Deep Learning with Keras and Tensorflow

Project 2 - Lending Club Loan Data Analysis

Problem Statement

- Build a deep learning model to predict the chance of default for future loans.

In [1]:

```
#importing necessary libraries
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

#importing libraries for visualisation
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import style

#importing libraries for tensorflow
import tensorflow as tf
```

In [2]:

```
#importing Data
data_file=r'C:\Users\sinun\OneDrive\Documents\Simplilearn\DEEP Learning\project\project2\loan_data.csv'
data_frame=pd.read_csv(data_file)
```

In [3]:

```
# Understanding the Data Variables
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   object
```

```
 2   int.rate          9578 non-null   float64
 3   installment       9578 non-null   float64
 4   log.annual.inc    9578 non-null   float64
 5   dti               9578 non-null   float64
 6   fico              9578 non-null   int64
 7   days.with.cr.line 9578 non-null   float64
 8   revol.bal         9578 non-null   int64
 9   revol.util        9578 non-null   float64
 10  inq.last.6mths    9578 non-null   int64
 11  delinq.2yrs       9578 non-null   int64
 12  pub.rec           9578 non-null   int64
 13  not.fully.paid    9578 non-null   int64
dtypes: float64(6), int64(7), object(1)
memory usage: 1.0+ MB
```

In [4]:

```
#understand the shape of Dataset
data_frame.shape
```

Out[4]: (9578, 14)

In [5]:

```
# Show the top 5 Rows of data
data_frame.head()
```

Out[5]:

	credit.policy	purpose	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.r
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	

In [6]:

```
# Performing Descriptive Analysis
data_frame.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
credit.policy	9578.0	0.804970	0.396245	0.000000	1.000000	1.000000	1.000000	1.000000e+00
int.rate	9578.0	0.122640	0.026847	0.060000	0.103900	0.122100	0.140700	2.164000e-01
installment	9578.0	319.089413	207.071301	15.670000	163.770000	268.950000	432.762500	9.401400e+02
log.annual.inc	9578.0	10.932117	0.614813	7.547502	10.558414	10.928884	11.291293	1.452835e+01
dti	9578.0	12.606679	6.883970	0.000000	7.212500	12.665000	17.950000	2.996000e+01
fico	9578.0	710.846314	37.970537	612.000000	682.000000	707.000000	737.000000	8.270000e+02
days.with.cr.line	9578.0	4560.767197	2496.930377	178.958333	2820.000000	4139.958333	5730.000000	1.763996e+04
revol.bal	9578.0	16913.963876	33756.189557	0.000000	3187.000000	8596.000000	18249.500000	1.207359e+06
revol.util	9578.0	46.799236	29.014417	0.000000	22.600000	46.300000	70.900000	1.190000e+02
inq.last.6mths	9578.0	1.577469	2.200245	0.000000	0.000000	1.000000	2.000000	3.300000e+01
delinq.2yrs	9578.0	0.163708	0.546215	0.000000	0.000000	0.000000	0.000000	1.300000e+01
pub.rec	9578.0	0.062122	0.262126	0.000000	0.000000	0.000000	0.000000	5.000000e+00
not.fully.paid	9578.0	0.160054	0.366676	0.000000	0.000000	0.000000	0.000000	1.000000e+00

CHECK FOR MISSING VALUES

In [7]:

```
# Checking for null values
data_frame.isnull().sum()
```

Out[7]:

```
credit.policy        0
purpose              0
int.rate             0
installment          0
log.annual.inc       0
dti                  0
fico                 0
days.with.cr.line    0
revol.bal            0
revol.util           0
inq.last.6mths       0
delinq.2yrs          0
```

```
pub.rec              0
not.fully.paid       0
dtype: int64
```

- No missing values in any columns

Feature Transformation

* Transform categorical values into numerical values (discrete)

In [8]:

```python
# Understand the type of each variable
data_frame.dtypes
```

Out[8]:

```
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
```

IDENTIFY THE CATEGORICAL COLUMN

In [9]:

```python
#nFind the values and their count for each column
imp_cols=data_frame.columns
def number_count(): # function defined for finding value counts in respective columns
    for col in imp_cols:
        print('Name of Variable :', col)
        print(data_frame[col].value_counts(),'\n\n')
number_count()
```

```
Name of Variable : credit.policy
1    7710
0    1868
Name: credit.policy, dtype: int64


Name of Variable : purpose
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64


Name of Variable : int.rate
0.1253    354
0.0894    299
0.1183    243
0.1218    215
0.0963    210
         ...
0.1756      1
0.1741      1
0.1772      1
0.1746      1
0.1941      1
Name: int.rate, Length: 249, dtype: int64


Name of Variable : installment
317.72    41
316.11    34
319.47    29
381.26    27
662.68    27
          ..
107.23     1
232.60     1
211.65     1
261.89     1
62.45      1
Name: installment, Length: 4788, dtype: int64
```

```
Name of Variable : log.annual.inc
11.002100    308
10.819778    248
10.308953    224
10.596635    224
10.714418    221
            ...
11.170717      1
11.956328      1
11.203679      1
10.292823      1
11.000499      1
Name: log.annual.inc, Length: 1987, dtype: int64


Name of Variable : dti
0.00     89
10.00    19
0.60     16
6.00     13
19.20    13
         ..
15.23     1
1.32      1
22.14     1
29.21     1
27.47     1
Name: dti, Length: 2529, dtype: int64


Name of Variable : fico
687    548
682    536
692    498
697    476
702    472
707    444
667    438
677    427
717    424
662    414
672    395
712    395
722    388
727    361
732    330
742    324
```

```
737    313
752    258
747    236
757    231
762    220
772    158
767    142
777    140
652    131
657    127
782    118
647    112
642    102
792     97
787     85
797     76
802     55
807     45
812     33
632      6
817      6
637      5
822      5
627      2
612      2
622      1
827      1
617      1
Name: fico, dtype: int64


Name of Variable : days.with.cr.line
3660.000000     50
3630.000000     48
3990.000000     46
4410.000000     44
3600.000000     41
                ..
2788.958333      1
11708.000000     1
5432.000000      1
6001.041667      1
10740.000000     1
Name: days.with.cr.line, Length: 2687, dtype: int64


Name of Variable : revol.bal
```

```
0        321
255       10
298       10
682        9
346        8
        ...
16077      1
1738       1
57033      1
54165      1
2047       1
Name: revol.bal, Length: 7869, dtype: int64


Name of Variable : revol.util
0.00      297
0.50       26
0.30       22
47.80      22
73.70      22
         ...
5.34        1
26.32       1
100.50      1
103.10      1
108.80      1
Name: revol.util, Length: 1035, dtype: int64


Name of Variable : inq.last.6mths
0     3637
1     2462
2     1384
3      864
4      475
5      278
6      165
7      100
8       72
9       47
10      23
12      15
11      15
15       9
13       6
14       6
18       4
```

```
16       3
19       2
17       2
24       2
33       1
27       1
25       1
20       1
28       1
32       1
31       1
Name: inq.last.6mths, dtype: int64


Name of Variable : delinq.2yrs
0     8458
1      832
2      192
3       65
4       19
5        6
6        2
8        1
11       1
13       1
7        1
Name: delinq.2yrs, dtype: int64


Name of Variable : pub.rec
0    9019
1     533
2      19
3       5
4       1
5       1
Name: pub.rec, dtype: int64


Name of Variable : not.fully.paid
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

In [10]:
```
# Categorical column is 'Purpose'  which should be coverted into numerical
data_frame['purpose'].value_counts()
```

Out[10]:
```
debt_consolidation    3957
all_other             2331
credit_card           1262
home_improvement       629
small_business         619
major_purchase         437
educational            343
Name: purpose, dtype: int64
```

One Hot Encoding to convert categorical values to numerical values

In [11]:
```
# One Hot Encoding to convert categorical column to numerical
data_frame1=pd.get_dummies(data_frame, columns=['purpose'],drop_first=True)
```

In [12]:
```
data_frame1.head()
```

Out[12]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0

In [13]:
```
# Displaying the columns in the dataset
data_frame1.columns
```

Out[13]:
```
Index(['credit.policy', 'int.rate', 'installment', 'log.annual.inc', 'dti',
       'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid',
```

```
 'purpose_credit_card', 'purpose_debt_consolidation',
 'purpose_educational', 'purpose_home_improvement',
 'purpose_major_purchase', 'purpose_small_business'],
dtype='object')
```

Exploratory Data Analysis (EDA) of different factors of the dataset.

Check whether there is Data IMBALANCE

In [14]:

```
# Check values in target variable
data_frame1['not.fully.paid'].value_counts()
```

Out[14]:

```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

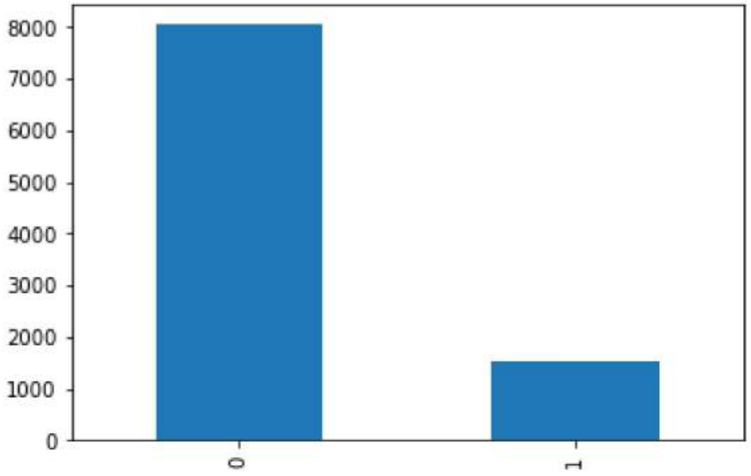
- From the value counts, its clear that there is DATA IMBALANCE
- Majority Class(Class 0) with 84 % of total values of target variable
- Minority Class(Class 1) with 16 % of total values of target variable

In [15]:

```
# Check for any data imbalance by plotting the count of outcomes by the values
data_frame1['not.fully.paid'].value_counts().plot(kind='bar')
```

Out[15]:

```
<AxesSubplot:>
```



- From the visual representation also we can see the data IMBALANCE. So when training the data, this might cause an issue. So necessary steps should be taken for resolving the issue.

Check for outliers in differnt variables

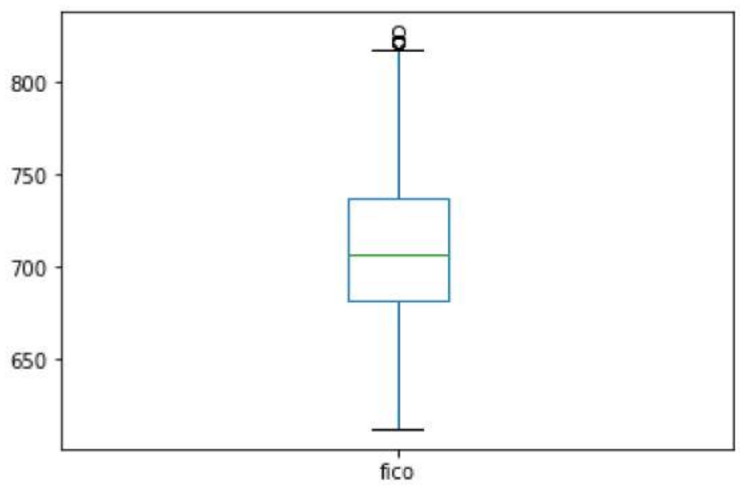
PLOT BOXPLOT of Different Variables

In [16]:

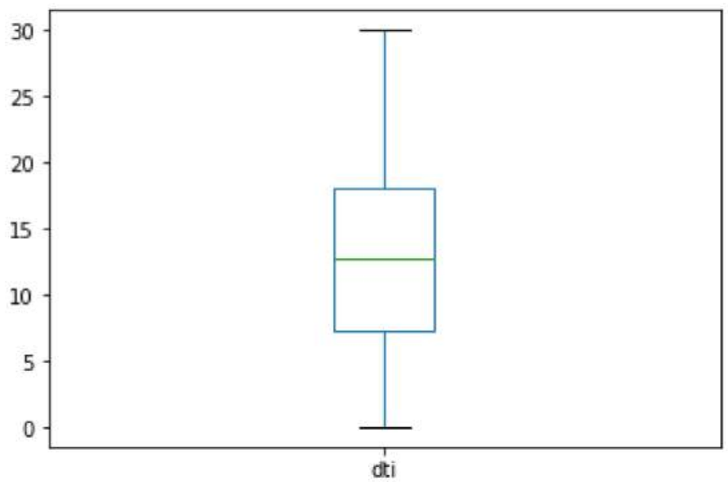
```
print('Boxplot of different variables to check for outliers :','\n\n ')
OutlierCheck_cols=['fico','dti','log.annual.inc']
 # function defined for finding value counts in respective columns
for col in OutlierCheck_cols:
    fig,axes = plt.subplots(1,1)
    data_frame1[col].plot(kind='box',ax=axes,subplots=True,
                                title=col)
```

```
Boxplot of different variables to check for outliers :
```

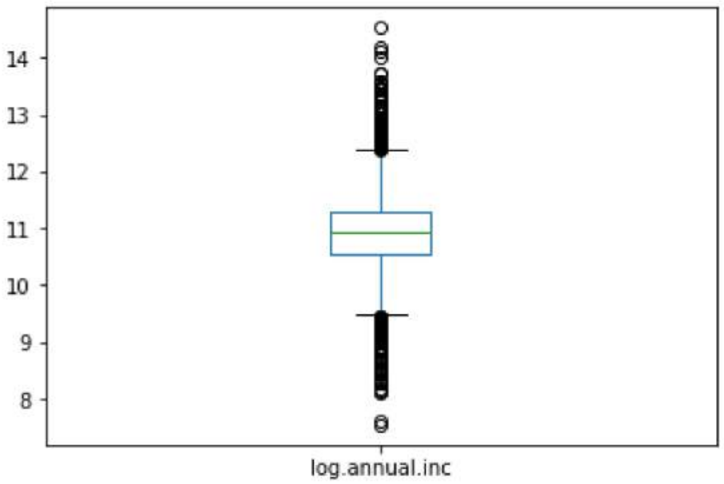
fico



dti



log.annual.inc



- In fico score we can see clearly that the values above 820 are outliers.
- In dti , no outliers are found
- In log.annual.inc column oultiers are found

Remove ouliers from 'fico' column

In [17]:
```
#find indexes of outliers in fico
print("Old Shape before outlier removal of fico: ", data_frame1.shape)
row_index=np.where(data_frame1['fico']>820)
row_index[0]
```

```
Old Shape before outlier removal of fico:  (9578, 19)
```

Out[17]:
```
array([ 154, 1477, 1613, 1883, 2476, 2495], dtype=int64)
```

In [18]:
```
#remove outliers from 'fico' column
data_frame1.drop( row_index[0], inplace = True)
print("New Shape after outlier removal of fico: ", data_frame1.shape)
```

```
New Shape after outlier removal of fico:  (9572, 19)
```

Remove ouliers from 'log.annual.inc' column

In [19]:

```
Q1 = np.percentile(data_frame1['log.annual.inc'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(data_frame1['log.annual.inc'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1
print("Old Shape before outlier removal in log.annual.inc: ", data_frame1.shape)
upper = np.where(data_frame1['log.annual.inc'] >= (Q3+1.5*IQR))
# Lower bound
lower = np.where(data_frame1['log.annual.inc'] <= (Q1-1.5*IQR))

''' Removing the Outliers '''
data_frame1.drop(upper[0], inplace = True)
data_frame1.drop(lower[0], inplace = True)

print("New Shape after outlier removal of log.annual.inc ", data_frame1.shape)
```

```
Old Shape before outlier removal in log.annual.inc:  (9572, 19)
New Shape after outlier removal of log.annual.inc  (9334, 19)
```

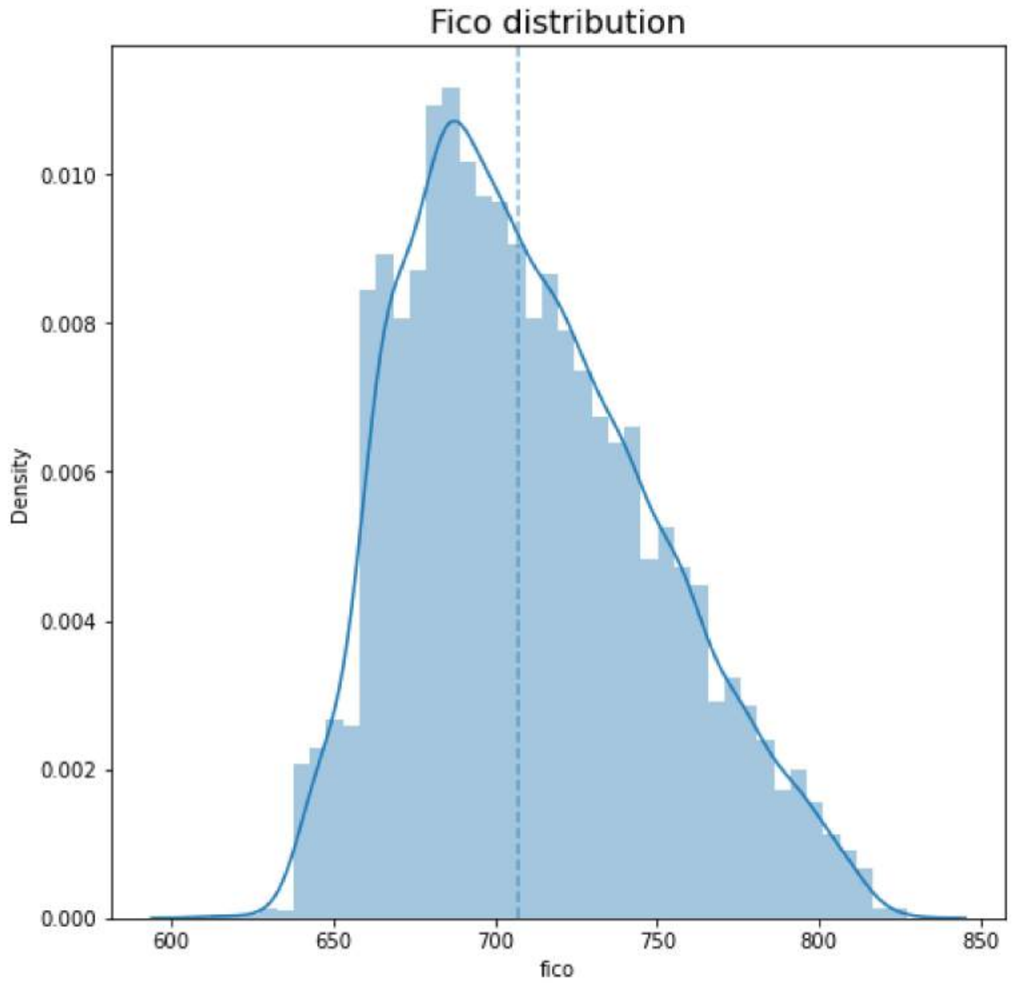
Understand the distribution of fico

In [20]:

```
fig, (ax1) = plt.subplots(1, 1, figsize=(8, 8))
sns.distplot(data_frame["fico"], ax=ax1)
ax1.set_title("Fico distribution", fontsize=16);
ax1.axvline(x=data_frame["fico"].median(), linestyle="--", alpha=0.5)
```

Out[20]:

```
<matplotlib.lines.Line2D at 0x1ab7df238b0>
```



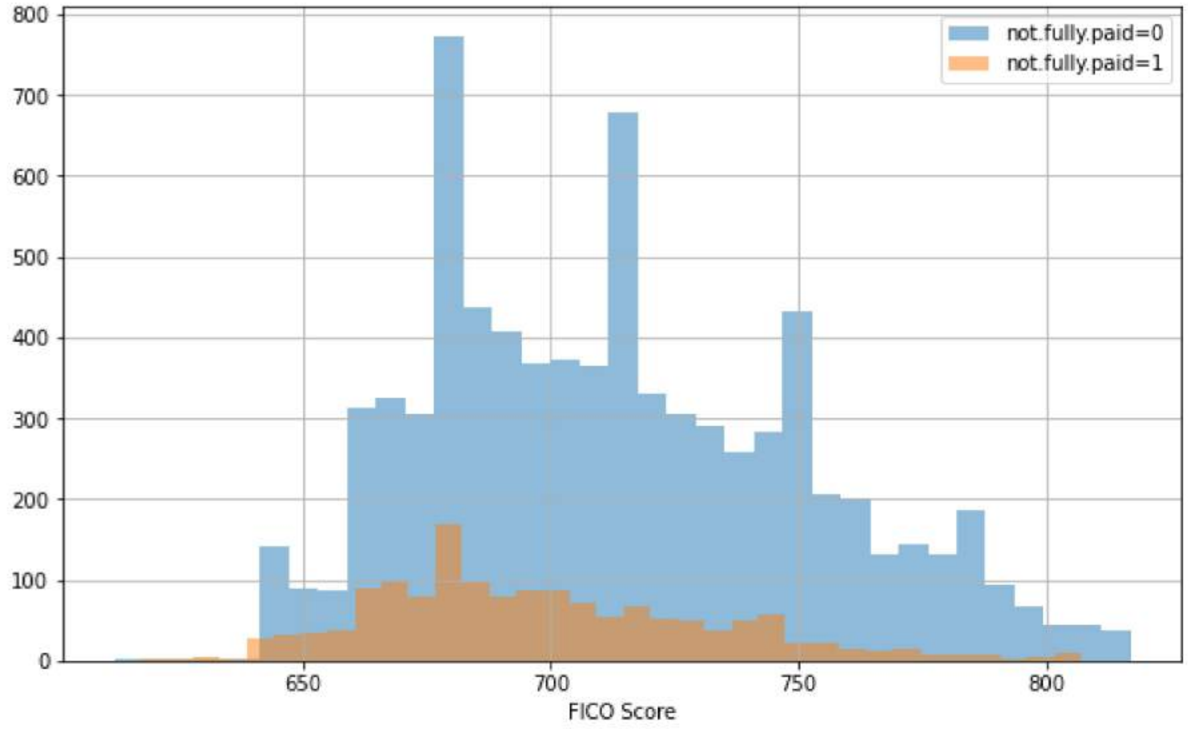
- Distribution of Fico is left skewed.

Histogram of two FICO distributions on top of each other for Loan fully Paid or not

In [21]:
```
plt.figure(figsize=(10,6))

data_frame1[data_frame1['not.fully.paid']==0]['fico'].hist(bins=35,alpha=0.5,label='not.fully.paid=0')
data_frame1[data_frame1['not.fully.paid']==1]['fico'].hist(bins=35,alpha=0.5,label='not.fully.paid=1')
plt.xlabel('FICO Score')
plt.legend()
```

Out[21]: <matplotlib.legend.Legend at 0x1ab7dedc850>

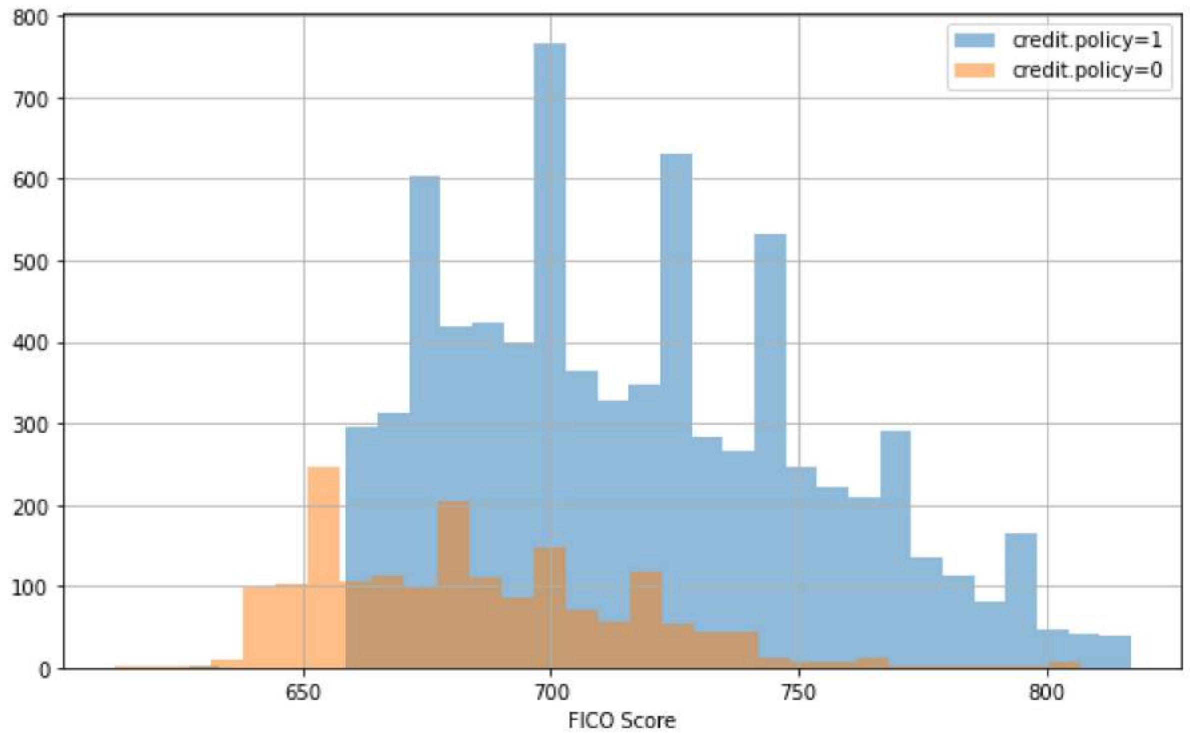


Histogram of two FICO distributions on top of each other,one for each credit.policy

In [22]:

```
plt.figure(figsize=(10,6))
data_frame1[data_frame1['credit.policy']==1]['fico'].hist(bins=30,alpha=0.5,label='credit.policy=1')
data_frame1[data_frame1['credit.policy']==0]['fico'].hist(bins=30,alpha=0.5,label='credit.policy=0')
plt.xlabel('FICO Score')
plt.legend()
```

Out[22]: <matplotlib.legend.Legend at 0x1ab7e08ed00>



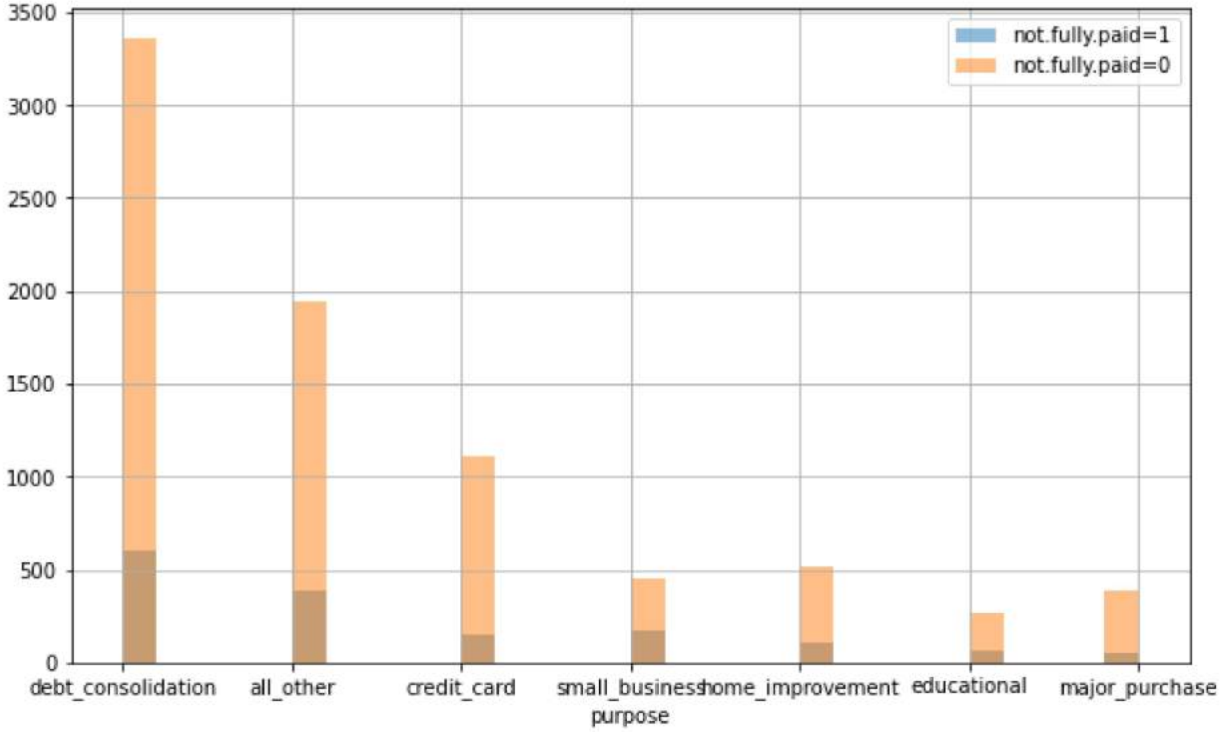
- From figure its understood that People with high FICO scores tend to meet the credit underwriting criteria.

Analysis of Loan Fully paid or not with the Purpose of Loan

In [23]:

```
plt.figure(figsize=(10,6))
data_frame[data_frame['not.fully.paid']==1]['purpose'].hist(bins=30,alpha=0.5,label='not.fully.paid=1')
data_frame[data_frame['not.fully.paid']==0]['purpose'].hist(bins=30,alpha=0.5,label='not.fully.paid=0')
plt.xlabel('purpose')
plt.legend()
```

Out[23]: <matplotlib.legend.Legend at 0x1ab7f37f850>



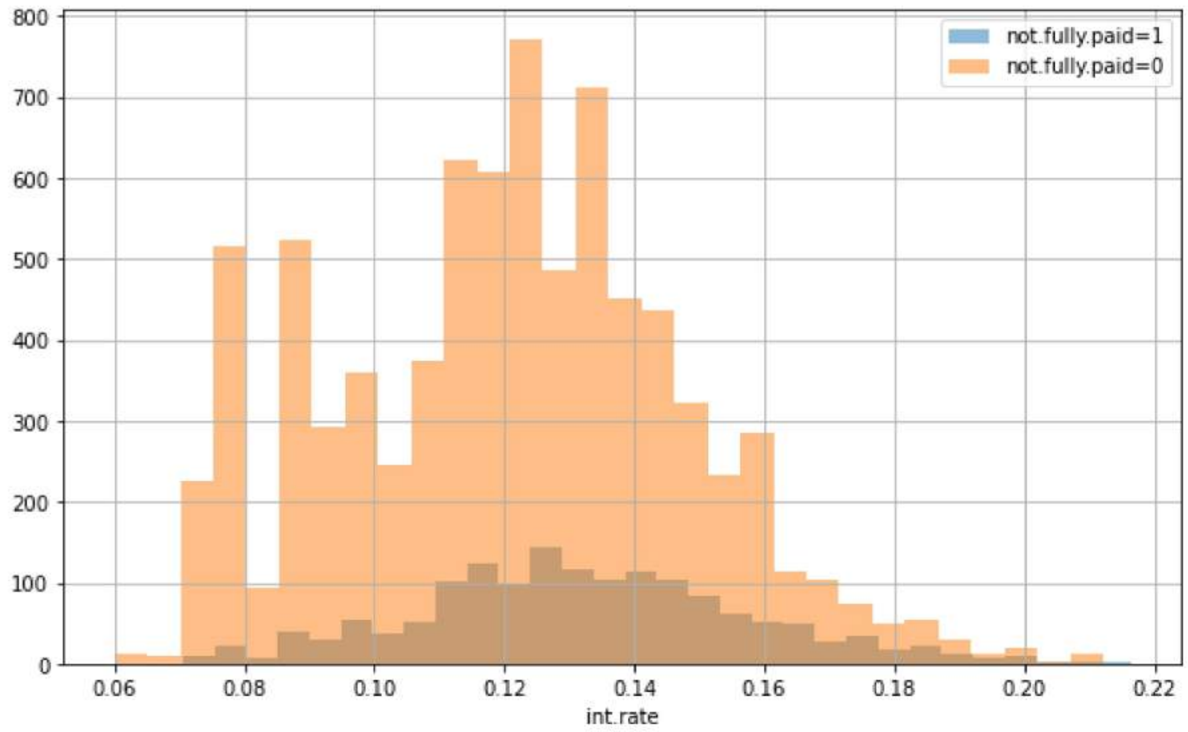
Analysis of Interest rate Versus Loan Fully Paid or not

In [24]:

```
plt.figure(figsize=(10,6))
data_frame[data_frame['not.fully.paid']==1]['int.rate'].hist(bins=30,alpha=0.5,label='not.fully.paid=1')
data_frame[data_frame['not.fully.paid']==0]['int.rate'].hist(bins=30,alpha=0.5,label='not.fully.paid=0')
plt.xlabel('int.rate')
plt.legend()
```

Out[24]:

```
<matplotlib.legend.Legend at 0x1ab7d62ea30>
```

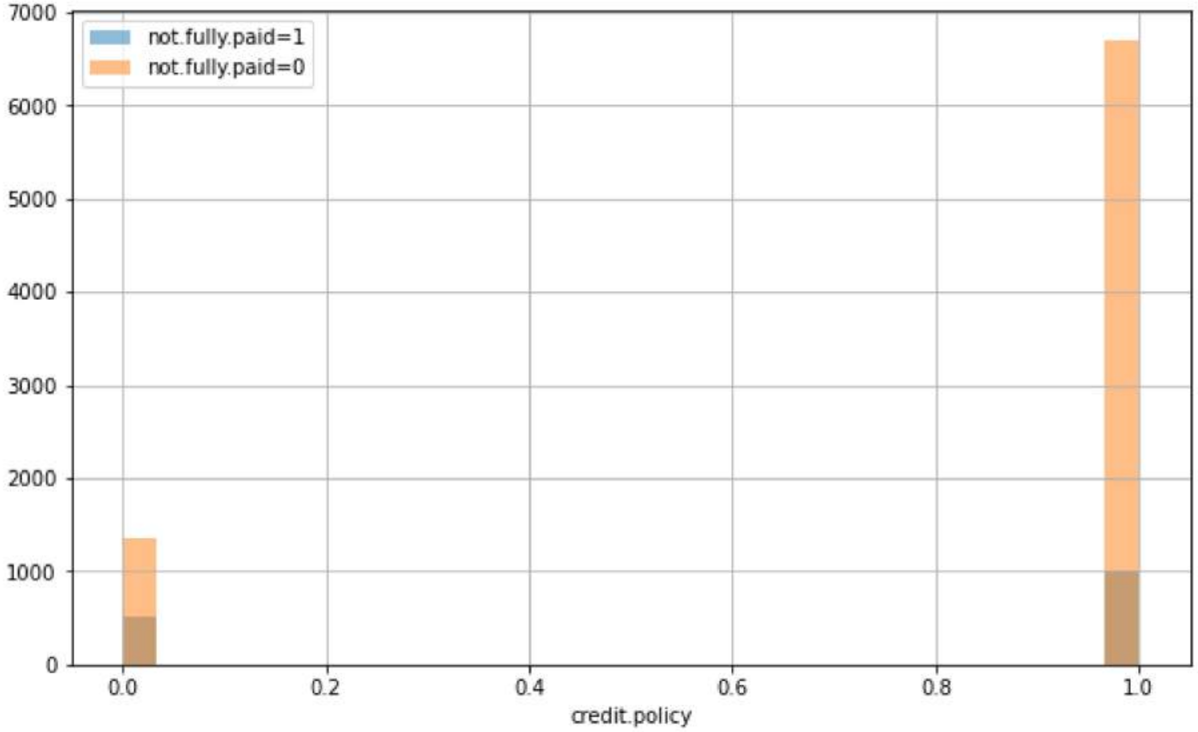


Analysis of Loan Fully Paid or Not Versus Credit Policy.

In [25]:

```
plt.figure(figsize=(10,6))
data_frame[data_frame['not.fully.paid']==1]['credit.policy'].hist(bins=30,alpha=0.5,label='not.fully.paid=1')
data_frame[data_frame['not.fully.paid']==0]['credit.policy'].hist(bins=30,alpha=0.5,label='not.fully.paid=0')
plt.xlabel('credit.policy')
plt.legend()
```

Out[25]: <matplotlib.legend.Legend at 0x1ab7e158640>

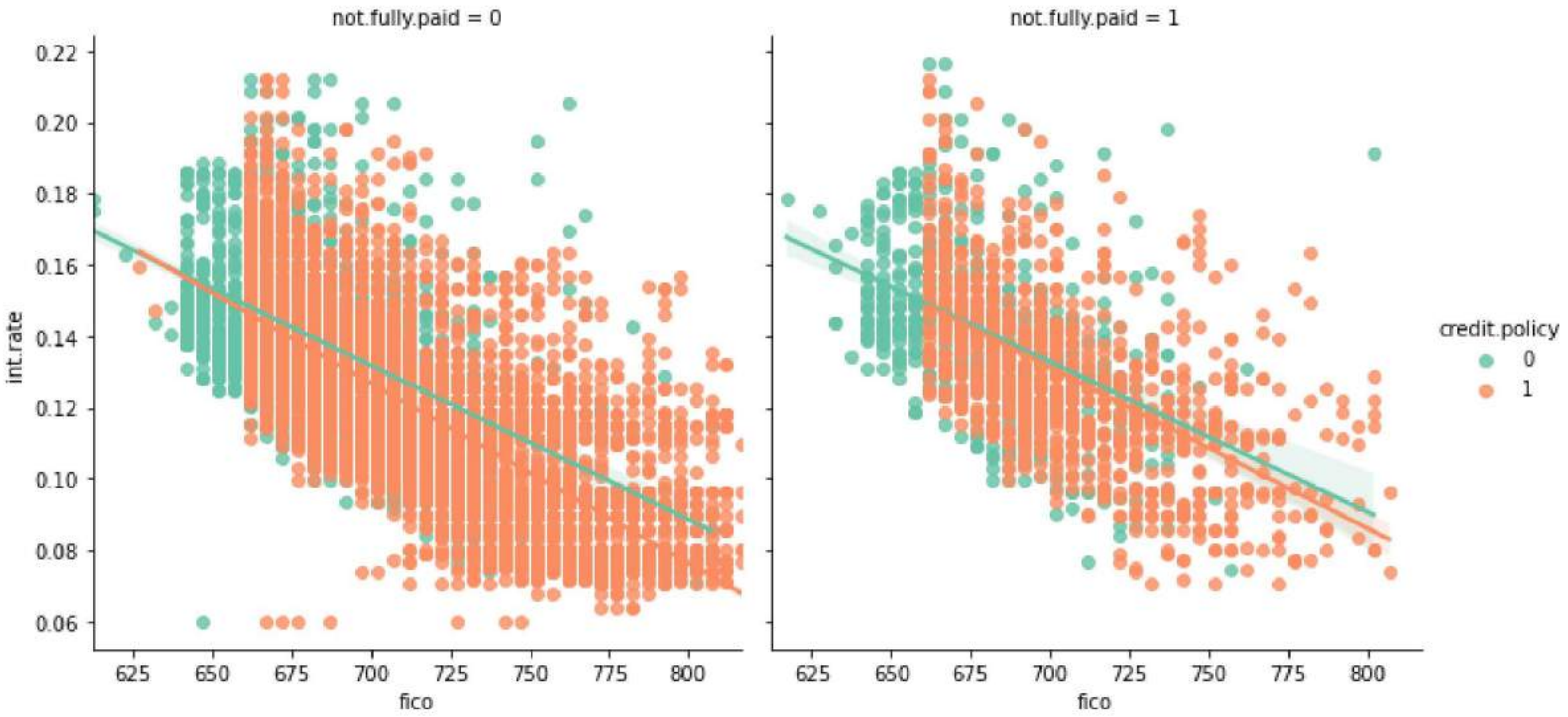


In [26]:

```
sns.lmplot(x='fico',y='int.rate',data=data_frame1,col='not.fully.paid',hue='credit.policy',palette='Set2')
```

Out[26]:

```
<seaborn.axisgrid.FacetGrid at 0x1ab7f5a0460>
```



- FICO score and interest rate has similar pattern of change for those who fully paid balances versus those who did not paid.

Check for Correlation of features in dataset

In [27]:

```
#Correlation of features
df_corr= data_frame1.corr()
df_corr
```

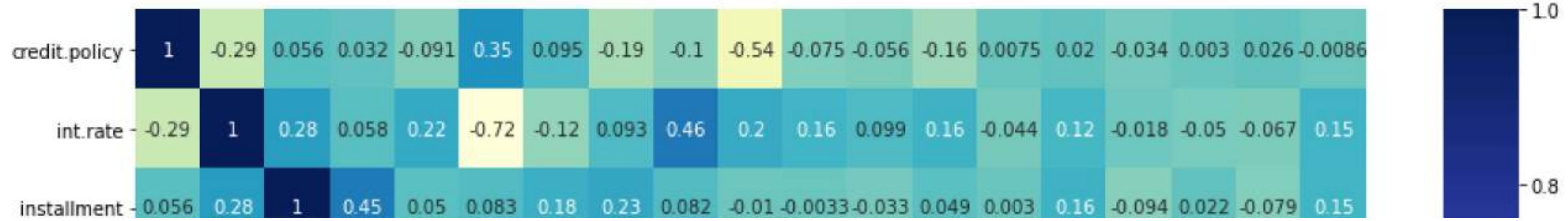
Out[27]:

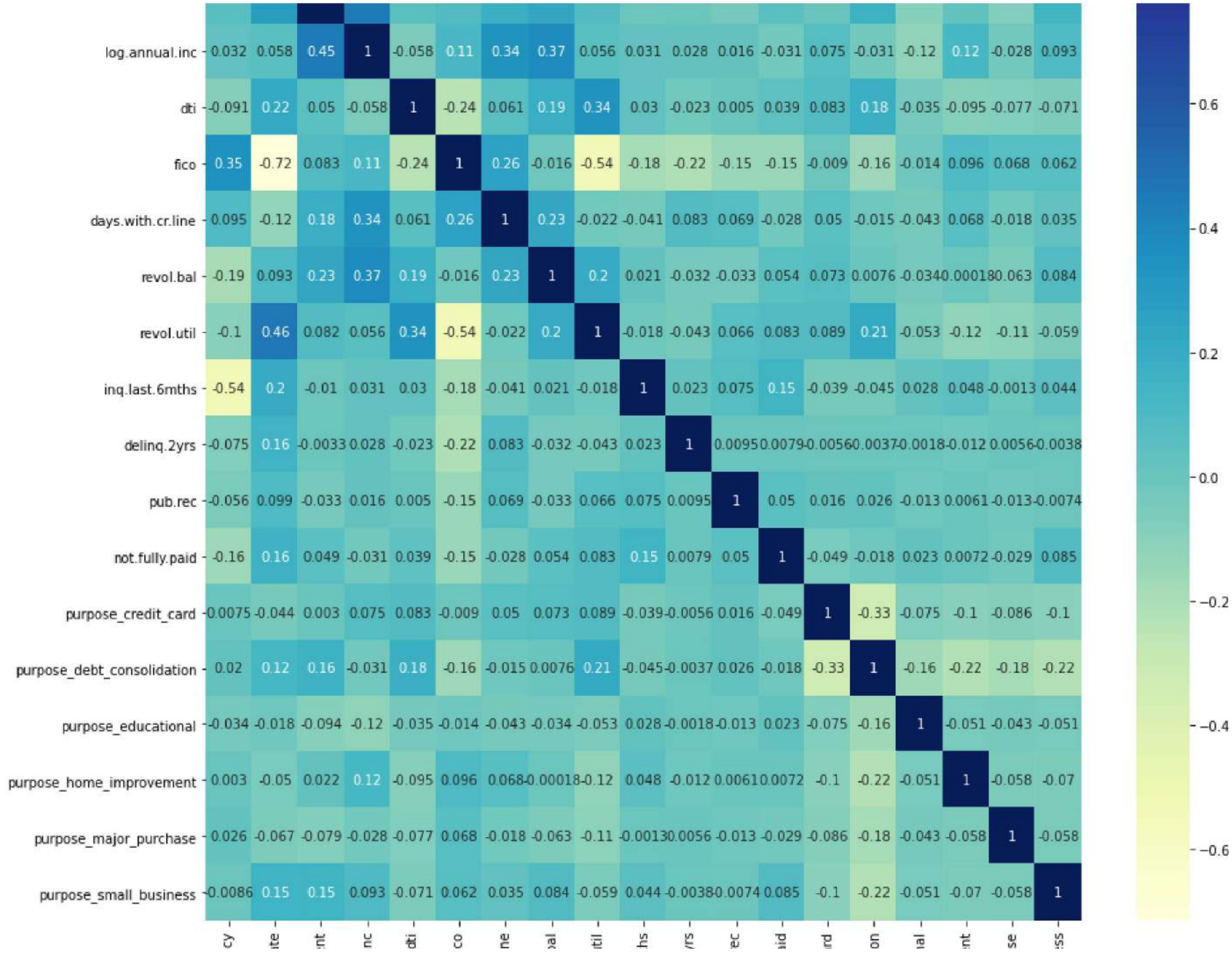
	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths
credit.policy	1.000000	-0.294582	0.055518	0.031517	-0.090858	0.345000	0.095135	-0.186928	-0.102826	-0.537847
int.rate	-0.294582	1.000000	0.277746	0.057742	0.218237	-0.715142	-0.122209	0.093105	0.462989	0.202904
installment	0.055518	0.277746	1.000000	0.450463	0.050022	0.083278	0.184023	0.232173	0.082150	-0.010424
log.annual.inc	0.031517	0.057742	0.450463	1.000000	-0.057753	0.111774	0.335185	0.373148	0.055984	0.030732

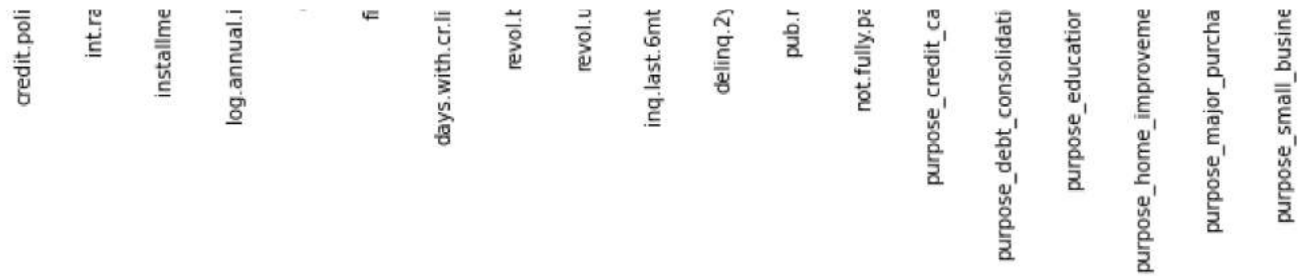
	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths
dti	-0.090858	0.218237	0.050022	-0.057753	1.000000	-0.238939	0.061088	0.186974	0.335072	0.029687
fico	0.345000	-0.715142	0.083278	0.111774	-0.238939	1.000000	0.260136	-0.015574	-0.540566	-0.183939
days.with.cr.line	0.095135	-0.122209	0.184023	0.335185	0.061088	0.260136	1.000000	0.229797	-0.022477	-0.041183
revol.bal	-0.186928	0.093105	0.232173	0.373148	0.186974	-0.015574	0.229797	1.000000	0.204526	0.021266
revol.util	-0.102826	0.462989	0.082150	0.055984	0.335072	-0.540566	-0.022477	0.204526	1.000000	-0.017505
inq.last.6mths	-0.537847	0.202904	-0.010424	0.030732	0.029687	-0.183939	-0.041183	0.021266	-0.017505	1.000000
delinq.2yrs	-0.075431	0.155277	-0.003266	0.028205	-0.023224	-0.215174	0.083218	-0.032301	-0.043102	0.023087
pub.rec	-0.055814	0.098518	-0.033497	0.015728	0.005021	-0.149431	0.069286	-0.033377	0.066443	0.074544
not.fully.paid	-0.158960	0.160133	0.049029	-0.031388	0.038897	-0.148112	-0.027755	0.054319	0.082774	0.147901
purpose_credit_card	0.007528	-0.044191	0.002966	0.074589	0.082752	-0.008976	0.050333	0.073265	0.088811	-0.039209
purpose_debt_consolidation	0.019816	0.123191	0.160014	-0.030545	0.179380	-0.155573	-0.014511	0.007556	0.214491	-0.045108
purpose_educational	-0.033714	-0.017986	-0.094271	-0.116598	-0.035130	-0.013916	-0.043150	-0.033866	-0.053009	0.027598
purpose_home_improvement	0.003032	-0.050385	0.022441	0.118513	-0.094690	0.095947	0.068403	-0.000183	-0.115709	0.048133
purpose_major_purchase	0.025803	-0.067241	-0.079414	-0.027868	-0.076987	0.067826	-0.018430	-0.062802	-0.108094	-0.001342
purpose_small_business	-0.008579	0.152697	0.145881	0.092785	-0.070673	0.062117	0.034734	0.083549	-0.059315	0.044470

In [28]:

```
# Plot the heatmap for correlation analysis
plt.figure(figsize=(15,15))
sns.heatmap(df_corr,annot=True,cmap="YlGnBu")
plt.show()
```




log.annual.inc
dti
fico
days.with.cr.line
revol.bal
revol.util
inq.last.6mths
delinq.2yrs
pub.rec
not.fully.paid
purpose_credit_card
purpose_debt_consolidation
purpose_educational
purpose_home_improvement
purpose_major_purchase
purpose_small_business
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6



- From the figure we can see that int.rate & revol.util has correlation coefficient of 0.47
- log.annual.inc & installment has correlation coefficient of 0.44
- Since no two varibles have correlation coefficient above 0.5, I am not dropping any columns

Create training and test dataset

In [29]:

```
#Seperate feature variables and target variable
x_data=data_frame1.drop('not.fully.paid',axis=1)
y_data=data_frame1['not.fully.paid']
```

In [30]:

```
#train-test splitting
from sklearn.model_selection import train_test_split
x_train,x_test,y_train, y_test = train_test_split(x_data,y_data,test_size=0.25,random_state=42)
```

SMOTE(Synthetic Minority Over Sampling Technique)

To perform oversampling to resolve the issue of DATA IMBALANCE.

In [31]:

```
from imblearn.over_sampling import SMOTE
smote=SMOTE(sampling_strategy='minority',random_state=42)
X_sm,y_sm=smote.fit_resample(x_train,y_train)
```

In [32]:

```
y_sm.value_counts()
```

Out[32]:

```
0    5871
1    5871
Name: not.fully.paid, dtype: int64
```

In [33]:
```
X_sm.shape
```
Out[33]: (11742, 18)

In [34]:
```
y_sm.shape
```
Out[34]: (11742,)

Build the Model

Steps for buliding Deep Learning Model

1.Define the model

2.Compiling the model

3.Fitting the model

4.Evaluating the model

5.Make Predictions

In [35]:
```
# import necessary libraries from tensorflow,keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization,Reshape
```

In [36]:
```
# Define the model

model = Sequential()
#Add initial layer
model.add(Reshape((18,),input_shape=(18,))) # 18 is no of columns in input xtrain
model.add(BatchNormalization())#Normalize the data
```

```
#Add 1st hidden layer
model.add(Dense(3000, activation="relu"))
model.add(BatchNormalization())

#Add 2nd hidden layer
model.add(Dense(1200, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.2))# Add droupout

#Add 3rd hidden layer
model.add(Dense(600, activation="relu"))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Add output layer
model.add(Dense(100, activation="relu"))
model.add(BatchNormalization())

# Add output layer
model.add(Dense(1, activation='sigmoid'))
```

In [37]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape (Reshape)           (None, 18)                0

 batch_normalization (BatchN  (None, 18)               72
 ormalization)

 dense (Dense)               (None, 3000)              57000

 batch_normalization_1 (Batc  (None, 3000)             12000
 hNormalization)

 dense_1 (Dense)             (None, 1200)              3601200

 batch_normalization_2 (Batc  (None, 1200)             4800
 hNormalization)

 dropout (Dropout)           (None, 1200)              0
```

```
 dense_2 (Dense)             (None, 600)               720600

 batch_normalization_3 (Batc  (None, 600)              2400
 hNormalization)

 dropout_1 (Dropout)         (None, 600)               0

 dense_3 (Dense)             (None, 100)               60100

 batch_normalization_4 (Batc  (None, 100)              400
 hNormalization)

 dense_4 (Dense)             (None, 1)                 101

=================================================================
Total params: 4,458,673
Trainable params: 4,448,837
Non-trainable params: 9,836
_________________________________________________________________
```

In [38]:

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

In [39]:

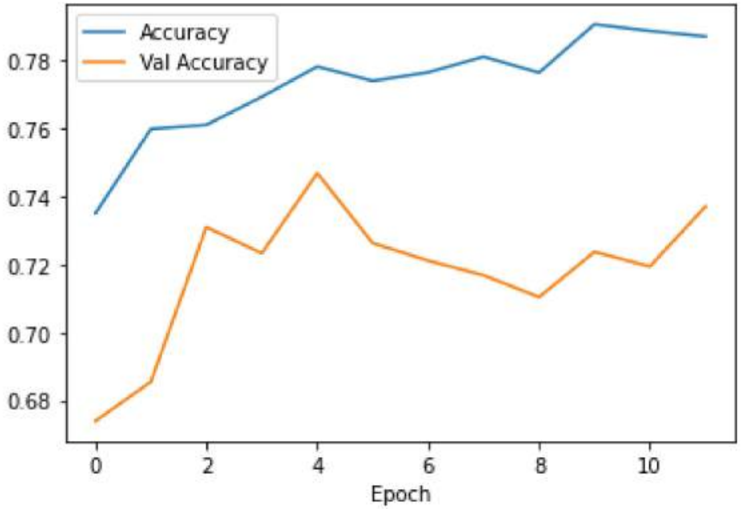
```
# Fit the model
h1= model.fit(X_sm,y_sm,
         validation_data=(x_test,y_test),
         epochs=12,
         batch_size=32)
```

```
Epoch 1/12
367/367 [==============================] - 18s 43ms/step - loss: 0.5424 - accuracy: 0.7350 - val_loss: 0.6073 - val_accuracy: 0.67
40
Epoch 2/12
367/367 [==============================] - 16s 43ms/step - loss: 0.4993 - accuracy: 0.7598 - val_loss: 0.6329 - val_accuracy: 0.68
55
Epoch 3/12
367/367 [==============================] - 18s 49ms/step - loss: 0.4885 - accuracy: 0.7609 - val_loss: 0.5983 - val_accuracy: 0.73
09
Epoch 4/12
367/367 [==============================] - 18s 48ms/step - loss: 0.4833 - accuracy: 0.7692 - val_loss: 0.5847 - val_accuracy: 0.72
32
Epoch 5/12
```

```
367/367 [==============================] - 17s 47ms/step - loss: 0.4728 - accuracy: 0.7781 - val_loss: 0.5655 - val_accuracy: 0.74
68
Epoch 6/12
367/367 [==============================] - 17s 48ms/step - loss: 0.4740 - accuracy: 0.7739 - val_loss: 0.5909 - val_accuracy: 0.72
62
Epoch 7/12
367/367 [==============================] - 17s 47ms/step - loss: 0.4706 - accuracy: 0.7764 - val_loss: 0.5965 - val_accuracy: 0.72
11
Epoch 8/12
367/367 [==============================] - 19s 51ms/step - loss: 0.4636 - accuracy: 0.7810 - val_loss: 0.6285 - val_accuracy: 0.71
68
Epoch 9/12
367/367 [==============================] - 17s 47ms/step - loss: 0.4691 - accuracy: 0.7763 - val_loss: 0.6111 - val_accuracy: 0.71
04
Epoch 10/12
367/367 [==============================] - 18s 49ms/step - loss: 0.4584 - accuracy: 0.7905 - val_loss: 0.5795 - val_accuracy: 0.72
37
Epoch 11/12
367/367 [==============================] - 18s 49ms/step - loss: 0.4563 - accuracy: 0.7885 - val_loss: 0.6355 - val_accuracy: 0.71
94
Epoch 12/12
367/367 [==============================] - 18s 48ms/step - loss: 0.4538 - accuracy: 0.7869 - val_loss: 0.5859 - val_accuracy: 0.73
69
```

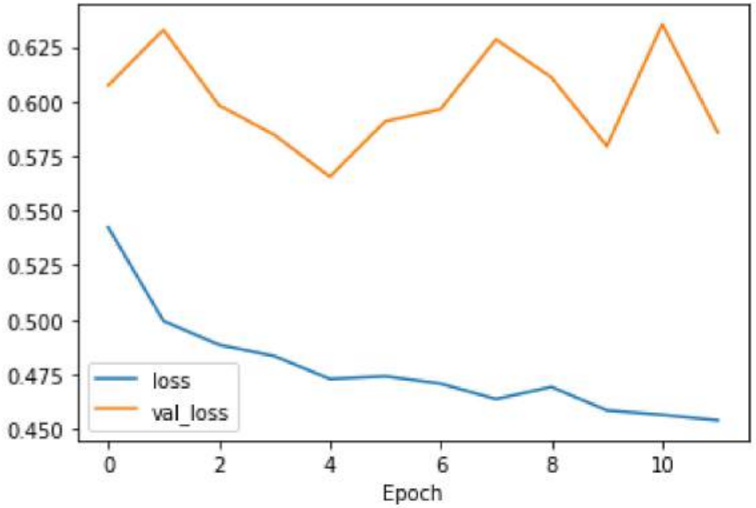
In [40]:

```
# Plot the Accuracy values
plt.plot(h1.history["accuracy"],label = 'Accuracy')
plt.plot(h1.history["val_accuracy"], label = 'Val Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



In [41]:

```
# Plot the loss values
plt.plot(h1.history["loss"],label = 'loss')
plt.plot(h1.history["val_loss"],label = 'val_loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



In [42]:

```
#Evaluate the model
```

```
model.evaluate(x_test, y_test, batch_size=32)
```

```
73/73 [==============================] - 1s 11ms/step - loss: 0.5859 - accuracy: 0.7369
```

Out[42]:
```
[0.5858753323554993, 0.7369322776794434]
```

In [43]:
```
## Make predictions
ypred = model.predict(x_test)
ypred
```

Out[43]:
```
array([[0.14280865],
       [0.01753595],
       [0.07088828],
       ...,
       [0.49807814],
       [0.22936234],
       [0.43922397]], dtype=float32)
```