dl-project

March 22, 2023

[1]:
```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
print('All Libraries are imported')
```

```
All Libraries are imported
```

[2]:
```
#load the data set
df = pd.read_csv('loan_data.csv')
```

[3]:
```
df.head()
```

[3]:

	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.rec	not.fully.paid
0	1	debt_consolidation	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	credit_card	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	debt_consolidation	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	debt_consolidation	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	credit_card	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0

[4]:
```
df.shape
```

[4]:
```
(9578, 14)
```

[5]:
```
df.describe()
```

[5]:

	credit.policy	int.rate	installment	log.annual.inc	dti
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000
mean	0.804970	0.122640	319.089413	10.932117	12.606679
std	0.396245	0.026847	207.071301	0.614813	6.883970
min	0.000000	0.060000	15.670000	7.547502	0.000000
25%	1.000000	0.103900	163.770000	10.558414	7.212500
50%	1.000000	0.122100	268.950000	10.928884	12.665000
75%	1.000000	0.140700	432.762500	11.291293	17.950000
max	1.000000	0.216400	940.140000	14.528354	29.960000

	fico	days.with.cr.line	revol.bal	revol.util
count	9578.000000	9578.000000	9.578000e+03	9578.000000
mean	710.846314	4560.767197	1.691396e+04	46.799236
std	37.970537	2496.930377	3.375619e+04	29.014417
min	612.000000	178.958333	0.000000e+00	0.000000
25%	682.000000	2820.000000	3.187000e+03	22.600000
50%	707.000000	4139.958333	8.596000e+03	46.300000
75%	737.000000	5730.000000	1.824950e+04	70.900000
max	827.000000	17639.958330	1.207359e+06	119.000000

	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
count	9578.000000	9578.000000	9578.000000	9578.000000
mean	1.577469	0.163708	0.062122	0.160054
std	2.200245	0.546215	0.262126	0.366676
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000
75%	2.000000	0.000000	0.000000	0.000000
max	33.000000	13.000000	5.000000	1.000000

[6]:
```
df.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
```

[7]:
```
# Missing value
df.isnull().sum().any()
```

[7]:
```
False
```

[9]:
```
df['not.fully.paid'].value_counts()
# 0-fully paid, 1-not paid
# imbalanced data
```

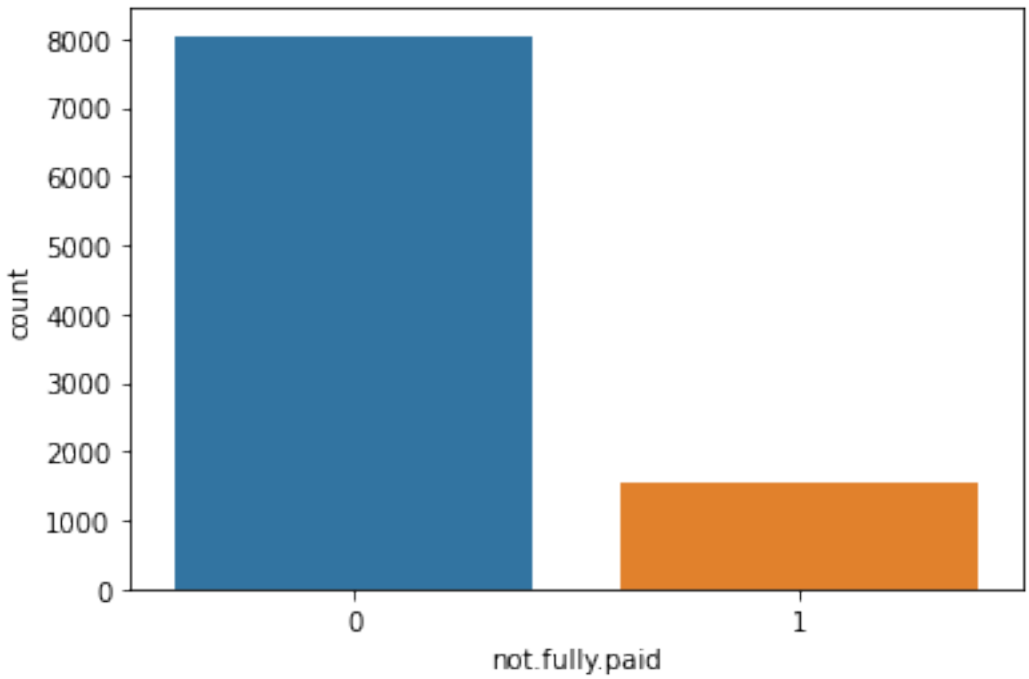
[9]:
```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

Exploratory data analysis of different factors of the data set

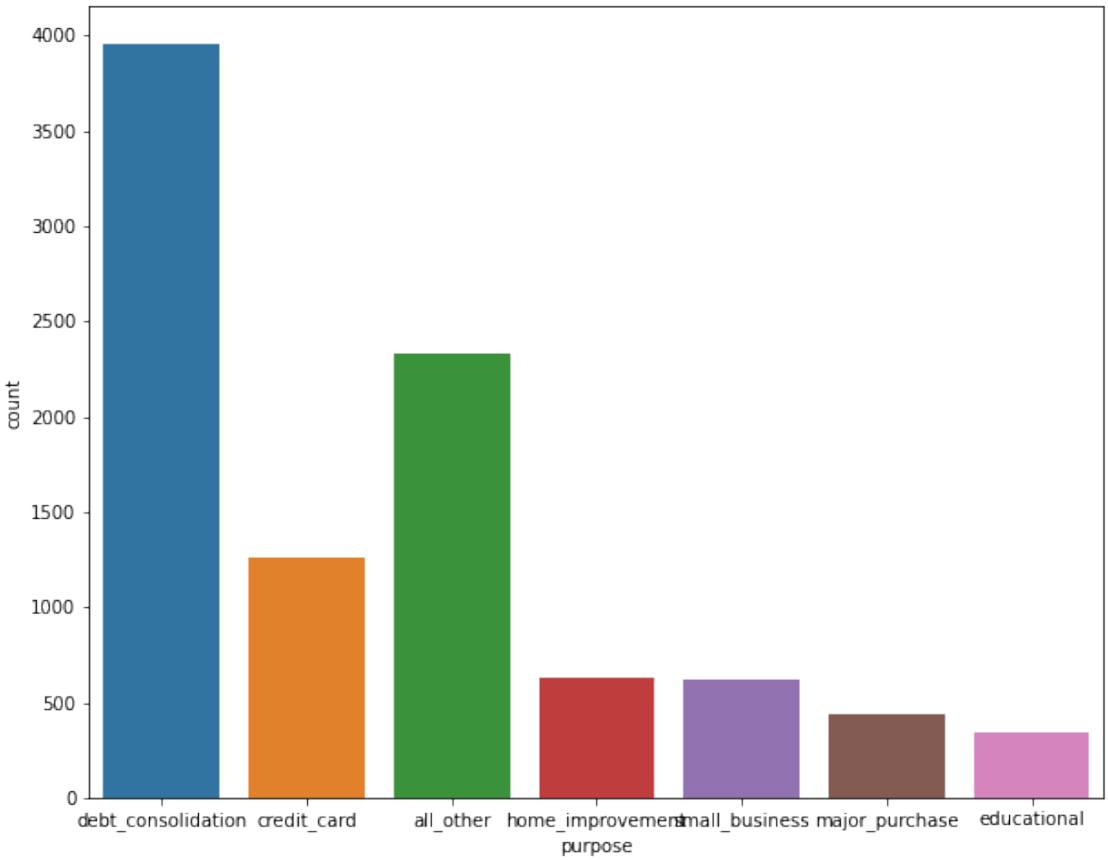
[10]:
```
df.dtypes
```

[10]:
```
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
```

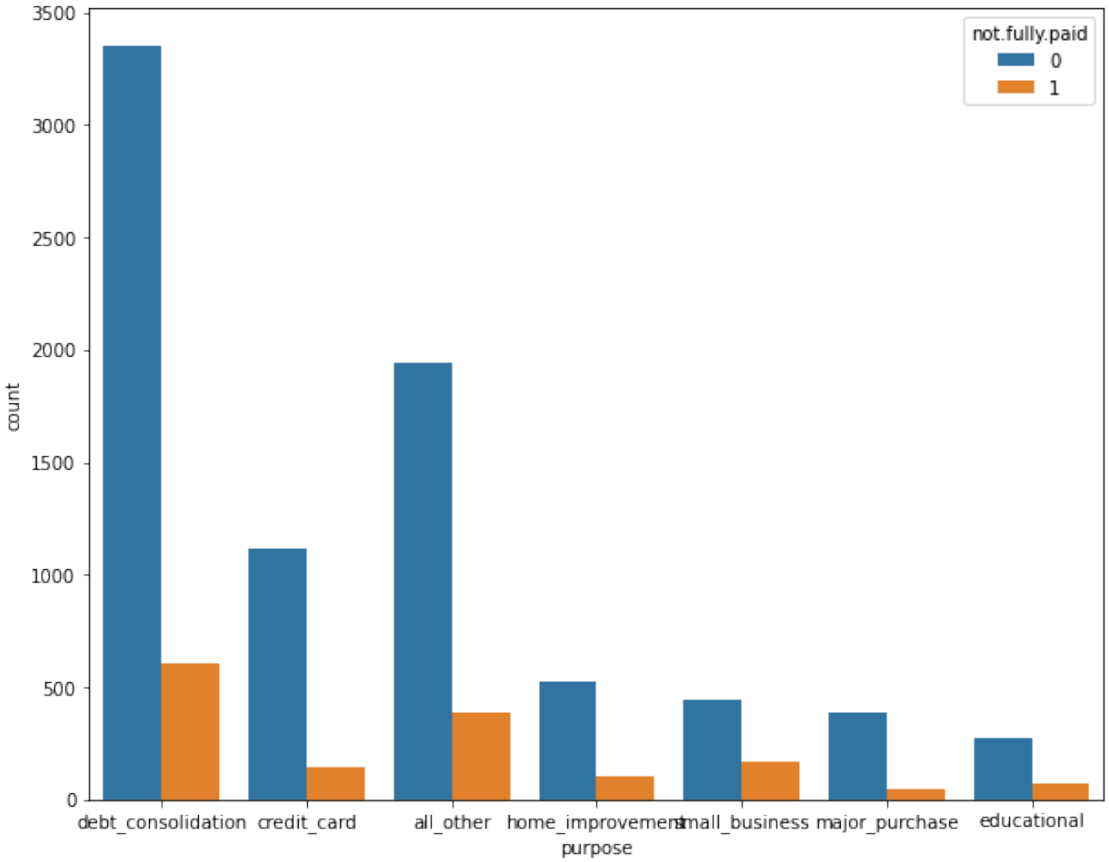
[11]:
```
sns.countplot(x=df['not.fully.paid'])
plt.show()
```



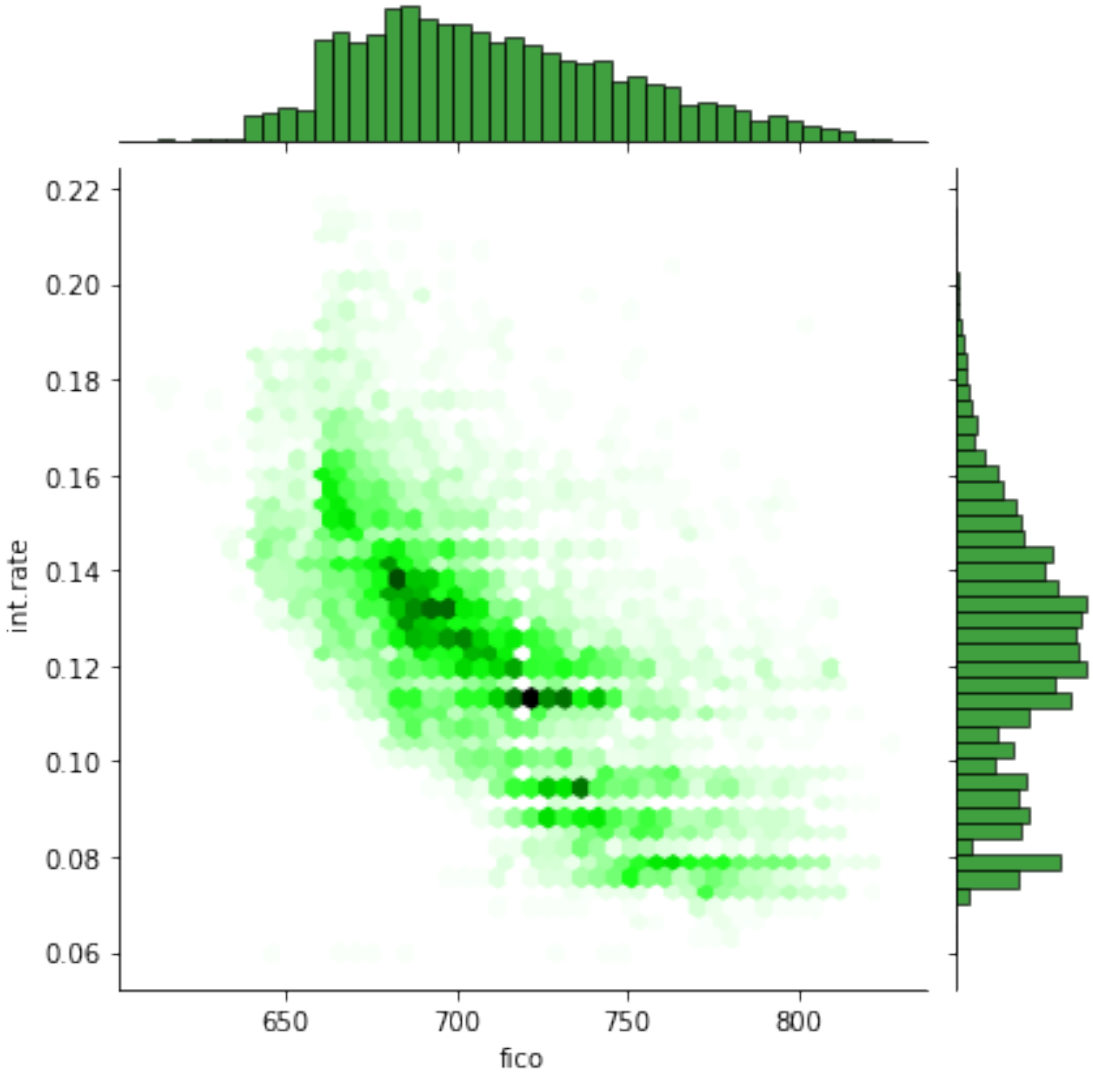
[12]:
```
plt.figure(figsize=(10,8))
sns.countplot(x=df['purpose'])
plt.show()
```



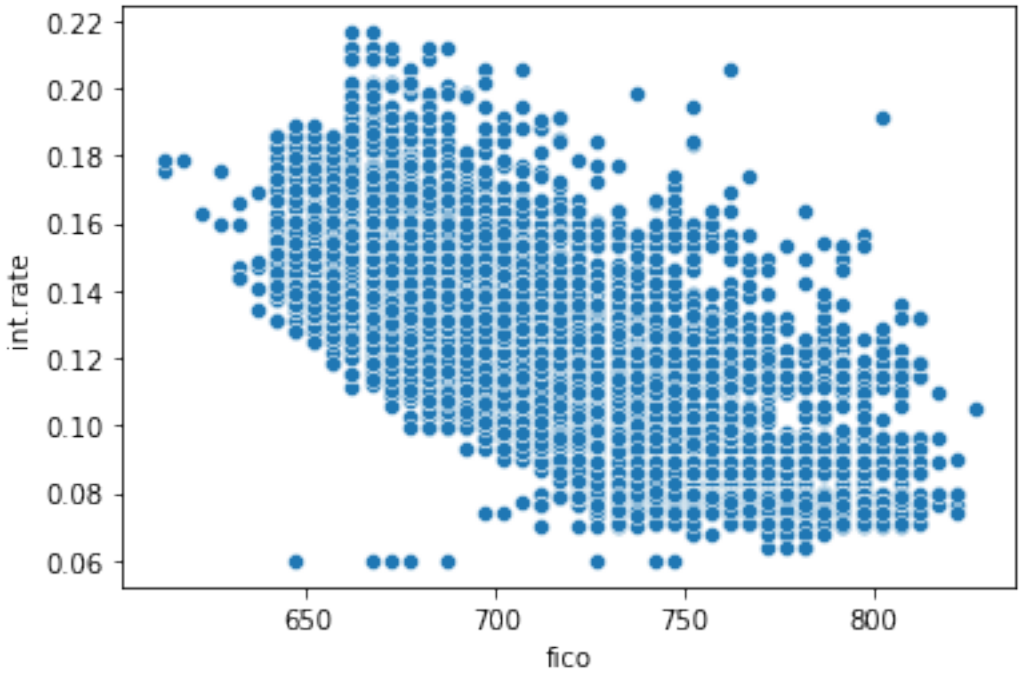
[13]:
```
# purpose----- not fully paid
plt.figure(figsize=(10,8))
sns.countplot(x=df['purpose'],hue='not.fully.paid',data=df)
plt.show()
```



[14]:
```
# Bi-variate Analysis
sns.jointplot(x='fico',y='int.rate',data=df,kind='hex',color='g')
plt.show()
```

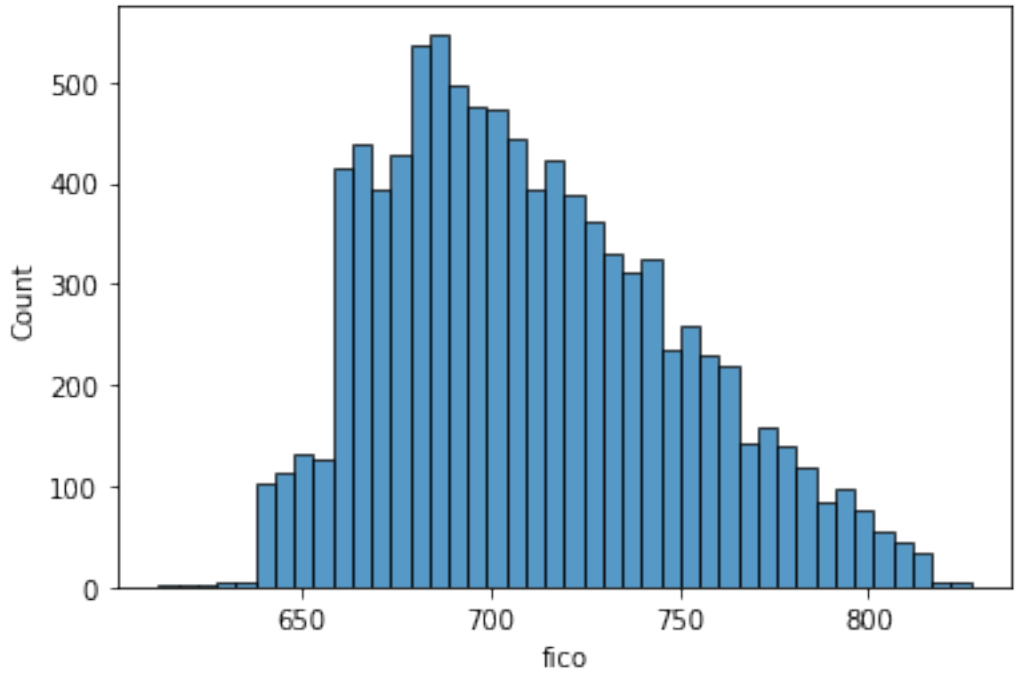


[15]:
```
sns.scatterplot(x='fico',y='int.rate',data=df)
plt.show()
```

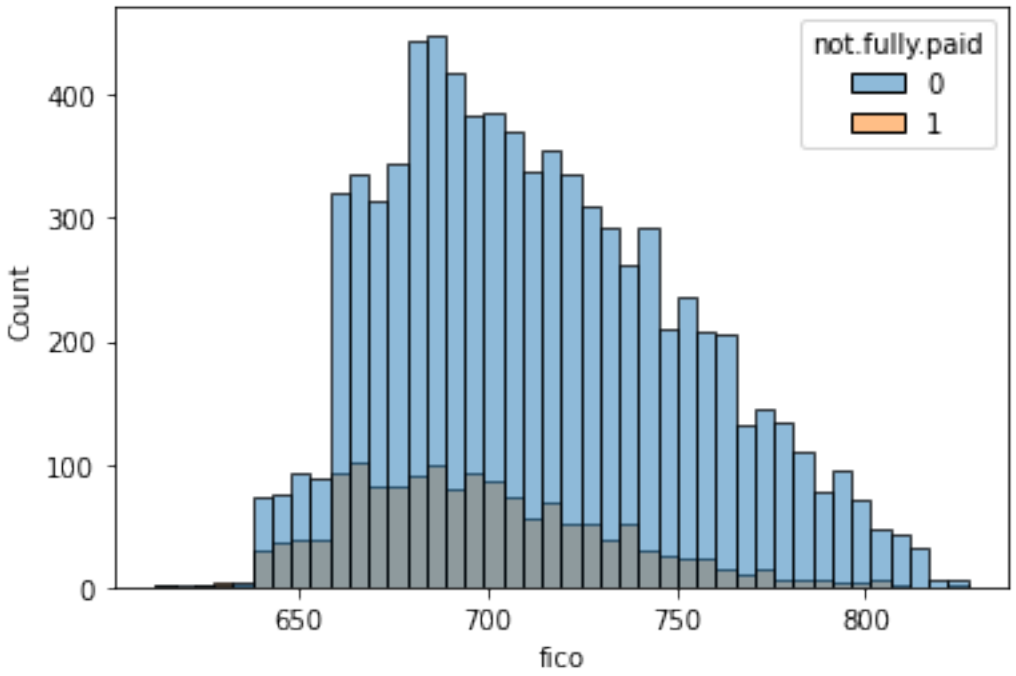


[16]:
```
sns.histplot(df['fico'])
```

[16]: <AxesSubplot:xlabel='fico', ylabel='Count'>



[17]:
```
sns.histplot(x='fico',hue='not.fully.paid',data=df)
plt.show()
```



Feature Transformation

Transforming categorical values into numerical values(discrete)

[36]:
```
# Handle imbalanced data set
df['not.fully.paid'].value_counts()
```

[36]:
```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

[37]:
```
not_fully_paid_0=df[df['not.fully.paid']==0]
not_fully_paid_1=df[df['not.fully.paid']==1]
```

[38]:
```
not_fully_paid_0.shape
```

[38]:
```
(8045, 14)
```

[39]:
```
not_fully_paid_1.shape
```

[39]:
```
(1533, 14)
```

[40]:
```
#resample
from sklearn.utils import resample
df_minor_upsample = resample(not_fully_paid_1,replace=True,n_samples=8045)
```

[43]:
```
new_df = pd.concat([not_fully_paid_0,df_minor_upsample])
```

[44]:
```
# shuffle
from sklearn.utils import shuffle
new_df=shuffle(new_df)
```

[46]:
```
new_df['not.fully.paid'].value_counts()
```

[46]:
```
1    8045
0    8045
Name: not.fully.paid, dtype: int64
```

[47]:
```
new_df.shape
```

[47]:
```
(16090, 14)
```

[48]:
```
new_df.dtypes
```

[48]:
```
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
```

[49]:
```
# convert purpose into numeric data type
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

[50]:
```
for i in new_df.columns:
    if new_df[i].dtypes =='object':
        new_df[i]=le.fit_transform(new_df[i])
```

[51]:
```
new_df.dtypes
```

[51]:

```
credit.policy          int64
purpose                int64
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
```

Additional Feature Engineering

You will check the correlation between features and will drop those features which have a strong correlation

This will help reduce the number of features and will leave you with the most relevant features

[52]:

```python
new_df.corr()
```

[52]:

```
                   credit.policy   purpose  int.rate  installment  \
credit.policy           1.000000  0.009956 -0.299594     0.057822
purpose                 0.009956  1.000000  0.134248     0.198623
int.rate               -0.299594  0.134248  1.000000     0.264181
installment             0.057822  0.198623  0.264181     1.000000
log.annual.inc          0.012198  0.113946  0.079315     0.473751
dti                    -0.099022 -0.048576  0.214052     0.018926
fico                    0.377818  0.069824 -0.682865     0.117298
days.with.cr.line       0.096084  0.059256 -0.097441     0.180832
revol.bal              -0.187238  0.069319  0.089495     0.253645
revol.util             -0.106824 -0.079071  0.430338     0.051048
inq.last.6mths         -0.537723  0.042744  0.177242    -0.023060
delinq.2yrs            -0.071108 -0.010893  0.149790    -0.006069
pub.rec                -0.064175  0.005841  0.109480    -0.027244
not.fully.paid         -0.196030  0.066310  0.215967     0.070602

                   log.annual.inc       dti      fico  days.with.cr.line  \
credit.policy            0.012198 -0.099022  0.377818           0.096084
purpose                  0.113946 -0.048576  0.069824           0.059256
int.rate                 0.079315  0.214052 -0.682865          -0.097441
installment              0.473751  0.018926  0.117298           0.180832
log.annual.inc           1.000000 -0.038153  0.106637           0.339438
dti                     -0.038153  1.000000 -0.229507           0.099784
```

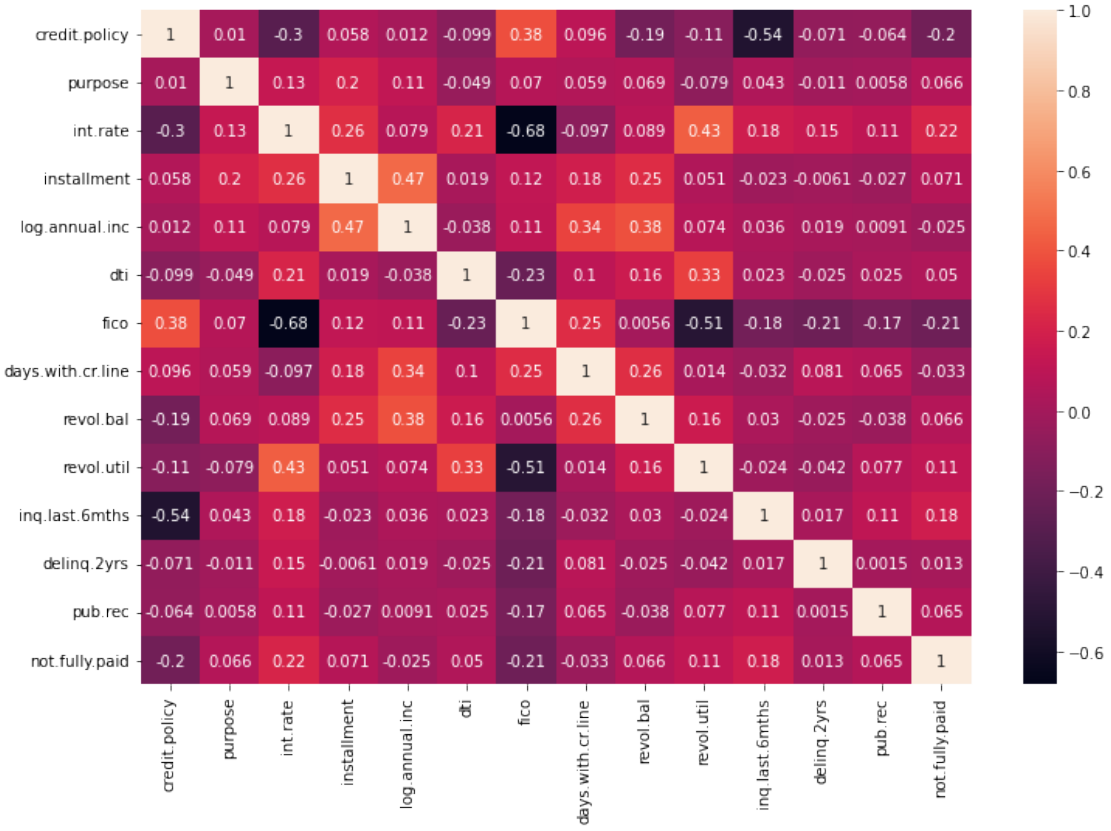
```
fico                    0.106637 -0.229507  1.000000           0.252452
days.with.cr.line       0.339438  0.099784  0.252452           1.000000
revol.bal               0.382583  0.162625  0.005634           0.257796
revol.util              0.074109  0.326800 -0.505498           0.014302
inq.last.6mths          0.035685  0.023381 -0.184308          -0.031732
delinq.2yrs             0.019154 -0.025076 -0.211182           0.081015
pub.rec                 0.009088  0.025374 -0.165606           0.064507
not.fully.paid         -0.025373  0.049897 -0.205492          -0.032590

                    revol.bal  revol.util  inq.last.6mths  delinq.2yrs  \
credit.policy      -0.187238   -0.106824       -0.537723    -0.071108
purpose             0.069319   -0.079071        0.042744    -0.010893
int.rate            0.089495    0.430338        0.177242     0.149790
installment         0.253645    0.051048       -0.023060    -0.006069
log.annual.inc      0.382583    0.074109        0.035685     0.019154
dti                 0.162625    0.326800        0.023381    -0.025076
fico                0.005634   -0.505498       -0.184308    -0.211182
days.with.cr.line   0.257796    0.014302       -0.031732     0.081015
revol.bal           1.000000    0.162692        0.029841    -0.024822
revol.util          0.162692    1.000000       -0.024078    -0.042334
inq.last.6mths      0.029841   -0.024078        1.000000     0.017139
delinq.2yrs        -0.024822   -0.042334        0.017139     1.000000
pub.rec            -0.037918    0.077428        0.107004     0.001464
not.fully.paid      0.065841    0.113406        0.176163     0.012830

                     pub.rec  not.fully.paid
credit.policy      -0.064175       -0.196030
purpose             0.005841        0.066310
int.rate            0.109480        0.215967
installment        -0.027244        0.070602
log.annual.inc      0.009088       -0.025373
dti                 0.025374        0.049897
fico               -0.165606       -0.205492
days.with.cr.line   0.064507       -0.032590
revol.bal          -0.037918        0.065841
revol.util          0.077428        0.113406
inq.last.6mths      0.107004        0.176163
delinq.2yrs         0.001464        0.012830
pub.rec             1.000000        0.064576
not.fully.paid      0.064576        1.000000
```

[53]:
```
plt.figure(figsize=[12,8])
sns.heatmap(new_df.corr(),annot=True)
```

[53]: <AxesSubplot:>



[55]:
```
# see the sorted results
new_df.corr().abs()['not.fully.paid'].sort_values(ascending=False)
```

[55]:
```
not.fully.paid        1.000000
int.rate              0.215967
fico                  0.205492
credit.policy         0.196030
inq.last.6mths        0.176163
revol.util            0.113406
installment           0.070602
purpose               0.066310
revol.bal             0.065841
pub.rec               0.064576
dti                   0.049897
days.with.cr.line     0.032590
log.annual.inc        0.025373
delinq.2yrs           0.012830
Name: not.fully.paid, dtype: float64
```

[56]:
```
new_df.columns
```

[56]:
```
Index(['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.rec', 'not.fully.paid'],
      dtype='object')
```

[57]:
```
# Consider the columns
X=new_df[['credit.policy','purpose', 'int.rate', 'installment','fico','revol.
 ↪bal','revol.util','inq.last.6mths','pub.rec']]
```

[58]:
```
X.shape
```

[58]:
```
(16090, 9)
```

[59]:
```
y=new_df['not.fully.paid']
```

[60]:
```
y.shape
```

[60]:
```
(16090,)
```

[61]:
```
# Create the train & test data
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.2,random_state=42)
```

[62]:
```
X_train.shape
```

[62]:
```
(12872, 9)
```

[63]:
```
X_test.shape
```

[63]:
```
(3218, 9)
```

[64]:
```
X_train
```

[64]:

	credit.policy	purpose	int.rate	installment	fico	revol.bal	\
5950	1	2	0.0894	406.68	817	306	
1743	1	6	0.1221	185.74	742	4	
6915	1	6	0.1531	174.08	702	30995	
513	1	0	0.1039	584.12	717	5506	
347	1	0	0.1141	16.47	667	12229	
…	…	…	…	…	…	…	
9566	0	0	0.2164	551.08	662	16441	
6742	1	2	0.1496	840.15	702	21111	
4183	1	4	0.0774	312.19	797	9109	
4115	1	2	0.1284	268.95	692	11798	
8310	0	4	0.1166	809.78	722	247970	

	revol.util	inq.last.6mths	pub.rec

```
5950         0.5                0         0
1743         0.0                1         1
6915        82.6                2         0
513         52.4                1         0
347         90.6                0         1
…            …                  …         …
9566        49.8                9         0
6742        75.7                3         0
4183        73.2                1         0
4115        88.7                1         0
8310        93.6               12         0

[12872 rows x 9 columns]
```

[66]:
```
# Apply scaling
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

[67]:
```
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

Create a deep learning model using Keras with Tensorflow backend

[72]:
```
#import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout
#from tensorflow.keras.callbacks import Earlystopping
```

[73]:
```
# Create the Architecture
# 2 ANN LAYER
model=Sequential()
model.add(Dense(16,activation='relu',input_shape=[9]))
model.add(Dropout(0.25))

model.add(Dense(10,activation='relu'))
model.add(Dropout(0.25))

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

[74]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                160
```

```
 dropout (Dropout)           (None, 16)                0

 dense_1 (Dense)             (None, 10)                170

 dropout_1 (Dropout)         (None, 10)                0

 dense_2 (Dense)             (None, 1)                 11

=================================================================
Total params: 341
Trainable params: 341
Non-trainable params: 0
_________________________________________________________________
```

[75]:
```
#Compile the model
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

[76]:
```
model.
  ↪fit(X_train,y_train,epochs=50,batch_size=256,validation_data=(X_test,y_test))
```

```
Epoch 1/50
51/51 [==============================] - 7s 8ms/step - loss: 0.7552 - accuracy:
0.4894 - val_loss: 0.6801 - val_accuracy: 0.5724
Epoch 2/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6919 - accuracy:
0.5544 - val_loss: 0.6610 - val_accuracy: 0.5923
Epoch 3/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6766 - accuracy:
0.5667 - val_loss: 0.6560 - val_accuracy: 0.5985
Epoch 4/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6736 - accuracy:
0.5755 - val_loss: 0.6534 - val_accuracy: 0.6013
Epoch 5/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6671 - accuracy:
0.5809 - val_loss: 0.6514 - val_accuracy: 0.6047
Epoch 6/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6660 - accuracy:
0.5860 - val_loss: 0.6501 - val_accuracy: 0.6050
Epoch 7/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6622 - accuracy:
0.5879 - val_loss: 0.6493 - val_accuracy: 0.6022
Epoch 8/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6589 - accuracy:
0.5927 - val_loss: 0.6480 - val_accuracy: 0.6069
Epoch 9/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6584 - accuracy:
0.5968 - val_loss: 0.6469 - val_accuracy: 0.6078
Epoch 10/50
```

```
51/51 [==============================] - 0s 3ms/step - loss: 0.6584 - accuracy:
0.6011 - val_loss: 0.6462 - val_accuracy: 0.6097
Epoch 11/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6578 - accuracy:
0.5970 - val_loss: 0.6457 - val_accuracy: 0.6137
Epoch 12/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6577 - accuracy:
0.5961 - val_loss: 0.6456 - val_accuracy: 0.6150
Epoch 13/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6542 - accuracy:
0.6047 - val_loss: 0.6442 - val_accuracy: 0.6184
Epoch 14/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6590 - accuracy:
0.5983 - val_loss: 0.6455 - val_accuracy: 0.6168
Epoch 15/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6539 - accuracy:
0.6024 - val_loss: 0.6437 - val_accuracy: 0.6153
Epoch 16/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6509 - accuracy:
0.6085 - val_loss: 0.6428 - val_accuracy: 0.6184
Epoch 17/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6539 - accuracy:
0.6098 - val_loss: 0.6425 - val_accuracy: 0.6200
Epoch 18/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6535 - accuracy:
0.6048 - val_loss: 0.6422 - val_accuracy: 0.6184
Epoch 19/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6539 - accuracy:
0.6085 - val_loss: 0.6419 - val_accuracy: 0.6221
Epoch 20/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6538 - accuracy:
0.6036 - val_loss: 0.6417 - val_accuracy: 0.6218
Epoch 21/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6521 - accuracy:
0.6082 - val_loss: 0.6420 - val_accuracy: 0.6215
Epoch 22/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6509 - accuracy:
0.6057 - val_loss: 0.6409 - val_accuracy: 0.6190
Epoch 23/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6527 - accuracy:
0.6064 - val_loss: 0.6413 - val_accuracy: 0.6196
Epoch 24/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6507 - accuracy:
0.6063 - val_loss: 0.6415 - val_accuracy: 0.6168
Epoch 25/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6514 - accuracy:
0.6071 - val_loss: 0.6414 - val_accuracy: 0.6162
Epoch 26/50
```

```
51/51 [==============================] - 0s 3ms/step - loss: 0.6505 - accuracy:
0.6081 - val_loss: 0.6413 - val_accuracy: 0.6190
Epoch 27/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6515 - accuracy:
0.6055 - val_loss: 0.6408 - val_accuracy: 0.6187
Epoch 28/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6498 - accuracy:
0.6057 - val_loss: 0.6405 - val_accuracy: 0.6221
Epoch 29/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6493 - accuracy:
0.6082 - val_loss: 0.6402 - val_accuracy: 0.6237
Epoch 30/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6496 - accuracy:
0.6130 - val_loss: 0.6407 - val_accuracy: 0.6196
Epoch 31/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6492 - accuracy:
0.6116 - val_loss: 0.6404 - val_accuracy: 0.6234
Epoch 32/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6474 - accuracy:
0.6130 - val_loss: 0.6399 - val_accuracy: 0.6249
Epoch 33/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6501 - accuracy:
0.6064 - val_loss: 0.6400 - val_accuracy: 0.6262
Epoch 34/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6502 - accuracy:
0.6075 - val_loss: 0.6393 - val_accuracy: 0.6243
Epoch 35/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6496 - accuracy:
0.6107 - val_loss: 0.6404 - val_accuracy: 0.6243
Epoch 36/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6493 - accuracy:
0.6103 - val_loss: 0.6401 - val_accuracy: 0.6262
Epoch 37/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6492 - accuracy:
0.6106 - val_loss: 0.6403 - val_accuracy: 0.6255
Epoch 38/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6500 - accuracy:
0.6071 - val_loss: 0.6401 - val_accuracy: 0.6209
Epoch 39/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6484 - accuracy:
0.6099 - val_loss: 0.6399 - val_accuracy: 0.6231
Epoch 40/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6484 - accuracy:
0.6118 - val_loss: 0.6389 - val_accuracy: 0.6249
Epoch 41/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6460 - accuracy:
0.6105 - val_loss: 0.6389 - val_accuracy: 0.6227
Epoch 42/50
```

```
51/51 [==============================] - 0s 3ms/step - loss: 0.6483 - accuracy:
0.6078 - val_loss: 0.6387 - val_accuracy: 0.6259
Epoch 43/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6485 - accuracy:
0.6088 - val_loss: 0.6388 - val_accuracy: 0.6200
Epoch 44/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6463 - accuracy:
0.6121 - val_loss: 0.6387 - val_accuracy: 0.6221
Epoch 45/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6470 - accuracy:
0.6139 - val_loss: 0.6383 - val_accuracy: 0.6240
Epoch 46/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6478 - accuracy:
0.6146 - val_loss: 0.6387 - val_accuracy: 0.6287
Epoch 47/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6481 - accuracy:
0.6140 - val_loss: 0.6391 - val_accuracy: 0.6302
Epoch 48/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6483 - accuracy:
0.6137 - val_loss: 0.6383 - val_accuracy: 0.6283
Epoch 49/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6454 - accuracy:
0.6166 - val_loss: 0.6375 - val_accuracy: 0.6277
Epoch 50/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6468 - accuracy:
0.6123 - val_loss: 0.6379 - val_accuracy: 0.6321
```

[76]:
```
<keras.callbacks.History at 0x7f6e90380f10>
```

[77]:
```
history=model.
  ↪fit(X_train,y_train,epochs=50,batch_size=256,validation_data=(X_test,y_test))
```

```
Epoch 1/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6465 - accuracy:
0.6154 - val_loss: 0.6382 - val_accuracy: 0.6311
Epoch 2/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6431 - accuracy:
0.6194 - val_loss: 0.6377 - val_accuracy: 0.6302
Epoch 3/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6450 - accuracy:
0.6124 - val_loss: 0.6373 - val_accuracy: 0.6299
Epoch 4/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6455 - accuracy:
0.6170 - val_loss: 0.6374 - val_accuracy: 0.6305
Epoch 5/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6461 - accuracy:
0.6179 - val_loss: 0.6371 - val_accuracy: 0.6293
Epoch 6/50
```

```
51/51 [==============================] - 0s 2ms/step - loss: 0.6446 - accuracy:
0.6191 - val_loss: 0.6374 - val_accuracy: 0.6290
Epoch 7/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6470 - accuracy:
0.6140 - val_loss: 0.6380 - val_accuracy: 0.6327
Epoch 8/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6461 - accuracy:
0.6222 - val_loss: 0.6375 - val_accuracy: 0.6346
Epoch 9/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6437 - accuracy:
0.6208 - val_loss: 0.6368 - val_accuracy: 0.6290
Epoch 10/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6465 - accuracy:
0.6213 - val_loss: 0.6374 - val_accuracy: 0.6274
Epoch 11/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6469 - accuracy:
0.6150 - val_loss: 0.6376 - val_accuracy: 0.6290
Epoch 12/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6444 - accuracy:
0.6189 - val_loss: 0.6370 - val_accuracy: 0.6311
Epoch 13/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6427 - accuracy:
0.6210 - val_loss: 0.6370 - val_accuracy: 0.6305
Epoch 14/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6459 - accuracy:
0.6184 - val_loss: 0.6373 - val_accuracy: 0.6308
Epoch 15/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6435 - accuracy:
0.6183 - val_loss: 0.6367 - val_accuracy: 0.6314
Epoch 16/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6451 - accuracy:
0.6189 - val_loss: 0.6375 - val_accuracy: 0.6324
Epoch 17/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6423 - accuracy:
0.6227 - val_loss: 0.6369 - val_accuracy: 0.6311
Epoch 18/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6430 - accuracy:
0.6225 - val_loss: 0.6364 - val_accuracy: 0.6308
Epoch 19/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6470 - accuracy:
0.6222 - val_loss: 0.6373 - val_accuracy: 0.6318
Epoch 20/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6444 - accuracy:
0.6184 - val_loss: 0.6371 - val_accuracy: 0.6296
Epoch 21/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6450 - accuracy:
0.6201 - val_loss: 0.6370 - val_accuracy: 0.6302
Epoch 22/50
```

```
51/51 [==============================] - 0s 2ms/step - loss: 0.6434 - accuracy:
0.6227 - val_loss: 0.6367 - val_accuracy: 0.6305
Epoch 23/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6449 - accuracy:
0.6202 - val_loss: 0.6371 - val_accuracy: 0.6305
Epoch 24/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6436 - accuracy:
0.6213 - val_loss: 0.6370 - val_accuracy: 0.6287
Epoch 25/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6436 - accuracy:
0.6240 - val_loss: 0.6368 - val_accuracy: 0.6305
Epoch 26/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6424 - accuracy:
0.6190 - val_loss: 0.6359 - val_accuracy: 0.6314
Epoch 27/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6435 - accuracy:
0.6220 - val_loss: 0.6365 - val_accuracy: 0.6305
Epoch 28/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6432 - accuracy:
0.6216 - val_loss: 0.6357 - val_accuracy: 0.6324
Epoch 29/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6444 - accuracy:
0.6224 - val_loss: 0.6365 - val_accuracy: 0.6305
Epoch 30/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6425 - accuracy:
0.6252 - val_loss: 0.6361 - val_accuracy: 0.6314
Epoch 31/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6404 - accuracy:
0.6258 - val_loss: 0.6358 - val_accuracy: 0.6287
Epoch 32/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6441 - accuracy:
0.6199 - val_loss: 0.6370 - val_accuracy: 0.6318
Epoch 33/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6436 - accuracy:
0.6219 - val_loss: 0.6365 - val_accuracy: 0.6311
Epoch 34/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6458 - accuracy:
0.6200 - val_loss: 0.6366 - val_accuracy: 0.6308
Epoch 35/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6422 - accuracy:
0.6241 - val_loss: 0.6361 - val_accuracy: 0.6299
Epoch 36/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6440 - accuracy:
0.6256 - val_loss: 0.6365 - val_accuracy: 0.6296
Epoch 37/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6424 - accuracy:
0.6193 - val_loss: 0.6360 - val_accuracy: 0.6290
Epoch 38/50
```

```
51/51 [==============================] - 0s 2ms/step - loss: 0.6428 - accuracy:
0.6199 - val_loss: 0.6362 - val_accuracy: 0.6280
Epoch 39/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6421 - accuracy:
0.6210 - val_loss: 0.6359 - val_accuracy: 0.6296
Epoch 40/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6421 - accuracy:
0.6230 - val_loss: 0.6361 - val_accuracy: 0.6308
Epoch 41/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6434 - accuracy:
0.6193 - val_loss: 0.6365 - val_accuracy: 0.6302
Epoch 42/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6422 - accuracy:
0.6231 - val_loss: 0.6363 - val_accuracy: 0.6308
Epoch 43/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6412 - accuracy:
0.6197 - val_loss: 0.6362 - val_accuracy: 0.6290
Epoch 44/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6417 - accuracy:
0.6213 - val_loss: 0.6359 - val_accuracy: 0.6280
Epoch 45/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6417 - accuracy:
0.6232 - val_loss: 0.6360 - val_accuracy: 0.6293
Epoch 46/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6420 - accuracy:
0.6254 - val_loss: 0.6363 - val_accuracy: 0.6302
Epoch 47/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6418 - accuracy:
0.6227 - val_loss: 0.6357 - val_accuracy: 0.6299
Epoch 48/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6430 - accuracy:
0.6239 - val_loss: 0.6364 - val_accuracy: 0.6327
Epoch 49/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6424 - accuracy:
0.6212 - val_loss: 0.6361 - val_accuracy: 0.6311
Epoch 50/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6379 - accuracy:
0.6287 - val_loss: 0.6352 - val_accuracy: 0.6330
```

[78]:
```
model.evaluate(X_test,y_test)
```

```
101/101 [==============================] - 0s 1ms/step - loss: 0.6352 -
accuracy: 0.6330
```

[78]:
```
[0.6352449059486389, 0.6330018639564514]
```

[79]:
```
model.evaluate(X_train,y_train)
```

```
403/403 [==============================] - 0s 960us/step - loss: 0.6328 -
accuracy: 0.6314
```

[79]:
```
[0.6328166723251343, 0.6313704252243042]
```

[80]:
```
y_pred=model.predict(X_test)
```

[81]:
```
y_pred
```

[81]:
```
array([[0.6831651 ],
       [0.47518575],
       [0.6351444 ],
       …,
       [0.4598096 ],
       [0.55074376],
       [0.48613152]], dtype=float32)
```

[82]:
```
predictions=(y_pred>0.5).astype('int')
```

[83]:
```
predictions
```

[83]:
```
array([[1],
       [0],
       [1],
       …,
       [0],
       [1],
       [0]])
```

[84]:
```
y_test
```

[84]:
```
4030    1
4020    1
8647    1
481     1
6841    1
       ..
1637    1
1427    0
922     0
4898    0
2620    1
Name: not.fully.paid, Length: 3218, dtype: int64
```

[85]:
```
from sklearn.metrics import␣
 ↪accuracy_score,confusion_matrix,classification_report
accuracy_score(predictions,y_test)
```

[85]: 0.6330018645121194

[86]:
```
print(classification_report(predictions,y_test))
```

```
              precision    recall  f1-score   support

           0       0.68      0.62      0.65      1775
           1       0.58      0.65      0.61      1443

    accuracy                           0.63      3218
   macro avg       0.63      0.63      0.63      3218
weighted avg       0.64      0.63      0.63      3218
```

[87]:
```
model.save('DL_PROJECT1.h5')
```

[ ]: