In [74]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

import tensorflow
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from sklearn.metrics import confusion_matrix, accuracy_score , classification_report

import warnings
warnings.filterwarnings('ignore')
```

In [75]:

```
#reading the data set
data = pd.read_csv("C:/Users/david/Desktop/personal/AI/course-4_PC AIML - Deep Learning/project/1596018188_datasets (1)/loan_data
data.head()
```

Out[75]:

	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
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 [76]:

```
# 1.Feature Transformation
    # Transform categorical values into numerical values (discrete)
```

In [77]:

```
df= pd.get_dummies(data, columns = ["purpose"])
df.head()
```

Out[77]:

	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 [78]:

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

In [79]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 20 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   credit.policy               9578 non-null   int64  
 1   int.rate                    9578 non-null   float64
 2   installment                 9578 non-null   float64
 3   log.annual.inc              9578 non-null   float64
 4   dti                         9578 non-null   float64
 5   fico                        9578 non-null   int64  
 6   days.with.cr.line           9578 non-null   float64
 7   revol.bal                   9578 non-null   int64  
 8   revol.util                  9578 non-null   float64
 9   inq.last.6mths              9578 non-null   int64  
 10  delinq.2yrs                 9578 non-null   int64  
 11  pub.rec                     9578 non-null   int64  
 12  not.fully.paid              9578 non-null   int64  
 13  purpose_all_other           9578 non-null   uint8  
 14  purpose_credit_card         9578 non-null   uint8  
 15  purpose_debt_consolidation  9578 non-null   uint8  
 16  purpose_educational         9578 non-null   uint8  
 17  purpose_home_improvement    9578 non-null   uint8  
 18  purpose_major_purchase      9578 non-null   uint8  
 19  purpose_small_business      9578 non-null   uint8  
dtypes: float64(6), int64(7), uint8(7)
memory usage: 1.0 MB
```

In [80]:
```
df.shape
```

Out[80]:
```
(9578, 20)
```

In [81]:
```
df.describe().T
```

Out[81]:

	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
purpose_all_other	9578.0	0.243370	0.429139	0.000000	0.000000	0.000000	0.000000	1.000000e+00
purpose_credit_card	9578.0	0.131760	0.338248	0.000000	0.000000	0.000000	0.000000	1.000000e+00
purpose_debt_consolidation	9578.0	0.413134	0.492422	0.000000	0.000000	0.000000	1.000000	1.000000e+00
purpose_educational	9578.0	0.035811	0.185829	0.000000	0.000000	0.000000	0.000000	1.000000e+00
purpose_home_improvement	9578.0	0.065671	0.247720	0.000000	0.000000	0.000000	0.000000	1.000000e+00
purpose_major_purchase	9578.0	0.045625	0.208682	0.000000	0.000000	0.000000	0.000000	1.000000e+00
purpose_small_business	9578.0	0.064627	0.245880	0.000000	0.000000	0.000000	0.000000	1.000000e+00

In [82]:

```
# here categorical columns are :
# 'credit.policy','not.fully.paid','inq.last.6mths','delinq.2yrs' ,'pub.rec' ,
# 'purpose_all_other', 'purpose_credit_card','purpose_debt_consolidation', 'purpose_educational',
# 'purpose_home_improvement', 'purpose_major_purchase','purpose_small_business'
```

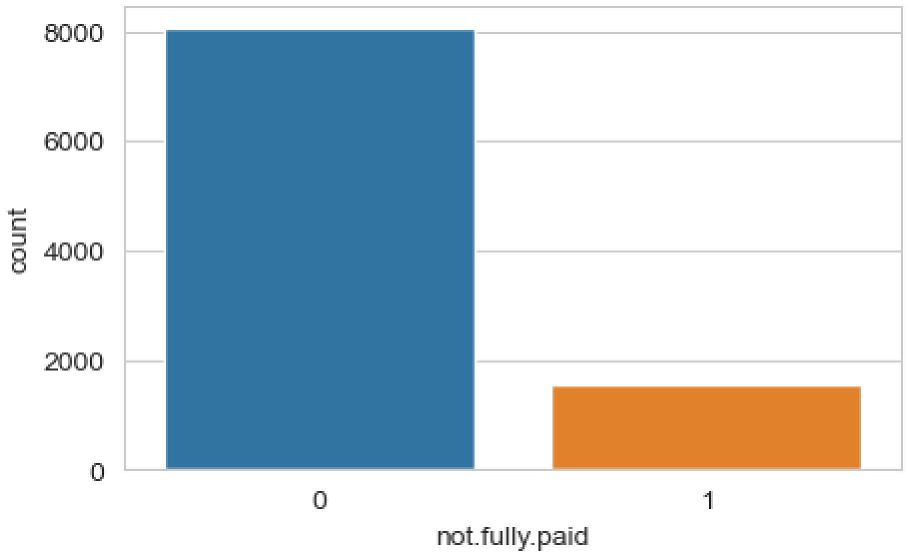
In [83]:
```
#checking balanced or not
```

In [84]:
```
df['not.fully.paid'].value_counts()
```

Out[84]:
```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

In [85]:
```
plt.figure(figsize=(5,3))
sns.countplot(x='not.fully.paid',data=df)
```

Out[85]:
```
<Axes: xlabel='not.fully.paid', ylabel='count'>
```



In [13]:
```
#imbalanced class , we have more samples of fully paid borrowers versus not fully paid borrowers
```

In [14]:
```
#handling imbalanced dataset - using resample

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

print('not_fully_paid_0', not_fully_paid_0.shape)
print('not_fully_paid_1', not_fully_paid_1.shape)
```

```
not_fully_paid_0 (8045, 20)
not_fully_paid_1 (1533, 20)
```

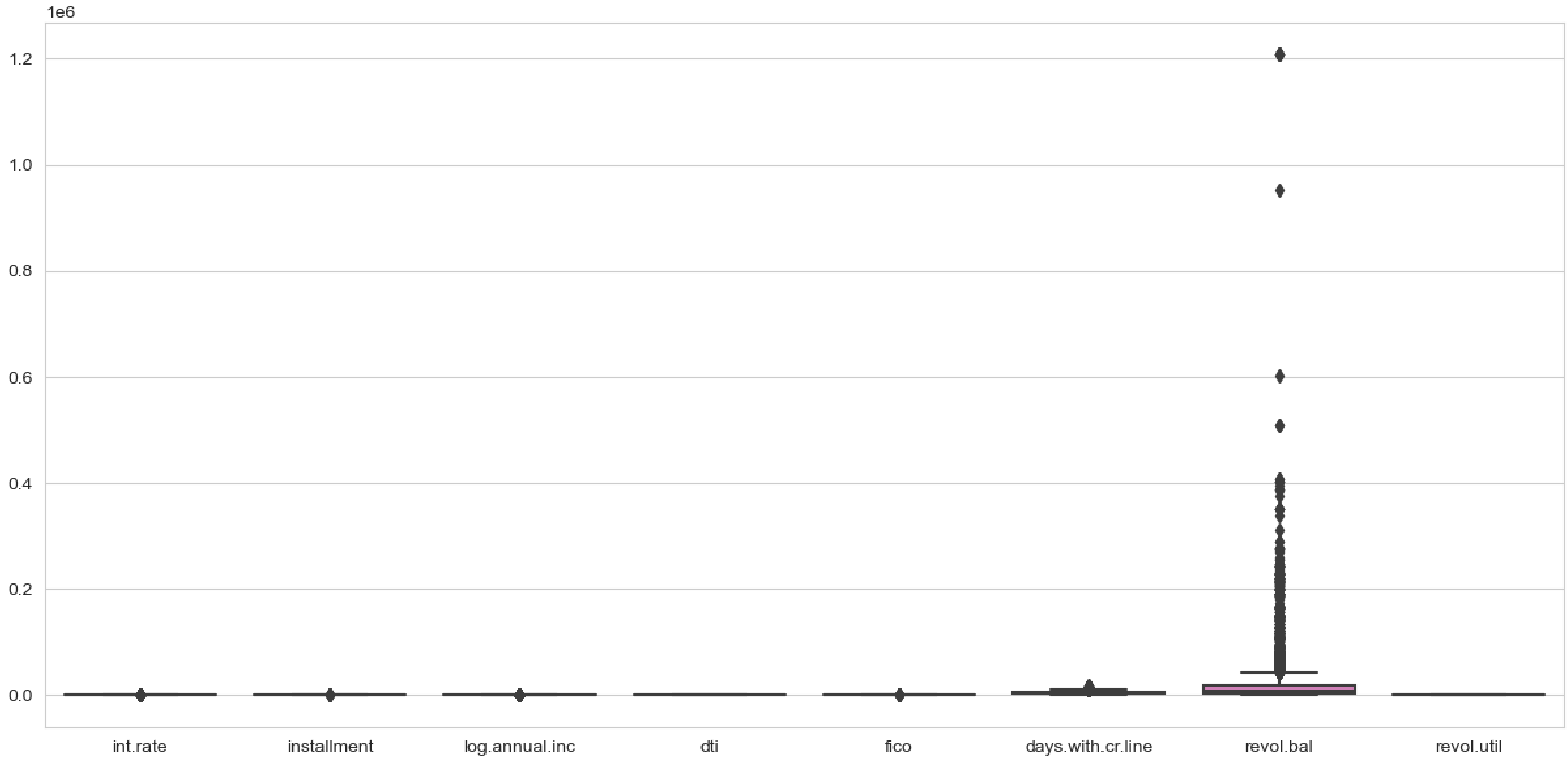
In [15]:
```
from sklearn.utils import resample
df_minority_upsampled = resample(not_fully_paid_1, replace = True, n_samples = 8045)
df = pd.concat([not_fully_paid_0, df_minority_upsampled])

from sklearn.utils import shuffle
df = shuffle(df)
```

In [16]:
```
#imbalanced data handled
df['not.fully.paid'].value_counts()
```

Out[16]:
```
0    8045
1    8045
Name: not.fully.paid, dtype: int64
```

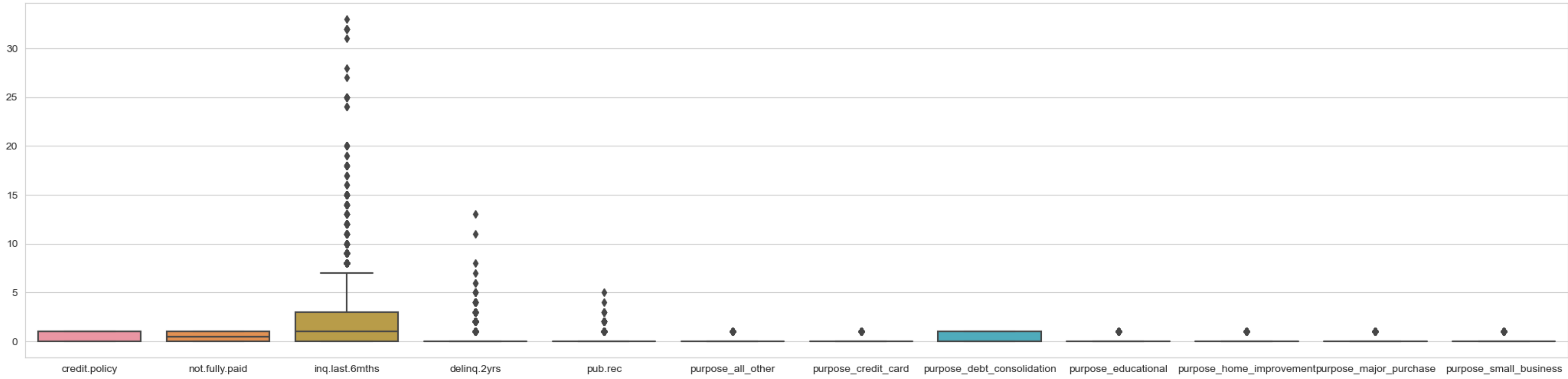
In [17]:
```
# Is there any outliers ?

cols_1=['int.rate', 'installment', 'log.annual.inc', 'dti','fico', 'days.with.cr.line', 'revol.bal', 'revol.util']

sns.set_style('whitegrid')
plt.figure(figsize = (15,7))
sns.boxplot(data=df[cols_1])
```

Out[17]:
```
<Axes: >
```



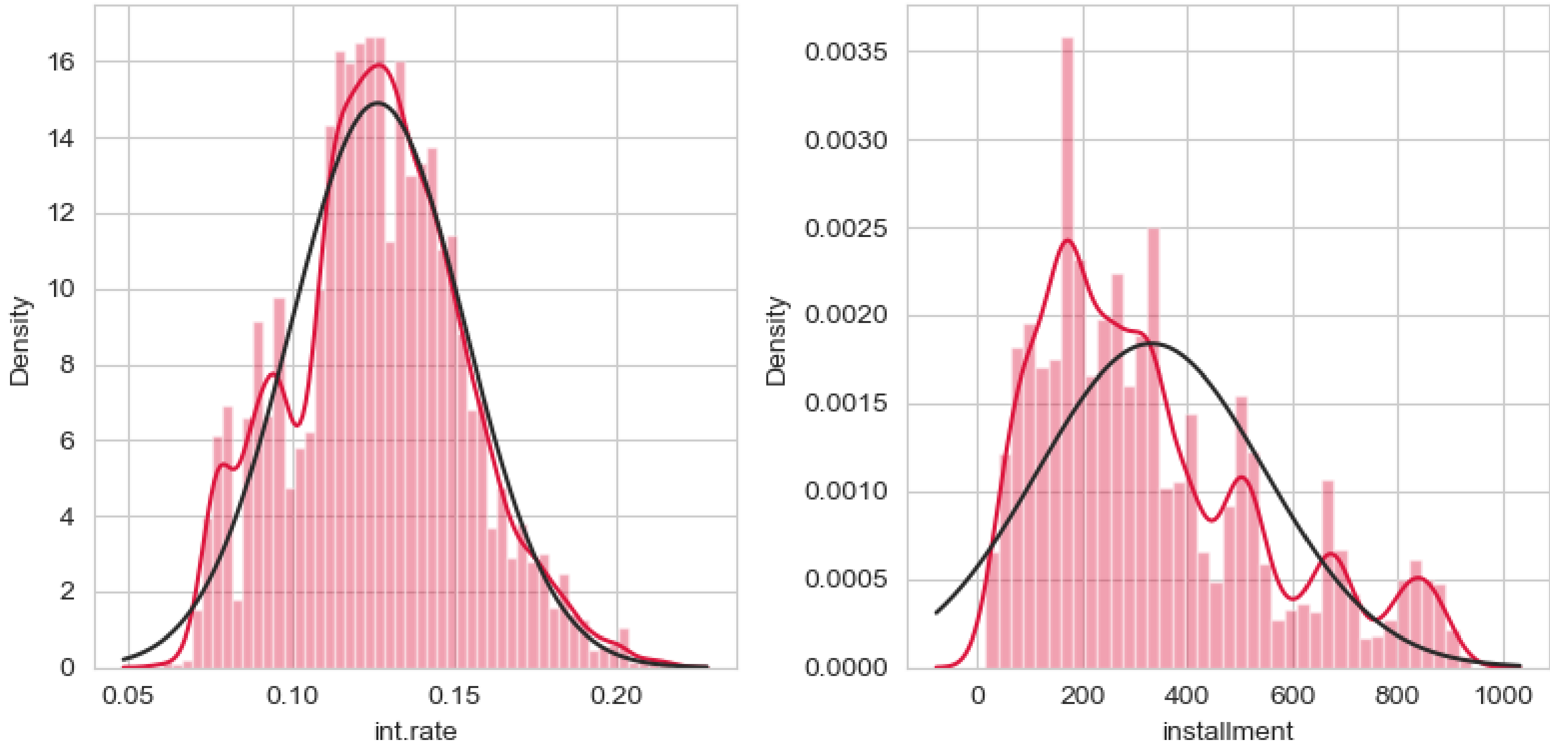
In [23]:
```
cols_2=['credit.policy','not.fully.paid','inq.last.6mths','delinq.2yrs','pub.rec',
       'purpose_all_other', 'purpose_credit_card',
       'purpose_debt_consolidation', 'purpose_educational',
       'purpose_home_improvement', 'purpose_major_purchase',
       'purpose_small_business']

sns.set_style('whitegrid')
plt.figure(figsize = (26,6))
sns.boxplot(data=df[cols_2])
```

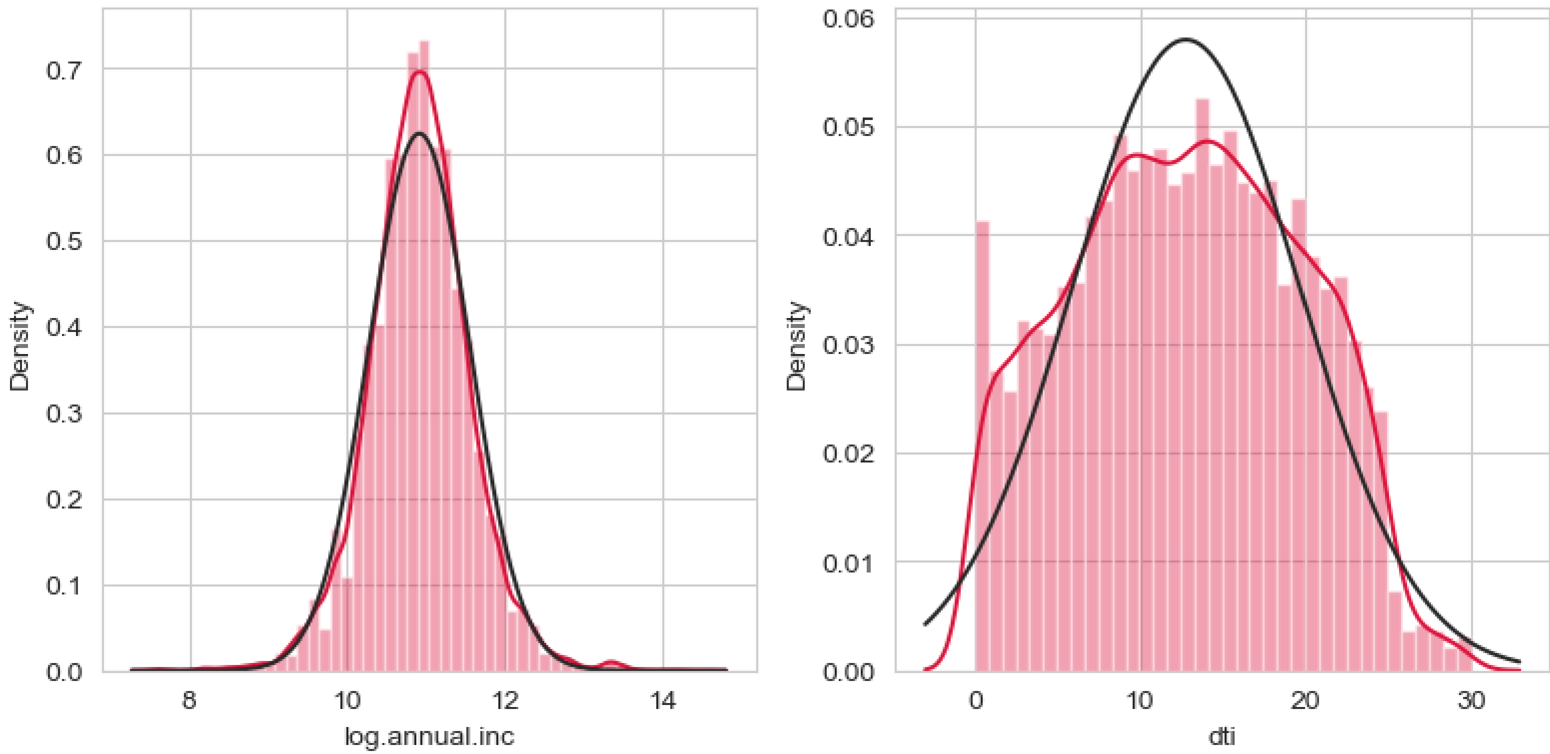
Out[23]: <Axes: >



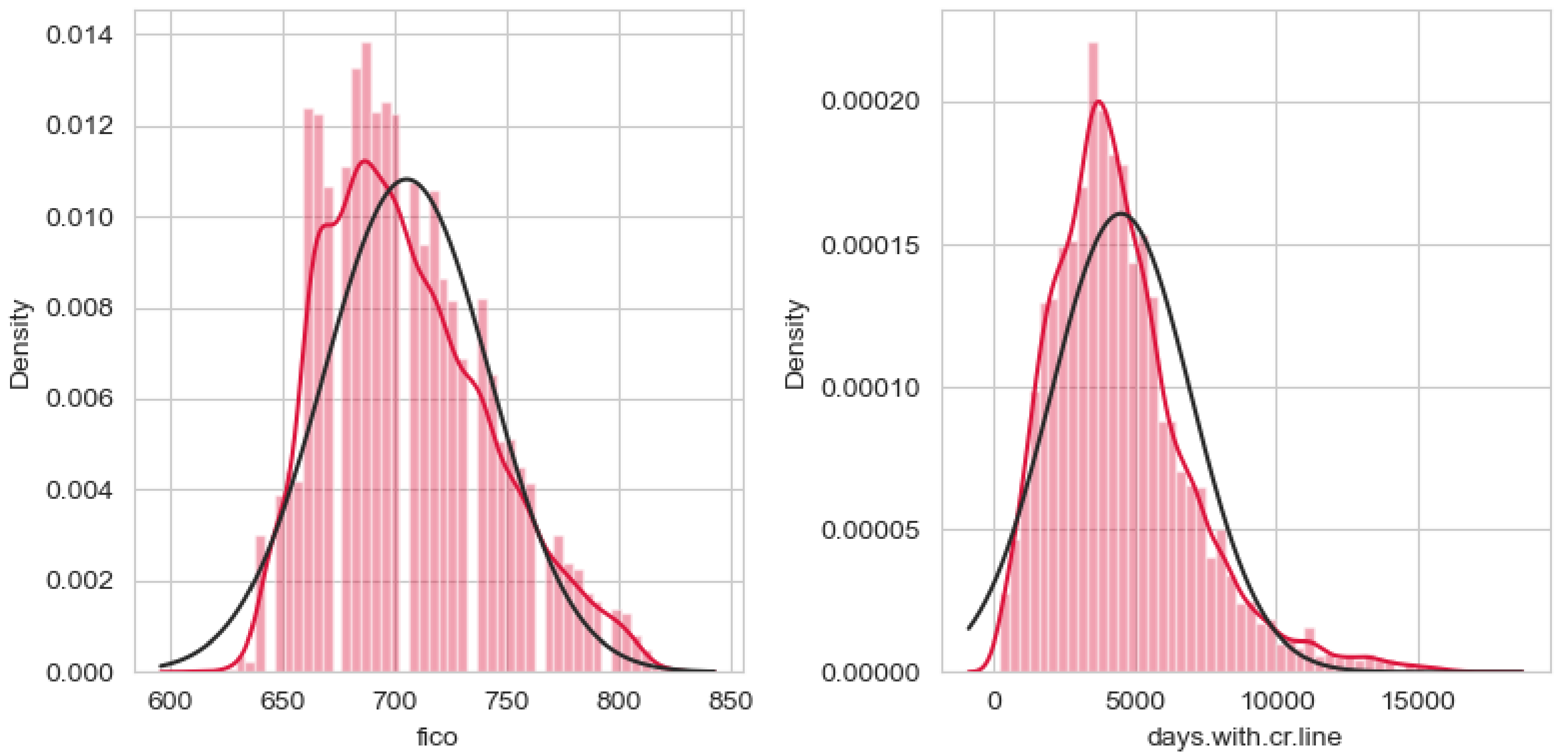
In [24]:
```
# outliers are present in the dataset
```

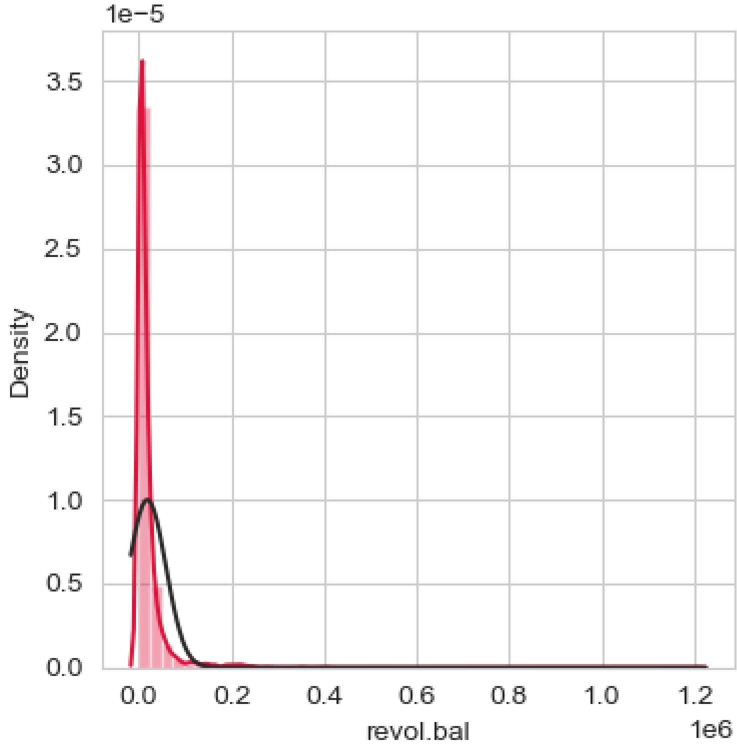
In [25]:
```
from scipy.stats import norm
import seaborn as sns
cols_1=['int.rate', 'installment', 'log.annual.inc', 'dti','fico', 'days.with.cr.line', 'revol.bal', 'revol.util']


for i in range(0,len(cols_1),2):
 plt.figure(figsize=(8,4))
 plt.subplot(121)
 sns.distplot(df[cols_1[i]], kde=True,fit = norm, color = 'crimson')
 plt.subplot(122)
 sns.distplot(df[cols_1[i+1]], kde=True,fit = norm, color = 'crimson')
 plt.tight_layout()
 plt.show()
```

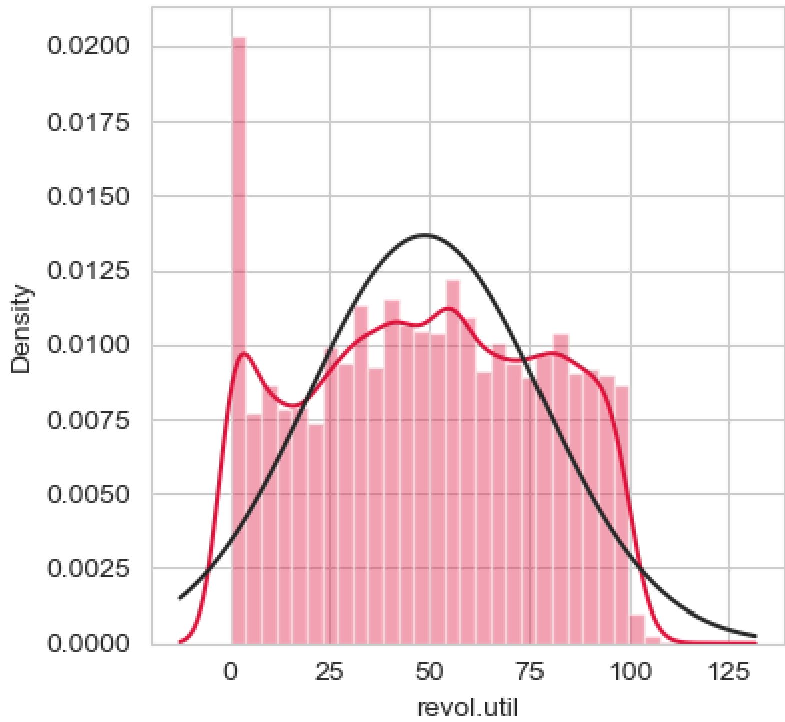

Density
0
2
4
6
8
10
12
14
16
0.05
0.10
0.15
0.20
int.rate
Density
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
0.0035
0
200
400
600
800
1000
installment



Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
8
10
12
14
log.annual.inc
Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0
10
20
30
dti


0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Density
600
650
700
750
800
850
fico
0.00020
0.00015
0.00010
0.00005
0.00000
Density
0
5000
10000
15000
days.with.cr.line





In [26]:

```
#Removing outliers
```

In [27]:

```
# Detect outliers in combined data set
def detect_outlier(feature):
    outliers = []
    data = df[feature]
    mean = np.mean(data)
    std =np.std(data)

    for y in data:
        z_score= (y - mean)/std
        if np.abs(z_score) > 3:
            outliers.append(y)
    print(f"\nOutlier caps for {feature}")
    print('  --95p: {:.1f} / {} values exceed that'.format(data.quantile(.95),
                                                             len([i for i in data
                                                                  if i > data.quantile(.95)])))

    print('  --3sd: {:.1f} / {} values exceed that'.format(mean + 3*(std), len(outliers)))
```

```
    print('  --99p: {:.1f} / {} values exceed that'.format(data.quantile(.99),
                                                          len([i for i in data
                                                               if i > data.quantile(.99)])))
```

In [28]:
```
# Determine what the upperbound should be for continuous features in dataframe.
for feat in df:
    detect_outlier(feat)
```

```
Outlier caps for credit.policy
  --95p: 1.0 / 0 values exceed that
  --3sd: 2.1 / 0 values exceed that
  --99p: 1.0 / 0 values exceed that

Outlier caps for int.rate
  --95p: 0.2 / 805 values exceed that
  --3sd: 0.2 / 36 values exceed that
  --99p: 0.2 / 158 values exceed that

Outlier caps for installment
  --95p: 807.6 / 801 values exceed that
  --3sd: 983.4 / 0 values exceed that
  --99p: 878.9 / 157 values exceed that

Outlier caps for log.annual.inc
  --95p: 11.9 / 800 values exceed that
  --3sd: 12.8 / 168 values exceed that
  --99p: 12.6 / 156 values exceed that

Outlier caps for dti
  --95p: 23.8 / 800 values exceed that
  --3sd: 33.5 / 0 values exceed that
  --99p: 26.8 / 160 values exceed that

Outlier caps for fico
  --95p: 777.0 / 672 values exceed that
  --3sd: 816.2 / 14 values exceed that
  --99p: 802.0 / 103 values exceed that

Outlier caps for days.with.cr.line
  --95p: 9150.0 / 802 values exceed that
  --3sd: 11948.3 / 239 values exceed that
  --99p: 12930.0 / 159 values exceed that

Outlier caps for revol.bal
  --95p: 62739.0 / 804 values exceed that
  --3sd: 138068.5 / 293 values exceed that
  --99p: 191303.0 / 158 values exceed that

Outlier caps for revol.util
  --95p: 94.6 / 793 values exceed that
  --3sd: 136.4 / 0 values exceed that
  --99p: 99.1 / 156 values exceed that
```

```
Outlier caps for inq.last.6mths
  --95p: 6.0 / 790 values exceed that
  --3sd: 9.6 / 234 values exceed that
  --99p: 12.0 / 127 values exceed that

Outlier caps for delinq.2yrs
  --95p: 1.0 / 507 values exceed that
  --3sd: 1.8 / 507 values exceed that
  --99p: 3.0 / 49 values exceed that

Outlier caps for pub.rec
  --95p: 1.0 / 33 values exceed that
  --3sd: 0.9 / 1128 values exceed that
  --99p: 1.0 / 33 values exceed that

Outlier caps for not.fully.paid
  --95p: 1.0 / 0 values exceed that
  --3sd: 2.0 / 0 values exceed that
  --99p: 1.0 / 0 values exceed that

Outlier caps for purpose_all_other
  --95p: 1.0 / 0 values exceed that
  --3sd: 1.5 / 0 values exceed that
  --99p: 1.0 / 0 values exceed that

Outlier caps for purpose_credit_card
  --95p: 1.0 / 0 values exceed that
  --3sd: 1.1 / 0 values exceed that
  --99p: 1.0 / 0 values exceed that

Outlier caps for purpose_debt_consolidation
  --95p: 1.0 / 0 values exceed that
  --3sd: 1.9 / 0 values exceed that
  --99p: 1.0 / 0 values exceed that

Outlier caps for purpose_educational
  --95p: 0.0 / 657 values exceed that
  --3sd: 0.6 / 657 values exceed that
  --99p: 1.0 / 0 values exceed that

Outlier caps for purpose_home_improvement
  --95p: 1.0 / 0 values exceed that
  --3sd: 0.8 / 1050 values exceed that
```

```
  --99p: 1.0 / 0 values exceed that

Outlier caps for purpose_major_purchase
  --95p: 0.0 / 668 values exceed that
  --3sd: 0.6 / 668 values exceed that
  --99p: 1.0 / 0 values exceed that

Outlier caps for purpose_small_business
  --95p: 1.0 / 0 values exceed that
  --3sd: 0.9 / 1402 values exceed that
  --99p: 1.0 / 0 values exceed that
```

In [29]:

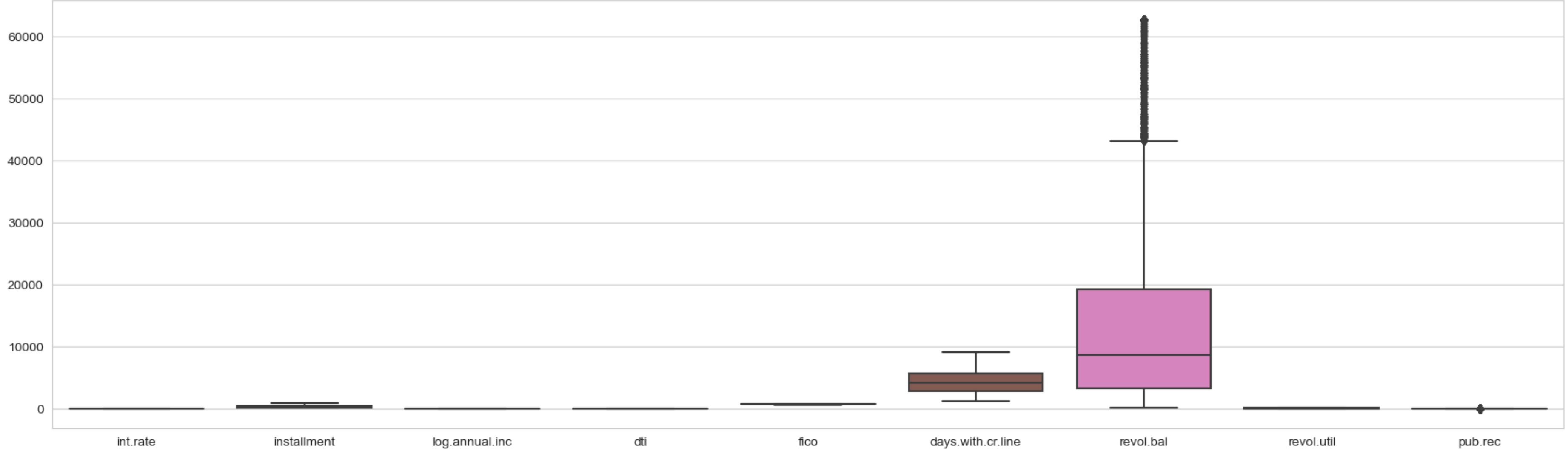
```
# Lower and Upper bounded outliers
for var in df:
    df[var].clip(lower = df[var].quantile(.05), upper = df[var].quantile(0.95), inplace=True)
```

In [31]:

```
cols1 = ['int.rate', 'installment', 'log.annual.inc', 'dti',
       'fico', 'days.with.cr.line', 'revol.bal', 'revol.util', 'pub.rec']

sns.set_style('whitegrid')
plt.figure(figsize = (21,6))
sns.boxplot(data=df[cols1])
```

Out[31]:

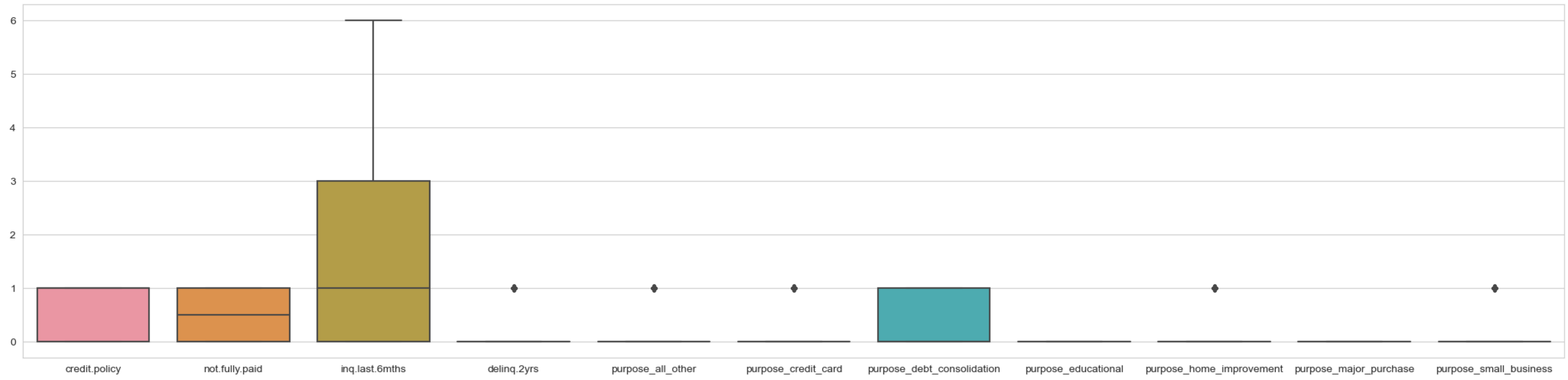
```
<Axes: >
```



In [ ]:
```
# After capping off outliers, we can see that outliers are now elimited in the numerical variables except revol.bal because it st
```

In [33]:
```
cols2=['credit.policy','not.fully.paid','inq.last.6mths','delinq.2yrs',
       'purpose_all_other', 'purpose_credit_card',
       'purpose_debt_consolidation', 'purpose_educational',
       'purpose_home_improvement', 'purpose_major_purchase',
       'purpose_small_business']

sns.set_style('whitegrid')
plt.figure(figsize = (26,6))
sns.boxplot(data=df[cols2])
```

Out[33]: <Axes: >



In [ ]:
```
# removed outliers
```

In [ ]:
```
# 3.  Additional Feature Engineering
    # You will check the correlation between features and will drop those features which have a strong correlation
```

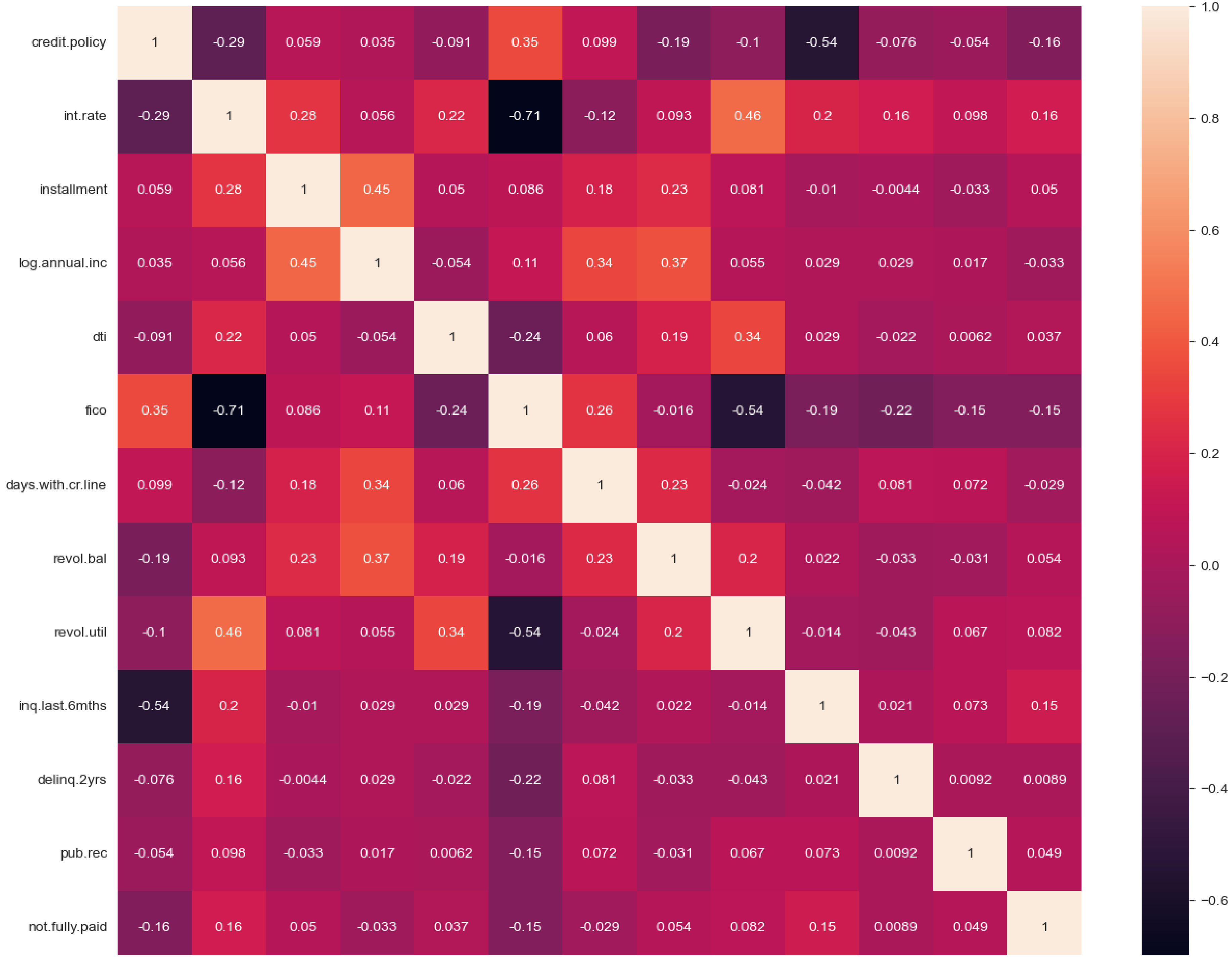
In [34]:
```
df.corr()
```

Out[34]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mt
credit.policy	1.000000	-0.282765	0.053693	0.019055	-0.072089	0.367873	0.086316	-0.096409	-0.084485	-0.5847
int.rate	-0.282765	1.000000	0.259678	0.077954	0.197713	-0.702432	-0.111021	0.105973	0.427965	0.1972
installment	0.053693	0.259678	1.000000	0.477643	0.027296	0.112259	0.196378	0.327932	0.057627	-0.0038
log.annual.inc	0.019055	0.077954	0.477643	1.000000	-0.030829	0.105569	0.373146	0.496275	0.083805	0.0314
dti	-0.072089	0.197713	0.027296	-0.030829	1.000000	-0.208458	0.101486	0.294946	0.318520	0.0175
fico	0.367873	-0.702432	0.112259	0.105569	-0.208458	1.000000	0.249862	-0.016108	-0.492924	-0.1870
days.with.cr.line	0.086316	-0.111021	0.196378	0.373146	0.101486	0.249862	1.000000	0.342745	0.023656	-0.0138
revol.bal	-0.096409	0.105973	0.327932	0.496275	0.294946	-0.016108	0.342745	1.000000	0.346329	-0.0055
revol.util	-0.084485	0.427965	0.057627	0.083805	0.318520	-0.492924	0.023656	0.346329	1.000000	-0.0401
inq.last.6mths	-0.584724	0.197232	-0.003825	0.031435	0.017553	-0.187073	-0.013847	-0.005507	-0.040142	1.0000
delinq.2yrs	-0.047963	0.164196	0.003193	0.015278	-0.021818	-0.229699	0.081987	-0.058498	-0.030781	-0.0000
pub.rec	-0.064586	0.102742	-0.028703	0.010706	0.030215	-0.162760	0.083195	-0.049377	0.080320	0.1020
not.fully.paid	-0.193486	0.217160	0.078110	-0.044045	0.041874	-0.209217	-0.034872	0.054858	0.103689	0.1906
purpose_all_other	-0.025119	-0.120703	-0.205774	-0.084775	-0.124782	0.054284	-0.077735	-0.119198	-0.120135	0.0091
purpose_credit_card	0.012249	-0.043799	-0.000311	0.076602	0.078221	-0.007528	0.047579	0.115706	0.089308	-0.0445
purpose_debt_consolidation	0.024277	0.089997	0.116552	-0.047857	0.181831	-0.140818	0.001324	0.047707	0.204263	-0.0589
purpose_educational	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
purpose_home_improvement	-0.021902	-0.038005	0.019066	0.103361	-0.086892	0.080116	0.068575	-0.015174	-0.108331	0.0786
purpose_major_purchase	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
purpose_small_business	-0.004159	0.165559	0.202025	0.135717	-0.062956	0.072581	0.055380	0.084634	-0.056934	0.0463

In [35]:

```
plt.figure(figsize = (15,12))
sns.heatmap(data = data.corr(), annot = True)
plt.show()
```


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
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

In [36]:
```
# from the correlation heatmaps we can observe that no two features have positive corelation of more than 0.7 , so we will not re
```

In [37]:
```
# 4. Modeling

    # After applying EDA and feature engineering, you are now ready to build the predictive models

    # In this part, you will create a deep learning model using Keras with Tensorflow backend
```

In [38]:
```
#Identify X and Y

X = df.drop("not.fully.paid", axis = 1)
y = df["not.fully.paid"]
```

In [39]:
```
# Split X and Y

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.10, stratify = y, random_state = 987)
X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size = 0.15, stratify = y_train, random_state = 987)
print(X_train.shape,y_train.shape)
print(X_val.shape,y_val.shape)
print(X_test.shape,y_test.shape)
```

```
(12308, 19) (12308,)
(2173, 19) (2173,)
(1609, 19) (1609,)
```

In [40]:
```
# feature scaling
scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_val_scale = scaler.transform(X_val)
X_test_scale = scaler.transform(X_test)
```

In [49]:
```
# Lets build the Model
model = Sequential()
# No of Input will be == (total number of train examples , 8)
# where 8 = feature
```

```python
model.add(Input(shape=(X_train_scale.shape[1],)))

# Hidden Layer 1
model.add(Dense(units=200,activation='relu'))
model.add(Dropout(0.2))
# Hidden Layer 2
model.add(Dense(units=200,activation='relu'))
model.add(Dropout(0.2))
# Hidden Layer 3
model.add(Dense(units=200,activation='relu'))
model.add(Dropout(0.2))

# Output Layer - this is a binary classification
model.add(Dense(units=1,activation='sigmoid'))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 200)               4000

 dropout (Dropout)           (None, 200)               0

 dense_1 (Dense)             (None, 200)               40200

 dropout_1 (Dropout)         (None, 200)               0

 dense_2 (Dense)             (None, 200)               40200

 dropout_2 (Dropout)         (None, 200)               0

 dense_3 (Dense)             (None, 1)                 201

=================================================================
Total params: 84601 (330.47 KB)
Trainable params: 84601 (330.47 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [51]:

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

In [52]:

```
history = model.fit(X_train_scale,y_train,
                    validation_data=(X_val_scale, y_val),epochs=100,verbose=1)
```

```
Epoch 1/100
385/385 [==============================] - 3s 5ms/step - loss: 0.6433 - accuracy: 0.6268 - val_loss: 0.6274 - val_accuracy: 0.65
21
Epoch 2/100
385/385 [==============================] - 2s 4ms/step - loss: 0.6285 - accuracy: 0.6455 - val_loss: 0.6227 - val_accuracy: 0.64
80
Epoch 3/100
385/385 [==============================] - 2s 4ms/step - loss: 0.6189 - accuracy: 0.6549 - val_loss: 0.6186 - val_accuracy: 0.66
13
Epoch 4/100
385/385 [==============================] - 2s 4ms/step - loss: 0.6107 - accuracy: 0.6618 - val_loss: 0.6177 - val_accuracy: 0.66
13
Epoch 5/100
385/385 [==============================] - 2s 4ms/step - loss: 0.6021 - accuracy: 0.6702 - val_loss: 0.6152 - val_accuracy: 0.66
36
Epoch 6/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5940 - accuracy: 0.6794 - val_loss: 0.6009 - val_accuracy: 0.67
79
Epoch 7/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5847 - accuracy: 0.6835 - val_loss: 0.5918 - val_accuracy: 0.67
83
Epoch 8/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5735 - accuracy: 0.6935 - val_loss: 0.5877 - val_accuracy: 0.67
88
Epoch 9/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5631 - accuracy: 0.7036 - val_loss: 0.5755 - val_accuracy: 0.68
61
Epoch 10/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5466 - accuracy: 0.7107 - val_loss: 0.5670 - val_accuracy: 0.69
40
Epoch 11/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5415 - accuracy: 0.7167 - val_loss: 0.5569 - val_accuracy: 0.70
64
Epoch 12/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5247 - accuracy: 0.7312 - val_loss: 0.5441 - val_accuracy: 0.70
78
Epoch 13/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5134 - accuracy: 0.7348 - val_loss: 0.5312 - val_accuracy: 0.72
02
Epoch 14/100
385/385 [==============================] - 2s 4ms/step - loss: 0.5041 - accuracy: 0.7423 - val_loss: 0.5238 - val_accuracy: 0.71
93
Epoch 15/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4913 - accuracy: 0.7535 - val_loss: 0.5203 - val_accuracy: 0.74
```

```
69
Epoch 16/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4806 - accuracy: 0.7601 - val_loss: 0.5070 - val_accuracy: 0.75
20
Epoch 17/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4714 - accuracy: 0.7667 - val_loss: 0.5044 - val_accuracy: 0.75
20
Epoch 18/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4577 - accuracy: 0.7711 - val_loss: 0.4916 - val_accuracy: 0.76
85
Epoch 19/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4525 - accuracy: 0.7831 - val_loss: 0.4923 - val_accuracy: 0.75
79
Epoch 20/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4437 - accuracy: 0.7806 - val_loss: 0.4695 - val_accuracy: 0.78
00
Epoch 21/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4334 - accuracy: 0.7918 - val_loss: 0.4653 - val_accuracy: 0.77
82
Epoch 22/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4234 - accuracy: 0.7942 - val_loss: 0.4578 - val_accuracy: 0.78
37
Epoch 23/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4163 - accuracy: 0.7996 - val_loss: 0.4695 - val_accuracy: 0.76
81
Epoch 24/100
385/385 [==============================] - 2s 4ms/step - loss: 0.4085 - accuracy: 0.8026 - val_loss: 0.4478 - val_accuracy: 0.78
60
Epoch 25/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3958 - accuracy: 0.8169 - val_loss: 0.4564 - val_accuracy: 0.78
00
Epoch 26/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3920 - accuracy: 0.8127 - val_loss: 0.4368 - val_accuracy: 0.80
03
Epoch 27/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3869 - accuracy: 0.8172 - val_loss: 0.4310 - val_accuracy: 0.81
04
Epoch 28/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3785 - accuracy: 0.8248 - val_loss: 0.4272 - val_accuracy: 0.81
18
Epoch 29/100
385/385 [==============================] - 2s 5ms/step - loss: 0.3741 - accuracy: 0.8268 - val_loss: 0.4145 - val_accuracy: 0.81
78
Epoch 30/100
```

```
385/385 [==============================] - 2s 5ms/step - loss: 0.3707 - accuracy: 0.8265 - val_loss: 0.4215 - val_accuracy: 0.81
78
Epoch 31/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3618 - accuracy: 0.8325 - val_loss: 0.4194 - val_accuracy: 0.80
86
Epoch 32/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3552 - accuracy: 0.8376 - val_loss: 0.4222 - val_accuracy: 0.81
13
Epoch 33/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3450 - accuracy: 0.8420 - val_loss: 0.4141 - val_accuracy: 0.81
50
Epoch 34/100
385/385 [==============================] - 2s 5ms/step - loss: 0.3429 - accuracy: 0.8444 - val_loss: 0.4186 - val_accuracy: 0.81
27
Epoch 35/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3373 - accuracy: 0.8494 - val_loss: 0.4000 - val_accuracy: 0.82
83
Epoch 36/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3272 - accuracy: 0.8519 - val_loss: 0.3950 - val_accuracy: 0.82
14
Epoch 37/100
385/385 [==============================] - 2s 5ms/step - loss: 0.3260 - accuracy: 0.8533 - val_loss: 0.3912 - val_accuracy: 0.82
37
Epoch 38/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3288 - accuracy: 0.8510 - val_loss: 0.3859 - val_accuracy: 0.83
71
Epoch 39/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3262 - accuracy: 0.8513 - val_loss: 0.3800 - val_accuracy: 0.83
02
Epoch 40/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3219 - accuracy: 0.8578 - val_loss: 0.3908 - val_accuracy: 0.83
34
Epoch 41/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3150 - accuracy: 0.8594 - val_loss: 0.3743 - val_accuracy: 0.83
62
Epoch 42/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3104 - accuracy: 0.8608 - val_loss: 0.3800 - val_accuracy: 0.83
43
Epoch 43/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3045 - accuracy: 0.8650 - val_loss: 0.3833 - val_accuracy: 0.84
68
Epoch 44/100
385/385 [==============================] - 2s 4ms/step - loss: 0.3025 - accuracy: 0.8656 - val_loss: 0.3772 - val_accuracy: 0.84
08
```

```
Epoch 45/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2999 - accuracy: 0.8663 - val_loss: 0.3739 - val_accuracy: 0.84
54
Epoch 46/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2910 - accuracy: 0.8740 - val_loss: 0.3742 - val_accuracy: 0.83
62
Epoch 47/100
385/385 [==============================] - 2s 5ms/step - loss: 0.2964 - accuracy: 0.8711 - val_loss: 0.3727 - val_accuracy: 0.85
04
Epoch 48/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2844 - accuracy: 0.8743 - val_loss: 0.3571 - val_accuracy: 0.85
60
Epoch 49/100
385/385 [==============================] - 2s 5ms/step - loss: 0.2857 - accuracy: 0.8737 - val_loss: 0.3569 - val_accuracy: 0.85
18
Epoch 50/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2825 - accuracy: 0.8763 - val_loss: 0.3505 - val_accuracy: 0.85
27
Epoch 51/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2763 - accuracy: 0.8823 - val_loss: 0.3402 - val_accuracy: 0.87
07
Epoch 52/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2800 - accuracy: 0.8772 - val_loss: 0.3692 - val_accuracy: 0.84
31
Epoch 53/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2749 - accuracy: 0.8819 - val_loss: 0.3629 - val_accuracy: 0.84
40
Epoch 54/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2695 - accuracy: 0.8854 - val_loss: 0.3784 - val_accuracy: 0.85
09
Epoch 55/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2603 - accuracy: 0.8854 - val_loss: 0.3619 - val_accuracy: 0.85
14
Epoch 56/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2714 - accuracy: 0.8854 - val_loss: 0.3581 - val_accuracy: 0.85
00
Epoch 57/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2626 - accuracy: 0.8868 - val_loss: 0.3641 - val_accuracy: 0.85
50
Epoch 58/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2691 - accuracy: 0.8867 - val_loss: 0.3559 - val_accuracy: 0.86
06
Epoch 59/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2604 - accuracy: 0.8880 - val_loss: 0.3659 - val_accuracy: 0.85
```

```
32
Epoch 60/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2595 - accuracy: 0.8906 - val_loss: 0.3451 - val_accuracy: 0.86
29
Epoch 61/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2489 - accuracy: 0.8926 - val_loss: 0.3366 - val_accuracy: 0.87
44
Epoch 62/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2488 - accuracy: 0.8923 - val_loss: 0.3303 - val_accuracy: 0.86
65
Epoch 63/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2486 - accuracy: 0.8920 - val_loss: 0.3442 - val_accuracy: 0.87
30
Epoch 64/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2452 - accuracy: 0.8954 - val_loss: 0.3297 - val_accuracy: 0.87
48
Epoch 65/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2505 - accuracy: 0.8923 - val_loss: 0.3360 - val_accuracy: 0.86
88
Epoch 66/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2459 - accuracy: 0.8957 - val_loss: 0.3175 - val_accuracy: 0.87
07
Epoch 67/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2362 - accuracy: 0.9023 - val_loss: 0.3336 - val_accuracy: 0.86
42
Epoch 68/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2387 - accuracy: 0.8991 - val_loss: 0.3439 - val_accuracy: 0.86
33
Epoch 69/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2339 - accuracy: 0.8992 - val_loss: 0.3362 - val_accuracy: 0.87
21
Epoch 70/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2299 - accuracy: 0.9058 - val_loss: 0.3377 - val_accuracy: 0.87
25
Epoch 71/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2298 - accuracy: 0.9043 - val_loss: 0.3159 - val_accuracy: 0.88
13
Epoch 72/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2353 - accuracy: 0.9001 - val_loss: 0.3295 - val_accuracy: 0.87
25
Epoch 73/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2363 - accuracy: 0.9006 - val_loss: 0.3427 - val_accuracy: 0.86
70
Epoch 74/100
```

```
385/385 [==============================] - 2s 4ms/step - loss: 0.2286 - accuracy: 0.9044 - val_loss: 0.3265 - val_accuracy: 0.87
44
Epoch 75/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2229 - accuracy: 0.9072 - val_loss: 0.3309 - val_accuracy: 0.86
29
Epoch 76/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2224 - accuracy: 0.9097 - val_loss: 0.3214 - val_accuracy: 0.87
67
Epoch 77/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2186 - accuracy: 0.9101 - val_loss: 0.3173 - val_accuracy: 0.87
44
Epoch 78/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2274 - accuracy: 0.9058 - val_loss: 0.3037 - val_accuracy: 0.88
96
Epoch 79/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2275 - accuracy: 0.9051 - val_loss: 0.3416 - val_accuracy: 0.86
88
Epoch 80/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2210 - accuracy: 0.9072 - val_loss: 0.3410 - val_accuracy: 0.87
80
Epoch 81/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2190 - accuracy: 0.9096 - val_loss: 0.3255 - val_accuracy: 0.87
90
Epoch 82/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2133 - accuracy: 0.9132 - val_loss: 0.3197 - val_accuracy: 0.87
80
Epoch 83/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2152 - accuracy: 0.9122 - val_loss: 0.3192 - val_accuracy: 0.87
85
Epoch 84/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2180 - accuracy: 0.9105 - val_loss: 0.3288 - val_accuracy: 0.87
53
Epoch 85/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2067 - accuracy: 0.9155 - val_loss: 0.2983 - val_accuracy: 0.88
86
Epoch 86/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2070 - accuracy: 0.9115 - val_loss: 0.3237 - val_accuracy: 0.87
21
Epoch 87/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2106 - accuracy: 0.9123 - val_loss: 0.3172 - val_accuracy: 0.87
94
Epoch 88/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2070 - accuracy: 0.9156 - val_loss: 0.3091 - val_accuracy: 0.88
45
```

```
Epoch 89/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2094 - accuracy: 0.9123 - val_loss: 0.3223 - val_accuracy: 0.88
91
Epoch 90/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2038 - accuracy: 0.9144 - val_loss: 0.3424 - val_accuracy: 0.87
30
Epoch 91/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2037 - accuracy: 0.9147 - val_loss: 0.3306 - val_accuracy: 0.88
22
Epoch 92/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2098 - accuracy: 0.9166 - val_loss: 0.3307 - val_accuracy: 0.87
71
Epoch 93/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2016 - accuracy: 0.9163 - val_loss: 0.3076 - val_accuracy: 0.89
05
Epoch 94/100
385/385 [==============================] - 2s 4ms/step - loss: 0.1970 - accuracy: 0.9192 - val_loss: 0.3079 - val_accuracy: 0.89
23
Epoch 95/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2043 - accuracy: 0.9170 - val_loss: 0.3175 - val_accuracy: 0.87
90
Epoch 96/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2034 - accuracy: 0.9180 - val_loss: 0.3220 - val_accuracy: 0.88
40
Epoch 97/100
385/385 [==============================] - 2s 4ms/step - loss: 0.1912 - accuracy: 0.9231 - val_loss: 0.3263 - val_accuracy: 0.88
73
Epoch 98/100
385/385 [==============================] - 2s 4ms/step - loss: 0.1941 - accuracy: 0.9211 - val_loss: 0.3277 - val_accuracy: 0.88
08
Epoch 99/100
385/385 [==============================] - 2s 4ms/step - loss: 0.2003 - accuracy: 0.9207 - val_loss: 0.3371 - val_accuracy: 0.88
13
Epoch 100/100
385/385 [==============================] - 2s 4ms/step - loss: 0.1973 - accuracy: 0.9183 - val_loss: 0.3008 - val_accuracy: 0.89
42
```

In [92]:

```
X_train_pred= model.predict(X_train_scale)
X_test_pred=model.predict(X_test_scale)
```

```
385/385 [==============================] - 1s 2ms/step
51/51 [==============================] - 0s 2ms/step
```

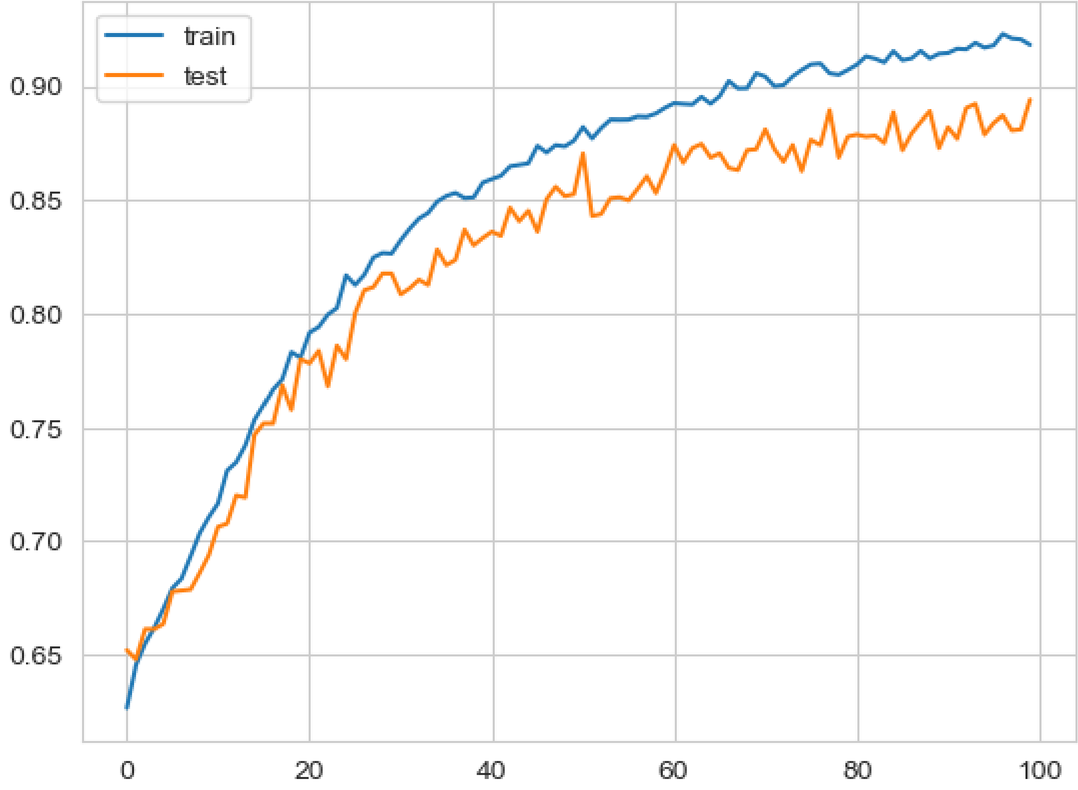
In [94]:
```
cm = confusion_matrix(y_pred=X_train_pred > 0.5,y_true=y_train)
cm
```

Out[94]:
```
array([[6049,  105],
       [  41, 6113]], dtype=int64)
```

In [95]:
```
cm = confusion_matrix(y_pred=X_test_pred > 0.5,y_true=y_test)
cm
```

Out[95]:
```
array([[658, 146],
       [ 26, 779]], dtype=int64)
```

In [61]:
```
from matplotlib import pyplot
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```



In [62]:
```
#training score
model.evaluate(X_train_scale,y_train)
```

```
385/385 [==============================] - 1s 2ms/step - loss: 0.0628 - accuracy: 0.9881
```

Out[62]:
```
[0.0628020241856575, 0.9881377816200256]
```

In [63]:
```
#validatn score
model.evaluate(X_val_scale,y_val)
```

```
68/68 [==============================] - 0s 2ms/step - loss: 0.3008 - accuracy: 0.8942
```

Out[63]:
```
[0.3008177578449249, 0.8941555619239807]
```

In [64]:
```
#testing score
model.evaluate(X_test_scale,y_test)
```

```
51/51 [==============================] - 0s 2ms/step - loss: 0.3353 - accuracy: 0.8931
```

Out[64]:
```
[0.33529841899871826, 0.8931013345718384]
```

In [65]:
```
predictions =(model.predict(X_test_scale)>0.5)
predictions2 =(model.predict(X_val_scale)>0.5)
```

```
51/51 [==============================] - 0s 2ms/step
68/68 [==============================] - 0s 2ms/step
```

In [66]:
```
accuracy_score(y_test, predictions)
```

Out[66]:
```
0.8931013051584835
```

In [67]:
```
accuracy_score(y_val,predictions2)
```

Out[67]:
```
0.8941555453290382
```

In [68]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.96      0.82      0.88       804
           1       0.84      0.97      0.90       805

    accuracy                           0.89      1609
   macro avg       0.90      0.89      0.89      1609
weighted avg       0.90      0.89      0.89      1609
```

In [69]:
```python
print(classification_report(y_val, predictions2))
```

```
              precision    recall  f1-score   support

           0       0.95      0.83      0.89      1087
           1       0.85      0.96      0.90      1086

    accuracy                           0.89      2173
   macro avg       0.90      0.89      0.89      2173
weighted avg       0.90      0.89      0.89      2173
```

In [ ]: