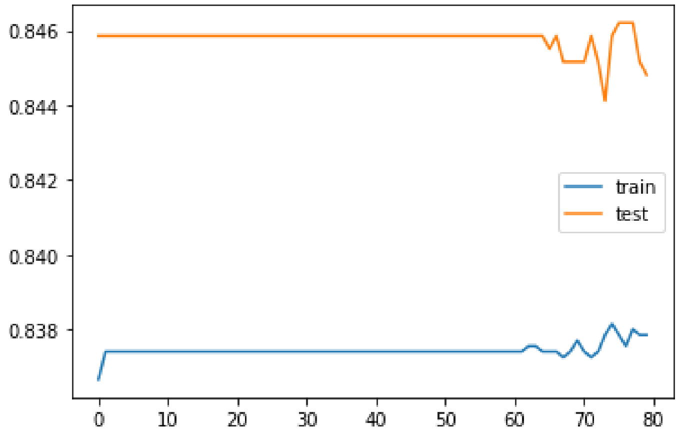
```
210/210 [==============================] - 0s 1ms/step - loss: 0.4123 - accuracy: 0.8380
90/90 [==============================] - 0s 1ms/step - loss: 0.4023 - accuracy: 0.8448
Train: 0.838, Test: 0.845
```

Ploting History

```
plt.plot(history.history['loss'],label='train')
plt.plot(history.history['val_loss'],label='test')
plt.legend()
plt.show()
```




```
plt.plot(history.history['accuracy'],label='train')
plt.plot(history.history['val_accuracy'],label='test')
plt.legend()
plt.show()
```


0.846
0.844
0.842
0.840
0.838
0
10
20
30
40
50
60
70
80
train
test