DL_Project_LendingClub_Loan_data_analysis

November 5, 2023

[1]:
```python
# import library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```python
# load the dataset
df=pd.read_csv('loan_data.csv')
```

[5]:
```python
df.head()
```

[5]:
```
   credit.policy             purpose  int.rate  installment  log.annual.inc  \
0              1  debt_consolidation    0.1189       829.10       11.350407
1              1         credit_card    0.1071       228.22       11.082143
2              1  debt_consolidation    0.1357       366.86       10.373491
3              1  debt_consolidation    0.1008       162.34       11.350407
4              1         credit_card    0.1426       102.92       11.299732

     dti  fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
0  19.48   737        5639.958333      28854        52.1               0
1  14.29   707        2760.000000      33623        76.7               0
2  11.63   682        4710.000000       3511        25.6               1
3   8.10   712        2699.958333      33667        73.2               1
4  14.97   667        4066.000000       4740        39.5               0

   delinq.2yrs  pub.rec  not.fully.paid
0            0        0               0
1            0        0               0
2            0        0               0
3            0        0               0
4            1        0               0
```

[6]:
```python
df.shape
```

[6]:
```
(9578, 14)
```

[12]:
```python
df.describe().transpose()
```

[12]:

```
                    count          mean           std         min  \
credit.policy      9578.0      0.804970      0.396245    0.000000
int.rate           9578.0      0.122640      0.026847    0.060000
installment        9578.0    319.089413    207.071301   15.670000
log.annual.inc     9578.0     10.932117      0.614813    7.547502
dti                9578.0     12.606679      6.883970    0.000000
fico               9578.0    710.846314     37.970537  612.000000
days.with.cr.line  9578.0   4560.767197   2496.930377  178.958333
revol.bal          9578.0  16913.963876  33756.189557    0.000000
revol.util         9578.0     46.799236     29.014417    0.000000
inq.last.6mths     9578.0      1.577469      2.200245    0.000000
delinq.2yrs        9578.0      0.163708      0.546215    0.000000
pub.rec            9578.0      0.062122      0.262126    0.000000
not.fully.paid     9578.0      0.160054      0.366676    0.000000

                           25%          50%           75%           max
credit.policy         1.000000     1.000000      1.000000  1.000000e+00
int.rate              0.103900     0.122100      0.140700  2.164000e-01
installment         163.770000   268.950000    432.762500  9.401400e+02
log.annual.inc       10.558414    10.928884     11.291293  1.452835e+01
dti                   7.212500    12.665000     17.950000  2.996000e+01
fico                682.000000   707.000000    737.000000  8.270000e+02
days.with.cr.line  2820.000000  4139.958333   5730.000000  1.763996e+04
revol.bal          3187.000000  8596.000000  18249.500000  1.207359e+06
revol.util           22.600000    46.300000     70.900000  1.190000e+02
inq.last.6mths        0.000000     1.000000      2.000000  3.300000e+01
delinq.2yrs           0.000000     0.000000      0.000000  1.300000e+01
pub.rec               0.000000     0.000000      0.000000  5.000000e+00
not.fully.paid        0.000000     0.000000      0.000000  1.000000e+00
```

[13]:

```
# missing value
df.isnull().sum()
```

[13]:

```
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 null value found. Ready to proceed

[14]:
```
df['not.fully.paid'].value_counts()
# 0- full paid, 1 - not paid
# imbalanced data
```

[14]:
```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

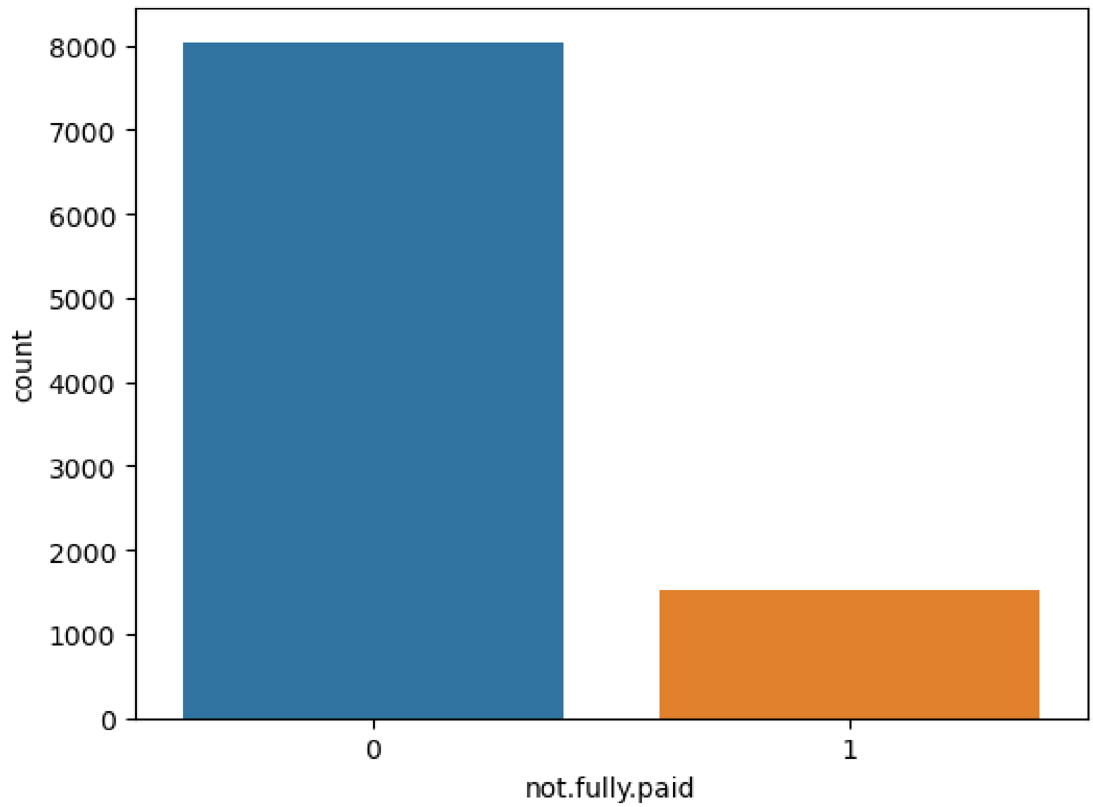
In the give dataset the target variable of 0 class contains more data than 1 class. 0 class has around 8045 data and 1 class has around 1533 data. So a imbalance of data in noticed here as per my analysis.

0.1 EDA of different factors of the dataset.

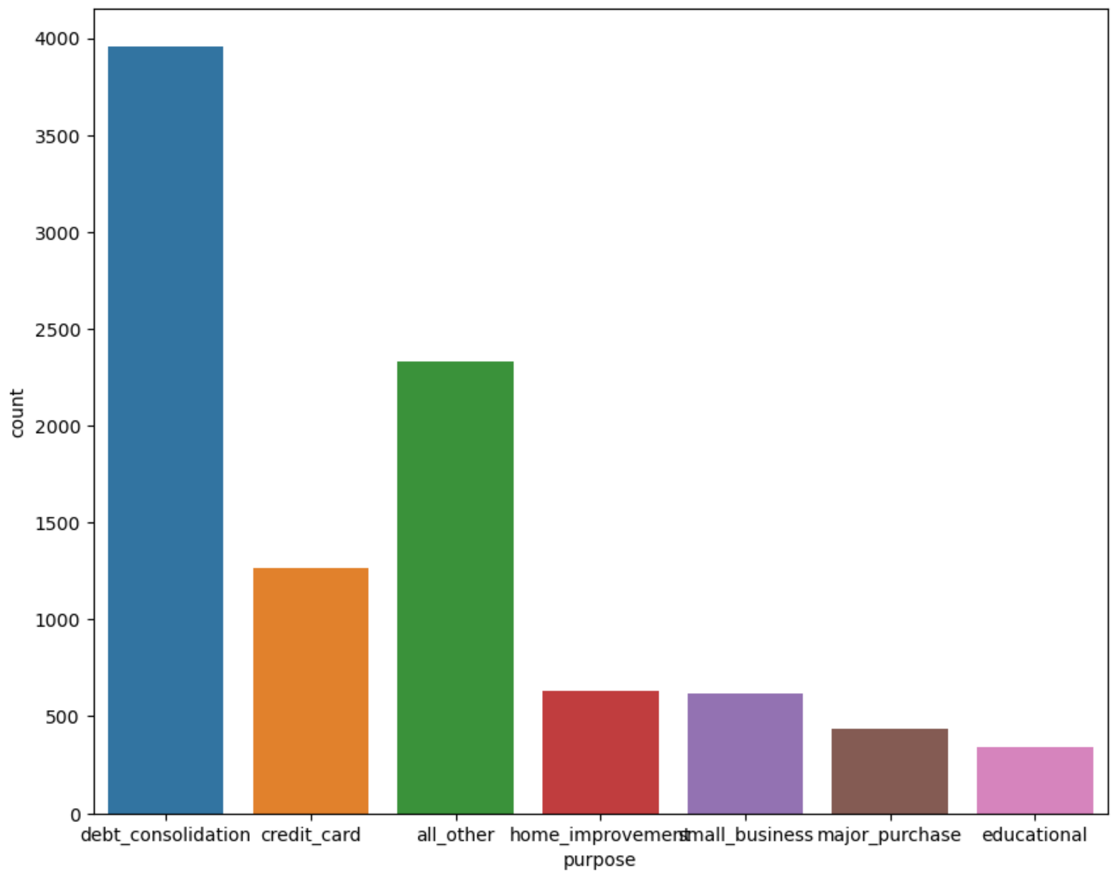
[15]:
```
df.dtypes
```

[15]:
```
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
```

[76]:
```
sns.countplot(x=df['not.fully.paid'])
plt.savefig('countplot.png')
plt.show()
```

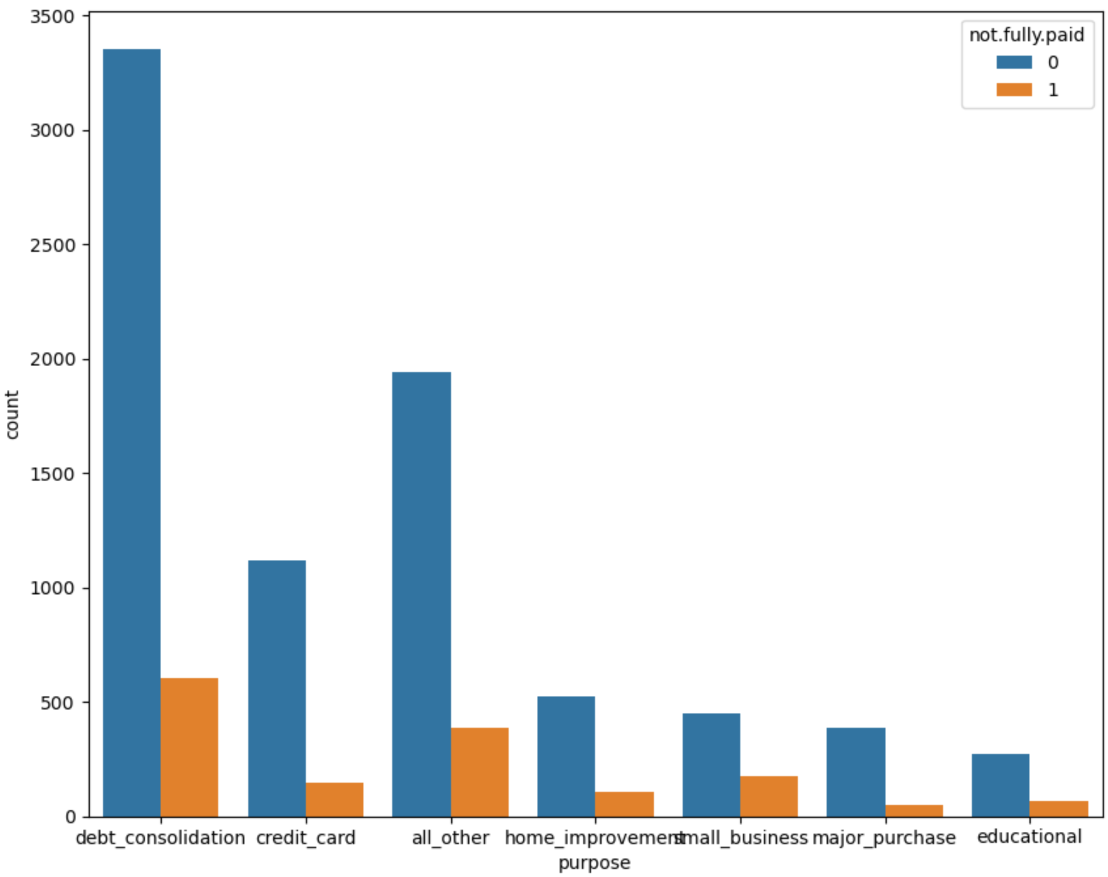


[77]:
```
plt.figure(figsize=(10,8))
sns.countplot(x=df['purpose'])
plt.savefig('countplot2.png')
plt.show()
```



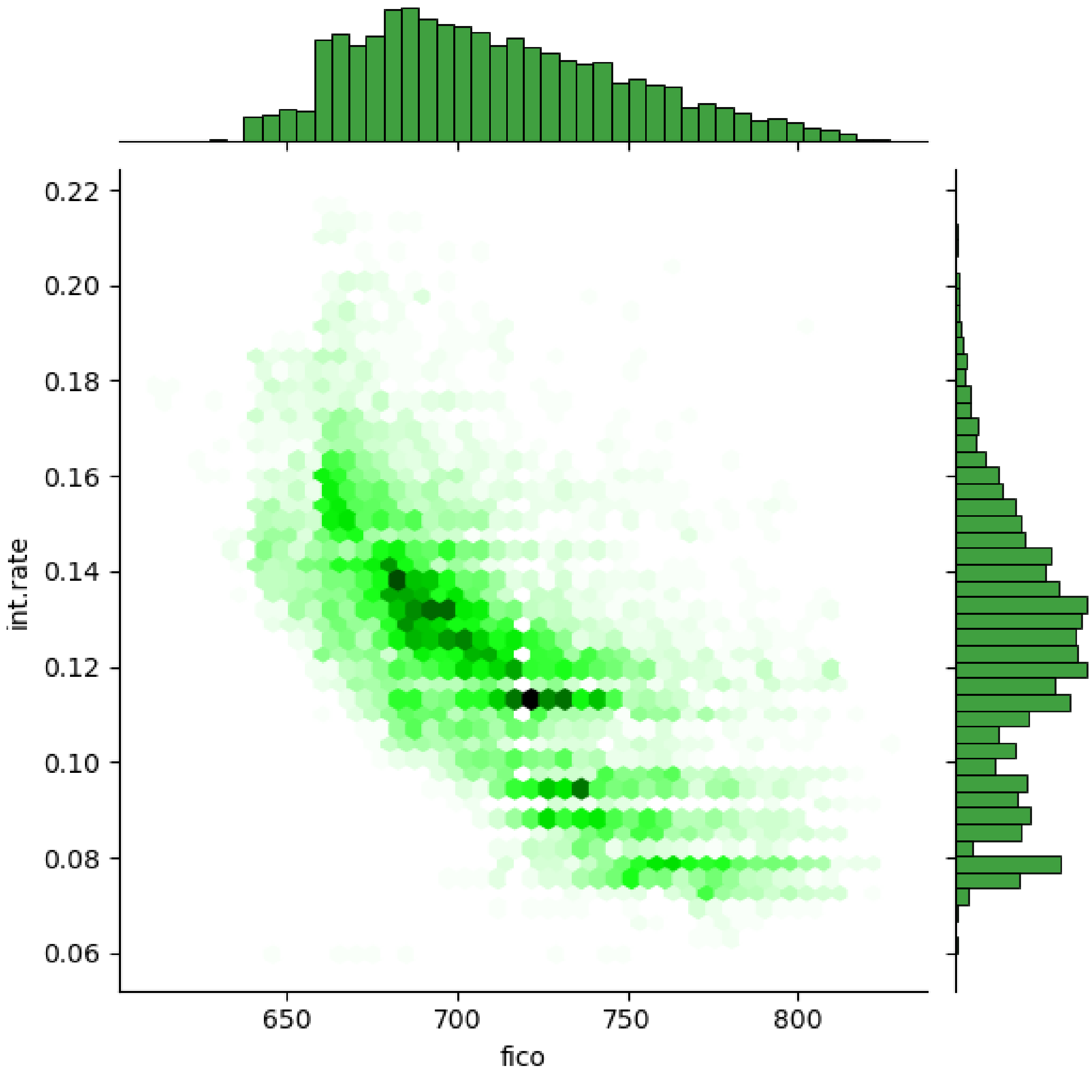
As per the observation from above chart debt_consolidation has the highest number of data occurance with reference to the purpose column.

[78]:
```
# purpose ---- not fully paid
plt.figure(figsize=(10,8))
sns.countplot(x='purpose',hue='not.fully.paid',data=df)
plt.savefig('countplot2.png')
plt.show()
```

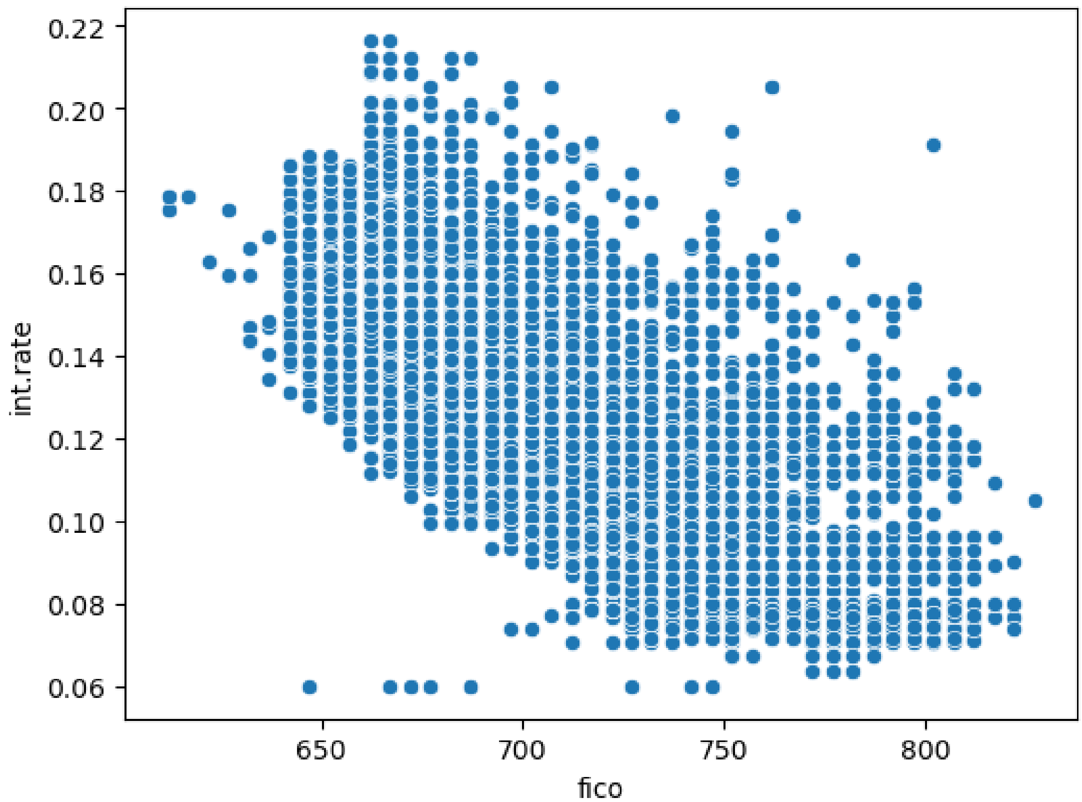


As per above chart, from the debt_consolidation data not_full_paid=0 has the highest number of occurance.

[79]:
```
# bi variate analysis
sns.jointplot(x='fico',y='int.rate',data=df,kind='hex',color='g')
plt.savefig('Jointplot.png')
plt.show()
```

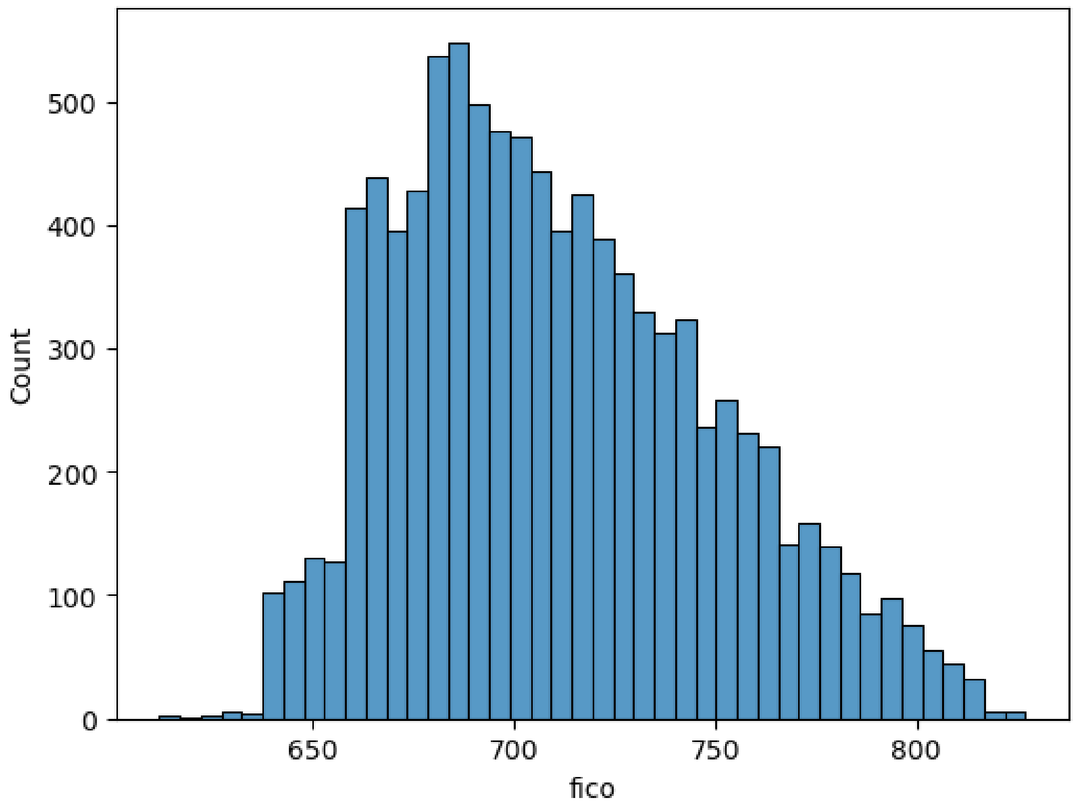


[80]:
```
sns.scatterplot(x='fico',y='int.rate',data=df)
plt.savefig('Scatterplot.png')
plt.show()
```

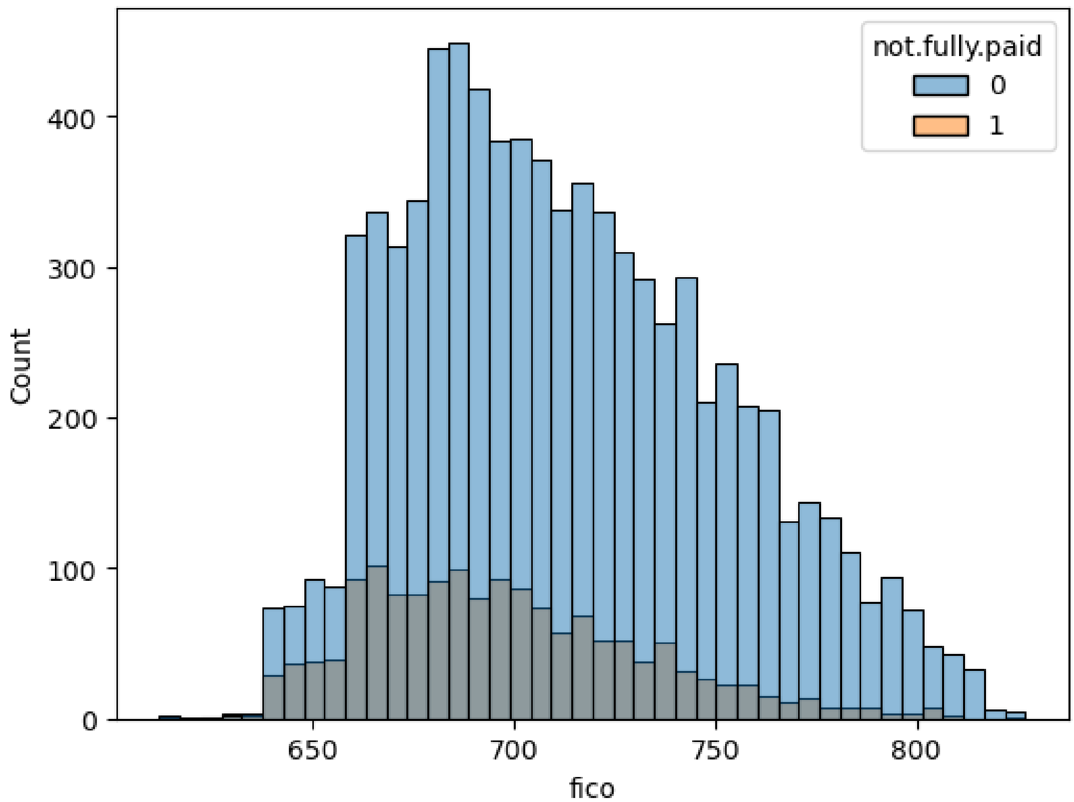


[81]:

```
sns.histplot(df['fico'])
plt.savefig('histplot.png')
plt.show()
```



[82]:
```
sns.histplot(x='fico',hue='not.fully.paid',data=df)
plt.savefig('histplot1.png')
plt.show()
```



Feature Transformation

Transform categorical values into numerical values (discrete)

[23]:
```
# handle imbalanced dataset
df['not.fully.paid'].value_counts()
```

[23]:
```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

[24]:
```
not_fully_paid_0=df[df['not.fully.paid']==0]
not_fully_paid_1=df[df['not.fully.paid']==1]
```

[25]:
```
not_fully_paid_0.shape
```

[25]:
```
(8045, 14)
```

[26]:
```
not_fully_paid_1.shape
```

[26]:
```
(1533, 14)
```

[27]:
```
# resample
from sklearn.utils import resample
df_minor_upsample=resample(not_fully_paid_1,replace=True,n_samples=8045)
```

[28]:
```
new_df=pd.concat([not_fully_paid_0,df_minor_upsample])
```

[29]:
```
# shuffle
from sklearn.utils import shuffle
new_df=shuffle(new_df)
```

[30]:
```
new_df['not.fully.paid'].value_counts()
```

[30]:
```
0    8045
1    8045
Name: not.fully.paid, dtype: int64
```

[31]:
```
new_df.shape
```

[31]:
```
(16090, 14)
```

[32]:
```
new_df.dtypes
```

[32]:
```
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
```

0.1.1 Using LabelEncoder to perform Feature Transformation by converting the categorical values into numerical values.

[33]:
```
# convert purpose into num data
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

[34]:
```
for i in new_df.columns:
    if new_df[i].dtypes=='object':
        new_df[i]=le.fit_transform(new_df[i])
```

[35]:
```
new_df.dtypes
```

[35]:
```
credit.policy          int64
purpose                int32
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
```

0.1.2 Additional Feature Engineering

0.1.3 Need to check the correlation between all features & will drop those features which have a strong correlation

0.1.4 This will help to reduce the number of features & will leave with the most relevant features.

[36]:
```
new_df.corr().transpose()
```

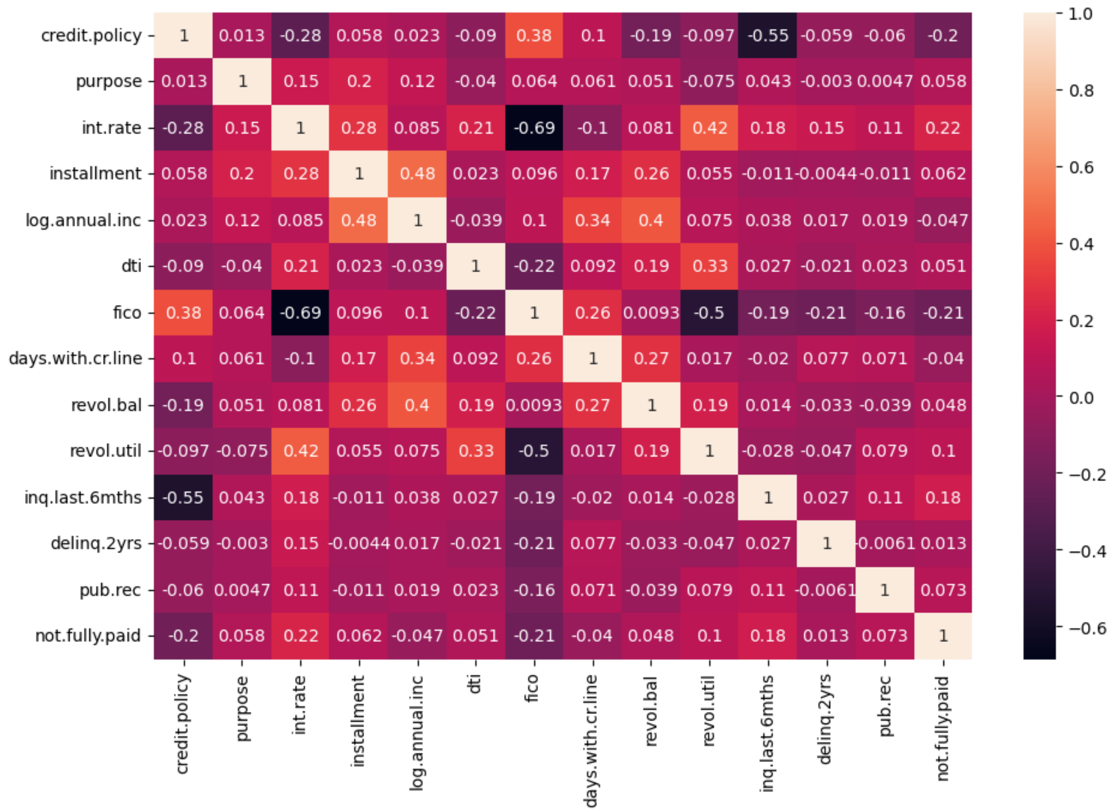
[36]:
```
                   credit.policy   purpose  int.rate  installment  \
credit.policy           1.000000  0.012819 -0.281133     0.057643
purpose                 0.012819  1.000000  0.153037     0.202618
int.rate               -0.281133  0.153037  1.000000     0.276712
installment             0.057643  0.202618  0.276712     1.000000
log.annual.inc          0.022957  0.118770  0.084610     0.477661
dti                    -0.089764 -0.040141  0.206080     0.023006
fico                    0.376964  0.063894 -0.685246     0.095973
days.with.cr.line       0.104494  0.060941 -0.104488     0.174049
revol.bal              -0.186621  0.050805  0.080539     0.257980
revol.util             -0.097121 -0.074617  0.423236     0.055020
inq.last.6mths         -0.548261  0.043370  0.175648    -0.010936
delinq.2yrs            -0.058617 -0.003005  0.148065    -0.004405
pub.rec                -0.059581  0.004688  0.110993    -0.011076
not.fully.paid         -0.198948  0.058403  0.215047     0.062114
```

```
                   log.annual.inc       dti      fico  days.with.cr.line  \
credit.policy            0.022957 -0.089764  0.376964           0.104494
purpose                  0.118770 -0.040141  0.063894           0.060941
int.rate                 0.084610  0.206080 -0.685246          -0.104488
installment              0.477661  0.023006  0.095973           0.174049
log.annual.inc           1.000000 -0.038630  0.099582           0.337939
dti                     -0.038630  1.000000 -0.224644           0.091962
fico                     0.099582 -0.224644  1.000000           0.258221
days.with.cr.line        0.337939  0.091962  0.258221           1.000000
revol.bal                0.403723  0.189329  0.009334           0.269405
revol.util               0.075415  0.327599 -0.501657           0.016797
inq.last.6mths           0.037921  0.026996 -0.188239          -0.019586
delinq.2yrs              0.017088 -0.020804 -0.207212           0.077249
pub.rec                  0.019092  0.022769 -0.160963           0.071423
not.fully.paid          -0.047235  0.050721 -0.206863          -0.040461

                   revol.bal  revol.util  inq.last.6mths  delinq.2yrs  \
credit.policy      -0.186621   -0.097121       -0.548261    -0.058617
purpose             0.050805   -0.074617        0.043370    -0.003005
int.rate            0.080539    0.423236        0.175648     0.148065
installment         0.257980    0.055020       -0.010936    -0.004405
log.annual.inc      0.403723    0.075415        0.037921     0.017088
dti                 0.189329    0.327599        0.026996    -0.020804
fico                0.009334   -0.501657       -0.188239    -0.207212
days.with.cr.line   0.269405    0.016797       -0.019586     0.077249
revol.bal           1.000000    0.189338        0.014073    -0.032943
revol.util          0.189338    1.000000       -0.027680    -0.046589
inq.last.6mths      0.014073   -0.027680        1.000000     0.026838
delinq.2yrs        -0.032943   -0.046589        0.026838     1.000000
pub.rec            -0.039172    0.079339        0.105057    -0.006084
not.fully.paid      0.048107    0.101615        0.175383     0.012862

                    pub.rec  not.fully.paid
credit.policy     -0.059581       -0.198948
purpose            0.004688        0.058403
int.rate           0.110993        0.215047
installment       -0.011076        0.062114
log.annual.inc     0.019092       -0.047235
dti                0.022769        0.050721
fico              -0.160963       -0.206863
days.with.cr.line  0.071423       -0.040461
revol.bal         -0.039172        0.048107
revol.util         0.079339        0.101615
inq.last.6mths     0.105057        0.175383
delinq.2yrs       -0.006084        0.012862
pub.rec            1.000000        0.073023
not.fully.paid     0.073023        1.000000
```

```
[83]: plt.figure(figsize=(11,7))
      sns.heatmap(new_df.corr(),annot=True)
      plt.savefig('heatmap.png')
      plt.show()
```




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

```
[38]: not.fully.paid       1.000000
      int.rate             0.215047
      fico                 0.206863
      credit.policy        0.198948
      inq.last.6mths       0.175383
      revol.util           0.101615
      pub.rec              0.073023
      installment          0.062114
      purpose              0.058403
      dti                  0.050721
      revol.bal            0.048107
      log.annual.inc       0.047235
      days.with.cr.line    0.040461
```

```
delinq.2yrs          0.012862
Name: not.fully.paid, dtype: float64
```

[39]:
```
new_df.columns
```

[39]:
```
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')
```

0.1.5 We are dropping few columns with highest corelation and keeping only limited columns

[42]:
```
# take columns
X=new_df[['credit.policy','purpose', 'int.rate', 'installment','fico','revol.
↪bal','revol.util','inq.last.6mths','pub.rec']]
```

[43]:
```
X.shape
```

[43]:
```
(16090, 9)
```

[44]:
```
y=new_df['not.fully.paid']
```

[45]:
```
# Create train set & test set
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)
```

[46]:
```
X_train.shape
```

[46]:
```
(12872, 9)
```

[47]:
```
X_test.shape
```

[47]:
```
(3218, 9)
```

[48]:
```
X_train
```

[48]:

	credit.policy	purpose	int.rate	installment	fico	revol.bal	\
1577	1	1	0.0907	458.39	792	4658	
1459	1	5	0.1148	725.31	762	6628	
883	1	1	0.1450	640.24	682	32682	
7255	1	0	0.1422	548.55	707	33113	
2939	1	2	0.1600	285.66	702	21438	
…	…	…	…	…	…	…	
6202	1	6	0.2121	755.69	672	500	
7318	1	5	0.1422	368.56	677	7634	
8580	0	2	0.1134	119.27	687	82141	

```
2178              1        2    0.1411       219.07   692        8959
5781              1        0    0.1322       135.21   702        1291

      revol.util  inq.last.6mths  pub.rec
1577        10.7               0        0
1459         7.3               1        0
883         96.1               0        0
7255        97.4               2        0
2939        90.6               1        1
…           …              …      …
6202        83.3               1        0
7318        50.9               0        0
8580        91.8               3        0
2178        90.5               0        0
5781        47.8               0        0

[12872 rows x 9 columns]
```

[49]:
```
# Apply scaling
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

[50]:
```
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

0.1.6 Create a deep learning model using Keras with Tensorflow backend

[51]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

[52]:
```
# create the architecture
# 2 ANN layer
model=Sequential()
model.add(Dense(19,activation='relu',input_shape=[9]))
model.add(Dropout(0.20))


model.add(Dense(10,activation='relu'))
model.add(Dropout(0.20))

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

[53]:
```
model.summary()
```

```
Model: "sequential"
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 19)                190

 dropout (Dropout)           (None, 19)                0

 dense_1 (Dense)             (None, 10)                200

 dropout_1 (Dropout)         (None, 10)                0

 dense_2 (Dense)             (None, 1)                 11

=================================================================
Total params: 401 (1.57 KB)
Trainable params: 401 (1.57 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[54]:
```
# compile the model
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

[55]:
```
early_stop=EarlyStopping(monitor='val_loss',min_delta=0.
↪001,mode='min',patience=10,verbose=1)
```

[58]:
```
model.fit(X_train,y_train,
          epochs=50,
          batch_size=256,
          validation_data=(X_test,y_test),
          callbacks=[early_stop])
```

```
Epoch 1/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6404 - accuracy:
0.6302 - val_loss: 0.6480 - val_accuracy: 0.6168
Epoch 2/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6420 - accuracy:
0.6273 - val_loss: 0.6479 - val_accuracy: 0.6190
Epoch 3/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6399 - accuracy:
0.6281 - val_loss: 0.6481 - val_accuracy: 0.6200
Epoch 4/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6423 - accuracy:
0.6265 - val_loss: 0.6480 - val_accuracy: 0.6215
Epoch 5/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6419 - accuracy:
0.6283 - val_loss: 0.6478 - val_accuracy: 0.6215
Epoch 6/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6421 - accuracy:
```

```
0.6260 - val_loss: 0.6480 - val_accuracy: 0.6165
Epoch 7/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6405 - accuracy:
0.6284 - val_loss: 0.6479 - val_accuracy: 0.6172
Epoch 8/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6380 - accuracy:
0.6315 - val_loss: 0.6473 - val_accuracy: 0.6193
Epoch 9/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6433 - accuracy:
0.6293 - val_loss: 0.6473 - val_accuracy: 0.6196
Epoch 10/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6400 - accuracy:
0.6280 - val_loss: 0.6473 - val_accuracy: 0.6212
Epoch 11/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6400 - accuracy:
0.6312 - val_loss: 0.6470 - val_accuracy: 0.6240
Epoch 12/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6400 - accuracy:
0.6280 - val_loss: 0.6468 - val_accuracy: 0.6287
Epoch 13/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6417 - accuracy:
0.6269 - val_loss: 0.6471 - val_accuracy: 0.6240
Epoch 14/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6392 - accuracy:
0.6259 - val_loss: 0.6470 - val_accuracy: 0.6255
Epoch 15/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6382 - accuracy:
0.6298 - val_loss: 0.6468 - val_accuracy: 0.6237
Epoch 16/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6407 - accuracy:
0.6255 - val_loss: 0.6466 - val_accuracy: 0.6255
Epoch 17/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6383 - accuracy:
0.6300 - val_loss: 0.6466 - val_accuracy: 0.6237
Epoch 18/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6400 - accuracy:
0.6310 - val_loss: 0.6461 - val_accuracy: 0.6249
Epoch 19/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6391 - accuracy:
0.6293 - val_loss: 0.6464 - val_accuracy: 0.6259
Epoch 20/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6387 - accuracy:
0.6311 - val_loss: 0.6464 - val_accuracy: 0.6283
Epoch 21/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6382 - accuracy:
0.6312 - val_loss: 0.6462 - val_accuracy: 0.6277
Epoch 21: early stopping
```

[58]:
```
<keras.src.callbacks.History at 0x172d5572bc0>
```

[59]:
```
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: 0.6388 - accuracy:
0.6315 - val_loss: 0.6463 - val_accuracy: 0.6243
Epoch 2/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6396 - accuracy:
0.6328 - val_loss: 0.6458 - val_accuracy: 0.6280
Epoch 3/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6383 - accuracy:
0.6325 - val_loss: 0.6465 - val_accuracy: 0.6271
Epoch 4/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6421 - accuracy:
0.6239 - val_loss: 0.6461 - val_accuracy: 0.6249
Epoch 5/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6391 - accuracy:
0.6306 - val_loss: 0.6461 - val_accuracy: 0.6252
Epoch 6/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6363 - accuracy:
0.6304 - val_loss: 0.6459 - val_accuracy: 0.6265
Epoch 7/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6386 - accuracy:
0.6295 - val_loss: 0.6458 - val_accuracy: 0.6283
Epoch 8/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6387 - accuracy:
0.6316 - val_loss: 0.6459 - val_accuracy: 0.6255
Epoch 9/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6377 - accuracy:
0.6325 - val_loss: 0.6459 - val_accuracy: 0.6227
Epoch 10/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6400 - accuracy:
0.6333 - val_loss: 0.6455 - val_accuracy: 0.6287
Epoch 11/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6373 - accuracy:
0.6321 - val_loss: 0.6454 - val_accuracy: 0.6268
Epoch 12/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6384 - accuracy:
0.6312 - val_loss: 0.6453 - val_accuracy: 0.6255
Epoch 13/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6375 - accuracy:
0.6296 - val_loss: 0.6457 - val_accuracy: 0.6259
Epoch 14/50
```

```
51/51 [==============================] - 0s 4ms/step - loss: 0.6385 - accuracy:
0.6294 - val_loss: 0.6451 - val_accuracy: 0.6299
Epoch 15/50
51/51 [==============================] - 0s 5ms/step - loss: 0.6379 - accuracy:
0.6328 - val_loss: 0.6449 - val_accuracy: 0.6274
Epoch 16/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6377 - accuracy:
0.6345 - val_loss: 0.6450 - val_accuracy: 0.6305
Epoch 17/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6368 - accuracy:
0.6321 - val_loss: 0.6449 - val_accuracy: 0.6318
Epoch 18/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6379 - accuracy:
0.6319 - val_loss: 0.6449 - val_accuracy: 0.6302
Epoch 19/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6373 - accuracy:
0.6357 - val_loss: 0.6450 - val_accuracy: 0.6255
Epoch 20/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6360 - accuracy:
0.6315 - val_loss: 0.6447 - val_accuracy: 0.6252
Epoch 21/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6381 - accuracy:
0.6346 - val_loss: 0.6445 - val_accuracy: 0.6287
Epoch 22/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6375 - accuracy:
0.6346 - val_loss: 0.6449 - val_accuracy: 0.6277
Epoch 23/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6384 - accuracy:
0.6305 - val_loss: 0.6445 - val_accuracy: 0.6352
Epoch 24/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6368 - accuracy:
0.6311 - val_loss: 0.6447 - val_accuracy: 0.6305
Epoch 25/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6365 - accuracy:
0.6364 - val_loss: 0.6445 - val_accuracy: 0.6314
Epoch 26/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6385 - accuracy:
0.6318 - val_loss: 0.6442 - val_accuracy: 0.6305
Epoch 27/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6372 - accuracy:
0.6341 - val_loss: 0.6443 - val_accuracy: 0.6364
Epoch 28/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6372 - accuracy:
0.6349 - val_loss: 0.6442 - val_accuracy: 0.6380
Epoch 29/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6362 - accuracy:
0.6309 - val_loss: 0.6442 - val_accuracy: 0.6358
Epoch 30/50
```

```
51/51 [==============================] - 0s 4ms/step - loss: 0.6386 - accuracy:
0.6328 - val_loss: 0.6437 - val_accuracy: 0.6318
Epoch 31/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6352 - accuracy:
0.6339 - val_loss: 0.6440 - val_accuracy: 0.6293
Epoch 32/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6362 - accuracy:
0.6332 - val_loss: 0.6440 - val_accuracy: 0.6346
Epoch 33/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6355 - accuracy:
0.6364 - val_loss: 0.6440 - val_accuracy: 0.6339
Epoch 34/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6374 - accuracy:
0.6314 - val_loss: 0.6440 - val_accuracy: 0.6336
Epoch 35/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6355 - accuracy:
0.6353 - val_loss: 0.6442 - val_accuracy: 0.6333
Epoch 36/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6361 - accuracy:
0.6369 - val_loss: 0.6441 - val_accuracy: 0.6401
Epoch 37/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6345 - accuracy:
0.6367 - val_loss: 0.6439 - val_accuracy: 0.6364
Epoch 38/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6365 - accuracy:
0.6331 - val_loss: 0.6434 - val_accuracy: 0.6358
Epoch 39/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6378 - accuracy:
0.6302 - val_loss: 0.6435 - val_accuracy: 0.6330
Epoch 40/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6380 - accuracy:
0.6317 - val_loss: 0.6435 - val_accuracy: 0.6339
Epoch 41/50
51/51 [==============================] - 0s 4ms/step - loss: 0.6350 - accuracy:
0.6332 - val_loss: 0.6433 - val_accuracy: 0.6364
Epoch 42/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6363 - accuracy:
0.6348 - val_loss: 0.6435 - val_accuracy: 0.6370
Epoch 43/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6353 - accuracy:
0.6338 - val_loss: 0.6431 - val_accuracy: 0.6358
Epoch 44/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6364 - accuracy:
0.6333 - val_loss: 0.6434 - val_accuracy: 0.6398
Epoch 45/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6360 - accuracy:
0.6346 - val_loss: 0.6433 - val_accuracy: 0.6321
Epoch 46/50
```

```
51/51 [==============================] - 0s 3ms/step - loss: 0.6351 - accuracy:
0.6311 - val_loss: 0.6434 - val_accuracy: 0.6377
Epoch 47/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6373 - accuracy:
0.6306 - val_loss: 0.6436 - val_accuracy: 0.6364
Epoch 48/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6361 - accuracy:
0.6297 - val_loss: 0.6432 - val_accuracy: 0.6389
Epoch 49/50
51/51 [==============================] - 0s 3ms/step - loss: 0.6345 - accuracy:
0.6335 - val_loss: 0.6432 - val_accuracy: 0.6380
Epoch 50/50
51/51 [==============================] - 0s 2ms/step - loss: 0.6362 - accuracy:
0.6333 - val_loss: 0.6433 - val_accuracy: 0.6349
```

[60]:
```
model.evaluate(X_test,y_test)
```

```
101/101 [==============================] - 0s 1ms/step - loss: 0.6433 -
accuracy: 0.6349
```

[60]:
```
[0.6433086395263672, 0.6348663568496704]
```

[61]:
```
y_pred=model.predict(X_test)
```

```
101/101 [==============================] - 0s 1ms/step
```

[62]:
```
y_pred
```

[62]:
```
array([[0.48226872],
       [0.7590577 ],
       [0.5538002 ],
       …,
       [0.66819197],
       [0.49153423],
       [0.5063078 ]], dtype=float32)
```

[63]:
```
predictions=(y_pred>0.5).astype('int')
```

[64]:
```
predictions
```

[64]:
```
array([[0],
       [1],
       [1],
       …,
       [1],
       [0],
       [1]])
```

[65]:
```
y_test
```

[65]:
```
5033    0
8164    1
9048    1
3109    0
8346    1
       ..
4421    0
2149    0
1766    0
430     0
5594    0
Name: not.fully.paid, Length: 3218, dtype: int64
```

[66]:
```
from sklearn.metrics import
 ↪accuracy_score,confusion_matrix,classification_report
accuracy_score(predictions,y_test)
```

[66]:
```
0.6348663766314481
```

[67]:
```
print(classification_report(predictions,y_test))
```

```
              precision    recall  f1-score   support

           0       0.65      0.63      0.64      1669
           1       0.62      0.64      0.63      1549

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

[68]:
```
model.save('loan_default1.h5')
```

```
C:\Users\10030099\AppData\Roaming\Python\Python310\site-
packages\keras\src\engine\training.py:3079: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

0.2 Model2 Architecture

[69]:
```
# batch Normalization
from tensorflow.keras.layers import BatchNormalization
```

[70]:

```python
# create the architecture model2
# First ANN layer
model1=Sequential()
model1.add(Dense(128,activation='relu',input_shape=[9]))
model1.add(BatchNormalization())
model1.add(Dropout(0.20))

# Second ANN layer
model1.add(Dense(64,activation='tanh'))
model1.add(BatchNormalization())
model1.add(Dropout(0.20))


# third ANN layer
model1.add(Dense(32,activation='relu'))
model1.add(BatchNormalization())
model1.add(Dropout(0.20))

# output layer
model1.add(Dense(1,activation='sigmoid'))
```

[71]:

```python
model1.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               1280

 batch_normalization (Batch  (None, 128)               512
 Normalization)

 dropout_2 (Dropout)         (None, 128)               0

 dense_4 (Dense)             (None, 64)                8256

 batch_normalization_1 (Bat  (None, 64)                256
 chNormalization)

 dropout_3 (Dropout)         (None, 64)                0

 dense_5 (Dense)             (None, 32)                2080

 batch_normalization_2 (Bat  (None, 32)                128
 chNormalization)

 dropout_4 (Dropout)         (None, 32)                0
```

```
 dense_6 (Dense)             (None, 1)                 33

=================================================================
Total params: 12545 (49.00 KB)
Trainable params: 12097 (47.25 KB)
Non-trainable params: 448 (1.75 KB)
_________________________________________________________________
```

[72]:
```python
# compile the model
model1.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

[73]:
```python
history=model1.fit(X_train,y_train,
          epochs=100,
          batch_size=256,
          validation_data=(X_test,y_test))
```

```
Epoch 1/100
51/51 [==============================] - 2s 9ms/step - loss: 0.7695 - accuracy:
0.5736 - val_loss: 0.6634 - val_accuracy: 0.5991
Epoch 2/100
51/51 [==============================] - 0s 5ms/step - loss: 0.7053 - accuracy:
0.5875 - val_loss: 0.6556 - val_accuracy: 0.6094
Epoch 3/100
51/51 [==============================] - 0s 6ms/step - loss: 0.6794 - accuracy:
0.6050 - val_loss: 0.6539 - val_accuracy: 0.6007
Epoch 4/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6726 - accuracy:
0.5998 - val_loss: 0.6520 - val_accuracy: 0.6072
Epoch 5/100
51/51 [==============================] - 0s 4ms/step - loss: 0.6608 - accuracy:
0.6141 - val_loss: 0.6503 - val_accuracy: 0.6203
Epoch 6/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6625 - accuracy:
0.6088 - val_loss: 0.6498 - val_accuracy: 0.6231
Epoch 7/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6519 - accuracy:
0.6172 - val_loss: 0.6466 - val_accuracy: 0.6206
Epoch 8/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6472 - accuracy:
0.6261 - val_loss: 0.6441 - val_accuracy: 0.6252
Epoch 9/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6421 - accuracy:
0.6234 - val_loss: 0.6432 - val_accuracy: 0.6259
Epoch 10/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6415 - accuracy:
0.6265 - val_loss: 0.6418 - val_accuracy: 0.6361
Epoch 11/100
```

```
51/51 [==============================] - 0s 5ms/step - loss: 0.6406 - accuracy:
0.6277 - val_loss: 0.6411 - val_accuracy: 0.6305
Epoch 12/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6389 - accuracy:
0.6276 - val_loss: 0.6403 - val_accuracy: 0.6370
Epoch 13/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6350 - accuracy:
0.6335 - val_loss: 0.6434 - val_accuracy: 0.6380
Epoch 14/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6340 - accuracy:
0.6321 - val_loss: 0.6407 - val_accuracy: 0.6377
Epoch 15/100
51/51 [==============================] - 0s 7ms/step - loss: 0.6342 - accuracy:
0.6329 - val_loss: 0.6391 - val_accuracy: 0.6370
Epoch 16/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6330 - accuracy:
0.6421 - val_loss: 0.6384 - val_accuracy: 0.6346
Epoch 17/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6311 - accuracy:
0.6388 - val_loss: 0.6381 - val_accuracy: 0.6392
Epoch 18/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6276 - accuracy:
0.6448 - val_loss: 0.6388 - val_accuracy: 0.6370
Epoch 19/100
51/51 [==============================] - 0s 6ms/step - loss: 0.6296 - accuracy:
0.6394 - val_loss: 0.6369 - val_accuracy: 0.6398
Epoch 20/100
51/51 [==============================] - 0s 7ms/step - loss: 0.6292 - accuracy:
0.6441 - val_loss: 0.6349 - val_accuracy: 0.6377
Epoch 21/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6280 - accuracy:
0.6412 - val_loss: 0.6348 - val_accuracy: 0.6352
Epoch 22/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6266 - accuracy:
0.6417 - val_loss: 0.6333 - val_accuracy: 0.6380
Epoch 23/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6271 - accuracy:
0.6457 - val_loss: 0.6333 - val_accuracy: 0.6467
Epoch 24/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6230 - accuracy:
0.6478 - val_loss: 0.6330 - val_accuracy: 0.6429
Epoch 25/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6216 - accuracy:
0.6548 - val_loss: 0.6319 - val_accuracy: 0.6392
Epoch 26/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6232 - accuracy:
0.6503 - val_loss: 0.6314 - val_accuracy: 0.6367
Epoch 27/100
```

```
51/51 [==============================] - 0s 5ms/step - loss: 0.6214 - accuracy:
0.6503 - val_loss: 0.6309 - val_accuracy: 0.6374
Epoch 28/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6214 - accuracy:
0.6503 - val_loss: 0.6292 - val_accuracy: 0.6448
Epoch 29/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6174 - accuracy:
0.6548 - val_loss: 0.6281 - val_accuracy: 0.6417
Epoch 30/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6161 - accuracy:
0.6558 - val_loss: 0.6274 - val_accuracy: 0.6405
Epoch 31/100
51/51 [==============================] - 0s 6ms/step - loss: 0.6189 - accuracy:
0.6555 - val_loss: 0.6274 - val_accuracy: 0.6423
Epoch 32/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6182 - accuracy:
0.6537 - val_loss: 0.6273 - val_accuracy: 0.6374
Epoch 33/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6152 - accuracy:
0.6547 - val_loss: 0.6269 - val_accuracy: 0.6374
Epoch 34/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6138 - accuracy:
0.6610 - val_loss: 0.6247 - val_accuracy: 0.6442
Epoch 35/100
51/51 [==============================] - 0s 6ms/step - loss: 0.6145 - accuracy:
0.6565 - val_loss: 0.6256 - val_accuracy: 0.6439
Epoch 36/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6122 - accuracy:
0.6627 - val_loss: 0.6231 - val_accuracy: 0.6467
Epoch 37/100
51/51 [==============================] - 0s 6ms/step - loss: 0.6120 - accuracy:
0.6590 - val_loss: 0.6223 - val_accuracy: 0.6454
Epoch 38/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6134 - accuracy:
0.6568 - val_loss: 0.6221 - val_accuracy: 0.6504
Epoch 39/100
51/51 [==============================] - 0s 7ms/step - loss: 0.6109 - accuracy:
0.6617 - val_loss: 0.6211 - val_accuracy: 0.6501
Epoch 40/100
51/51 [==============================] - 0s 6ms/step - loss: 0.6090 - accuracy:
0.6640 - val_loss: 0.6221 - val_accuracy: 0.6507
Epoch 41/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6071 - accuracy:
0.6659 - val_loss: 0.6184 - val_accuracy: 0.6529
Epoch 42/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6088 - accuracy:
0.6638 - val_loss: 0.6184 - val_accuracy: 0.6582
Epoch 43/100
```

```
51/51 [==============================] - 0s 5ms/step - loss: 0.6044 - accuracy:
0.6688 - val_loss: 0.6170 - val_accuracy: 0.6610
Epoch 44/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6063 - accuracy:
0.6660 - val_loss: 0.6166 - val_accuracy: 0.6551
Epoch 45/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6067 - accuracy:
0.6653 - val_loss: 0.6146 - val_accuracy: 0.6619
Epoch 46/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6045 - accuracy:
0.6690 - val_loss: 0.6138 - val_accuracy: 0.6672
Epoch 47/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5998 - accuracy:
0.6750 - val_loss: 0.6137 - val_accuracy: 0.6625
Epoch 48/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6042 - accuracy:
0.6670 - val_loss: 0.6112 - val_accuracy: 0.6690
Epoch 49/100
51/51 [==============================] - 0s 5ms/step - loss: 0.6021 - accuracy:
0.6696 - val_loss: 0.6142 - val_accuracy: 0.6656
Epoch 50/100
51/51 [==============================] - 0s 6ms/step - loss: 0.6009 - accuracy:
0.6718 - val_loss: 0.6096 - val_accuracy: 0.6706
Epoch 51/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5992 - accuracy:
0.6712 - val_loss: 0.6084 - val_accuracy: 0.6700
Epoch 52/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5946 - accuracy:
0.6783 - val_loss: 0.6069 - val_accuracy: 0.6718
Epoch 53/100
51/51 [==============================] - 0s 9ms/step - loss: 0.5976 - accuracy:
0.6734 - val_loss: 0.6079 - val_accuracy: 0.6765
Epoch 54/100
51/51 [==============================] - 0s 8ms/step - loss: 0.5961 - accuracy:
0.6743 - val_loss: 0.6046 - val_accuracy: 0.6722
Epoch 55/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5966 - accuracy:
0.6757 - val_loss: 0.6064 - val_accuracy: 0.6737
Epoch 56/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5948 - accuracy:
0.6804 - val_loss: 0.6057 - val_accuracy: 0.6784
Epoch 57/100
51/51 [==============================] - 0s 7ms/step - loss: 0.5935 - accuracy:
0.6773 - val_loss: 0.6066 - val_accuracy: 0.6706
Epoch 58/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5886 - accuracy:
0.6833 - val_loss: 0.6047 - val_accuracy: 0.6737
Epoch 59/100
```

```
51/51 [==============================] - 0s 7ms/step - loss: 0.5944 - accuracy:
0.6807 - val_loss: 0.6001 - val_accuracy: 0.6843
Epoch 60/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5920 - accuracy:
0.6770 - val_loss: 0.6008 - val_accuracy: 0.6809
Epoch 61/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5938 - accuracy:
0.6792 - val_loss: 0.6018 - val_accuracy: 0.6731
Epoch 62/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5870 - accuracy:
0.6859 - val_loss: 0.5972 - val_accuracy: 0.6740
Epoch 63/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5865 - accuracy:
0.6816 - val_loss: 0.5963 - val_accuracy: 0.6762
Epoch 64/100
51/51 [==============================] - 0s 7ms/step - loss: 0.5835 - accuracy:
0.6872 - val_loss: 0.5964 - val_accuracy: 0.6774
Epoch 65/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5859 - accuracy:
0.6850 - val_loss: 0.5954 - val_accuracy: 0.6812
Epoch 66/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5859 - accuracy:
0.6846 - val_loss: 0.5926 - val_accuracy: 0.6833
Epoch 67/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5872 - accuracy:
0.6882 - val_loss: 0.5917 - val_accuracy: 0.6871
Epoch 68/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5811 - accuracy:
0.6892 - val_loss: 0.5919 - val_accuracy: 0.6865
Epoch 69/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5822 - accuracy:
0.6906 - val_loss: 0.5882 - val_accuracy: 0.6889
Epoch 70/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5804 - accuracy:
0.6913 - val_loss: 0.5909 - val_accuracy: 0.6790
Epoch 71/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5774 - accuracy:
0.6868 - val_loss: 0.5893 - val_accuracy: 0.6896
Epoch 72/100
51/51 [==============================] - 0s 4ms/step - loss: 0.5789 - accuracy:
0.6873 - val_loss: 0.5856 - val_accuracy: 0.6930
Epoch 73/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5803 - accuracy:
0.6969 - val_loss: 0.5853 - val_accuracy: 0.6812
Epoch 74/100
51/51 [==============================] - 0s 7ms/step - loss: 0.5779 - accuracy:
0.6872 - val_loss: 0.5837 - val_accuracy: 0.6849
Epoch 75/100
```

```
51/51 [==============================] - 0s 6ms/step - loss: 0.5792 - accuracy:
0.6913 - val_loss: 0.5883 - val_accuracy: 0.6871
Epoch 76/100
51/51 [==============================] - 0s 7ms/step - loss: 0.5784 - accuracy:
0.6910 - val_loss: 0.5836 - val_accuracy: 0.6911
Epoch 77/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5742 - accuracy:
0.6947 - val_loss: 0.5804 - val_accuracy: 0.6899
Epoch 78/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5766 - accuracy:
0.6889 - val_loss: 0.5809 - val_accuracy: 0.6939
Epoch 79/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5731 - accuracy:
0.6959 - val_loss: 0.5824 - val_accuracy: 0.6930
Epoch 80/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5715 - accuracy:
0.6979 - val_loss: 0.5796 - val_accuracy: 0.6970
Epoch 81/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5742 - accuracy:
0.6944 - val_loss: 0.5777 - val_accuracy: 0.7020
Epoch 82/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5731 - accuracy:
0.6945 - val_loss: 0.5809 - val_accuracy: 0.6976
Epoch 83/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5718 - accuracy:
0.6939 - val_loss: 0.5788 - val_accuracy: 0.6967
Epoch 84/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5740 - accuracy:
0.6948 - val_loss: 0.5766 - val_accuracy: 0.6939
Epoch 85/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5693 - accuracy:
0.7007 - val_loss: 0.5777 - val_accuracy: 0.6871
Epoch 86/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5711 - accuracy:
0.6990 - val_loss: 0.5768 - val_accuracy: 0.6992
Epoch 87/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5679 - accuracy:
0.6989 - val_loss: 0.5755 - val_accuracy: 0.6920
Epoch 88/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5649 - accuracy:
0.7011 - val_loss: 0.5766 - val_accuracy: 0.6948
Epoch 89/100
51/51 [==============================] - 0s 7ms/step - loss: 0.5618 - accuracy:
0.7067 - val_loss: 0.5714 - val_accuracy: 0.7007
Epoch 90/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5649 - accuracy:
0.7040 - val_loss: 0.5724 - val_accuracy: 0.7048
Epoch 91/100
```

```
51/51 [==============================] - 0s 5ms/step - loss: 0.5638 - accuracy:
0.7008 - val_loss: 0.5751 - val_accuracy: 0.7001
Epoch 92/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5682 - accuracy:
0.6972 - val_loss: 0.5733 - val_accuracy: 0.7026
Epoch 93/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5660 - accuracy:
0.7047 - val_loss: 0.5693 - val_accuracy: 0.7020
Epoch 94/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5604 - accuracy:
0.7059 - val_loss: 0.5676 - val_accuracy: 0.7057
Epoch 95/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5663 - accuracy:
0.7028 - val_loss: 0.5670 - val_accuracy: 0.7119
Epoch 96/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5637 - accuracy:
0.7058 - val_loss: 0.5688 - val_accuracy: 0.7042
Epoch 97/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5629 - accuracy:
0.7049 - val_loss: 0.5710 - val_accuracy: 0.7011
Epoch 98/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5626 - accuracy:
0.7044 - val_loss: 0.5673 - val_accuracy: 0.7060
Epoch 99/100
51/51 [==============================] - 0s 6ms/step - loss: 0.5577 - accuracy:
0.7105 - val_loss: 0.5646 - val_accuracy: 0.7067
Epoch 100/100
51/51 [==============================] - 0s 5ms/step - loss: 0.5632 - accuracy:
0.6970 - val_loss: 0.5638 - val_accuracy: 0.7063
```

[74]:
```
model1.evaluate(X_test,y_test)
```

```
101/101 [==============================] - 0s 2ms/step - loss: 0.5638 -
accuracy: 0.7063
```

[74]:
```
[0.5638065934181213, 0.7063393592834473]
```

[75]:
```
model1.evaluate(X_train,y_train)
```

```
403/403 [==============================] - 1s 1ms/step - loss: 0.4888 -
accuracy: 0.7710
```

[75]:
```
[0.4888094365596771, 0.7709757685661316]
```

1 Hyparameter tuning in Keras

[65]:
```
!pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.5-py3-none-any.whl (129 kB)
                           129.5/129.5

kB 2.5 MB/s eta 0:00:00
Collecting keras-core (from keras-tuner)
  Downloading keras_core-0.1.7-py3-none-any.whl (950 kB)
                           950.8/950.8

kB 8.7 MB/s eta 0:00:00
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from keras-tuner) (23.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from keras-tuner) (2.31.0)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: absl-py in /usr/local/lib/python3.10/dist-
packages (from keras-core->keras-tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from keras-core->keras-tuner) (1.23.5)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages
(from keras-core->keras-tuner) (13.6.0)
Collecting namex (from keras-core->keras-tuner)
  Downloading namex-0.0.7-py3-none-any.whl (5.8 kB)
Requirement already satisfied: h5py in /usr/local/lib/python3.10/dist-packages
(from keras-core->keras-tuner) (3.9.0)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.10/dist-
packages (from keras-core->keras-tuner) (0.1.8)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->keras-tuner) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2023.7.22)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras-core->keras-tuner)
(3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras-core->keras-tuner)
(2.16.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-
packages (from markdown-it-py>=2.2.0->rich->keras-core->keras-tuner) (0.1.2)
```

```
Installing collected packages: namex, kt-legacy, keras-core, keras-tuner
Successfully installed keras-core-0.1.7 keras-tuner-1.4.5 kt-legacy-1.0.5
namex-0.0.7
```

[66]:
```
import keras_tuner
import tensorflow
```

```
Using TensorFlow backend
```

[67]:
```
def build_model(hp):
    model=Sequential()

    # first hidden layer
    model.add(Dense(units=hp.Int('units',min_value=32,max_value=1024,step=16),
                    activation=hp.
 ↪Choice('activation',['relu','tanh']),input_shape=[9]))

    model.add(BatchNormalization())
    model.add(Dropout(hp.Float('rate',min_value=0.1,max_value=0.5,step=0.1)))


    # Second hidden layer
    model.add(Dense(units=hp.Int('units',min_value=32,max_value=1024,step=16),
                    activation=hp.Choice('activation',['relu','tanh'])))

    model.add(BatchNormalization())
    model.add(Dropout(hp.Float('rate',min_value=0.1,max_value=0.5,step=0.1)))

     # third hidden layer
    model.add(Dense(units=hp.Int('units',min_value=32,max_value=1024,step=16),
                    activation=hp.Choice('activation',['relu','tanh'])))

    model.add(BatchNormalization())
    model.add(Dropout(hp.Float('rate',min_value=0.1,max_value=0.5,step=0.1)))

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

    learning_rate=hp.Float('learning_rate',min_value=0.001,max_value=0.1,step=0.
 ↪01)

    model.compile(loss='binary_crossentropy',
                  optimizer=tensorflow.keras.optimizers.Adam(learning_rate),
                 metrics=['accuracy'])
    return model
```

[68]:
```
import keras_tuner as kt
```

[69]:
```
build_model(kt.HyperParameters())
```

[69]:
```
<keras.src.engine.sequential.Sequential at 0x79a757119390>
```

[70]:
```
rtuner=kt.RandomSearch(hypermodel=build_model,
                       objective='val_accuracy',
                       max_trials=10
                      )
```

[71]:
```
rtuner.search(X_train,y_train,
             epochs=50,validation_data=(X_test,y_test),
             verbose=2)
```

```
Trial 10 Complete [00h 02m 24s]
val_accuracy: 0.6277191042900085

Best val_accuracy So Far: 0.7560596466064453
Total elapsed time: 00h 24m 39s
```

[72]:
```
par=rtuner.get_best_hyperparameters()
```

for h_param in [f"units{i}" for i in range(1,4)]+['learning_rate']:

print(h_param,rtuner.get_best_hyperparameters()[0].get(h_param))

[73]:
```
par
```

[73]:
```
[<keras_tuner.src.engine.hyperparameters.hyperparameters.HyperParameters at
0x79a7302faef0>]
```

[74]:
```
models=rtuner.get_best_models()
```

[75]:
```
len(models)
```

[75]: 1

[76]:
```
models[0].summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 448)               4480

 batch_normalization (Batch  (None, 448)               1792
 Normalization)

 dropout (Dropout)           (None, 448)               0
```

```
 dense_1 (Dense)             (None, 448)               201152

 batch_normalization_1 (Bat  (None, 448)               1792
 chNormalization)

 dropout_1 (Dropout)         (None, 448)               0

 dense_2 (Dense)             (None, 448)               201152

 batch_normalization_2 (Bat  (None, 448)               1792
 chNormalization)

 dropout_2 (Dropout)         (None, 448)               0

 dense_3 (Dense)             (None, 1)                 449

=================================================================
Total params: 412609 (1.57 MB)
Trainable params: 409921 (1.56 MB)
Non-trainable params: 2688 (10.50 KB)
_________________________________________________________________
```

[77]:
```
y_pred=models[0].predict(X_test)>=0.5
```

```
101/101 [==============================] - 0s 2ms/step
```

[78]:
```
y_pred
```

[78]:
```
array([[False],
       [ True],
       [ True],
       …,
       [False],
       [ True],
       [ True]])
```

[79]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

[79]: 0.7560596643878186

1.0.1 After applyting hyper parameter tuning we noticed that the final accuracy score has come to 0.75 which is a very good outcome

[ ]:
```

```