```python
import pandas as pd
import numpy as np

# Data exploration
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import skew

# Data wrangling and feature engineering
from sklearn import preprocessing
from sklearn.feature_selection import chi2
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.preprocessing import LabelEncoder

# Deep learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score, confusion_matrix

df = pd.read_csv('/loan_data.csv')
df.head()
```

```
   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
```

```
df.shape
```

```
(9578, 14)
```

```
df.info()
```

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

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

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

```
not_fully_paid0 = df[df['not.fully.paid']==0]
not_fully_paid1 = df[df['not.fully.paid']==1]
```

```
print('not_fully_paid1    :',not_fully_paid1.shape,'\
n','not_fully_paid0    :',not_fully_paid0.shape)
```

```
not_fully_paid1    : (1533, 14)
 not_fully_paid0    : (8045, 14)
```

```python
from sklearn.utils import resample
df_minority = resample(not_fully_paid1,n_samples = 8045,replace=True)

df = pd.concat([not_fully_paid0,df_minority])

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

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

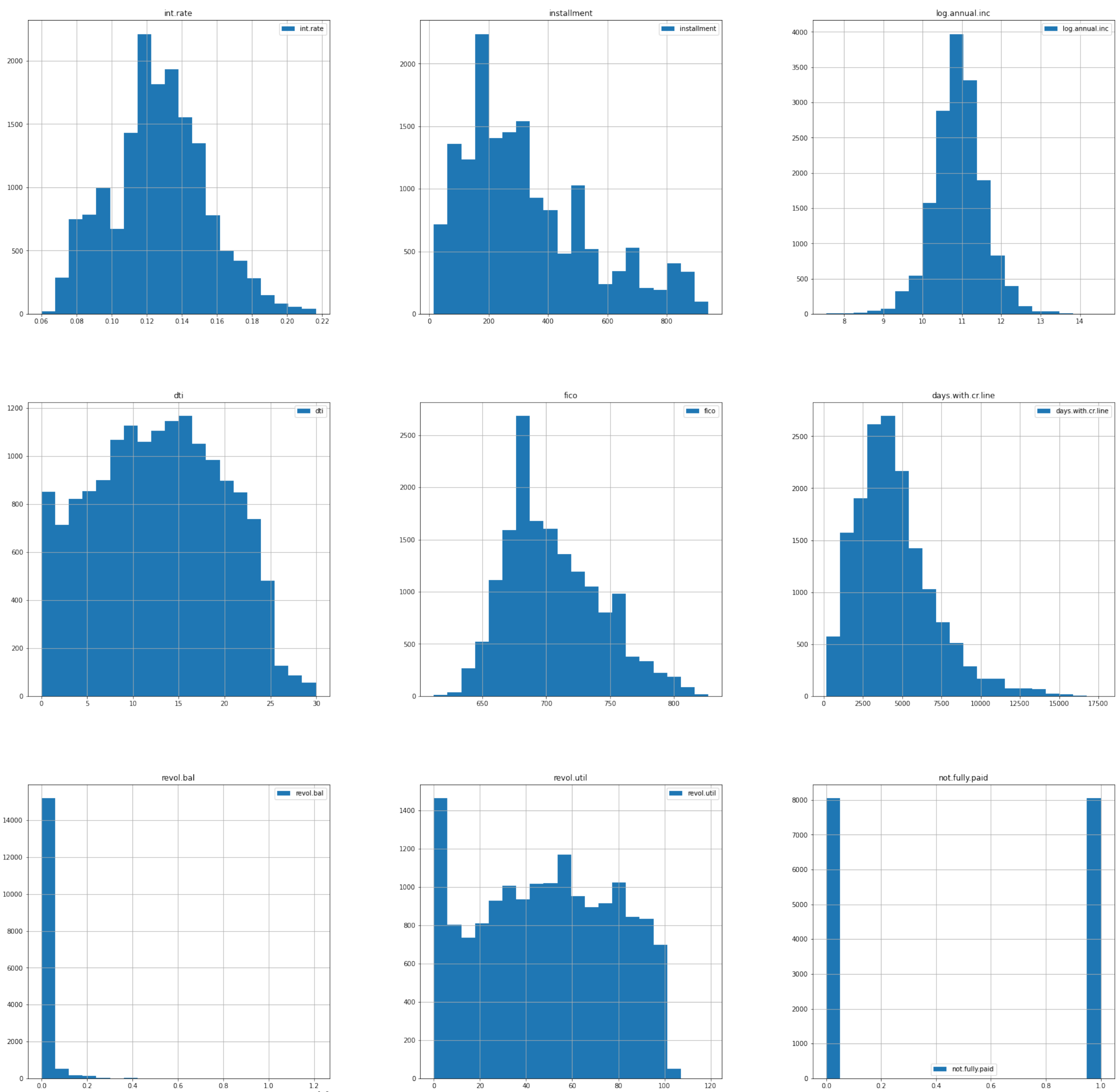
```
1    8045
0    8045
Name: not.fully.paid, dtype: int64
```

```python
# Separate data to include numerical data only
num_data = df[["int.rate", "installment", "log.annual.inc", "dti",
"fico", "days.with.cr.line", "revol.bal",
               "revol.util", "not.fully.paid"]]

# Check the features in the categorical data
cat_data = df[["credit.policy", "purpose", "inq.last.6mths",
"delinq.2yrs", "not.fully.paid"]]

num_data.describe()
```

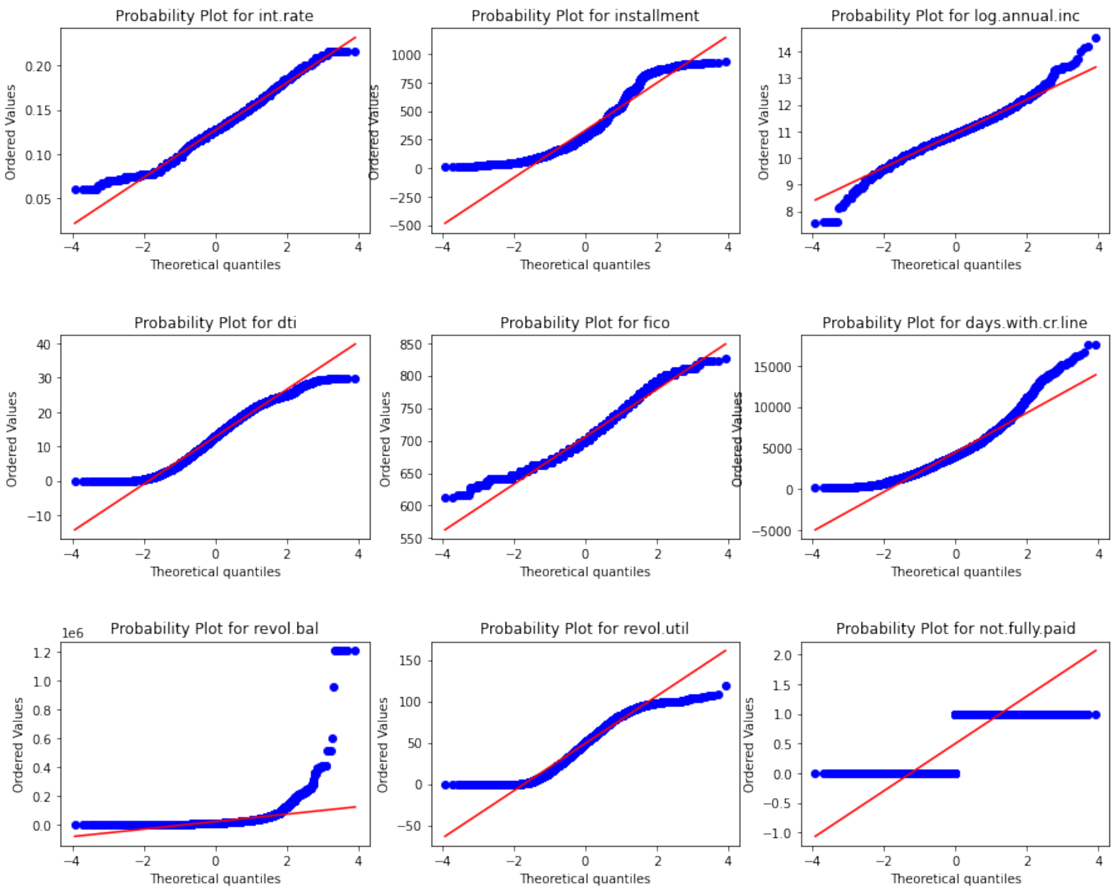
```
           int.rate   installment  log.annual.inc           dti
fico  \
count  16090.000000  16090.000000    16090.000000  16090.000000
16090.000000
mean       0.126717    331.233010       10.918552     12.779421
705.695152
std        0.026932    216.046425        0.639732      6.968271
36.941948
min        0.060000     15.670000        7.547502      0.000000
612.000000
25%        0.110300    167.120000       10.524064      7.280000
677.000000
50%        0.126100    277.450000       10.915088     12.890000
702.000000
75%        0.143800    469.780000       11.289832     18.240000
732.000000
max        0.216400    940.140000       14.528354     29.960000
827.000000

       days.with.cr.line     revol.bal    revol.util  not.fully.paid
count       16090.000000  1.609000e+04  16090.000000    16090.000000
mean         4494.871600  1.921108e+04     48.959622        0.500000
std          2474.362329  4.398412e+04     29.142480        0.500016
```

min	178.958333	0.000000e+00	0.000000	0.000000
25%	2790.000000	3.163250e+03	24.900000	0.000000
50%	4110.000000	8.764000e+03	49.800000	0.500000
75%	5670.041667	1.914400e+04	73.500000	1.000000
max	17639.958330	1.207359e+06	119.000000	1.000000

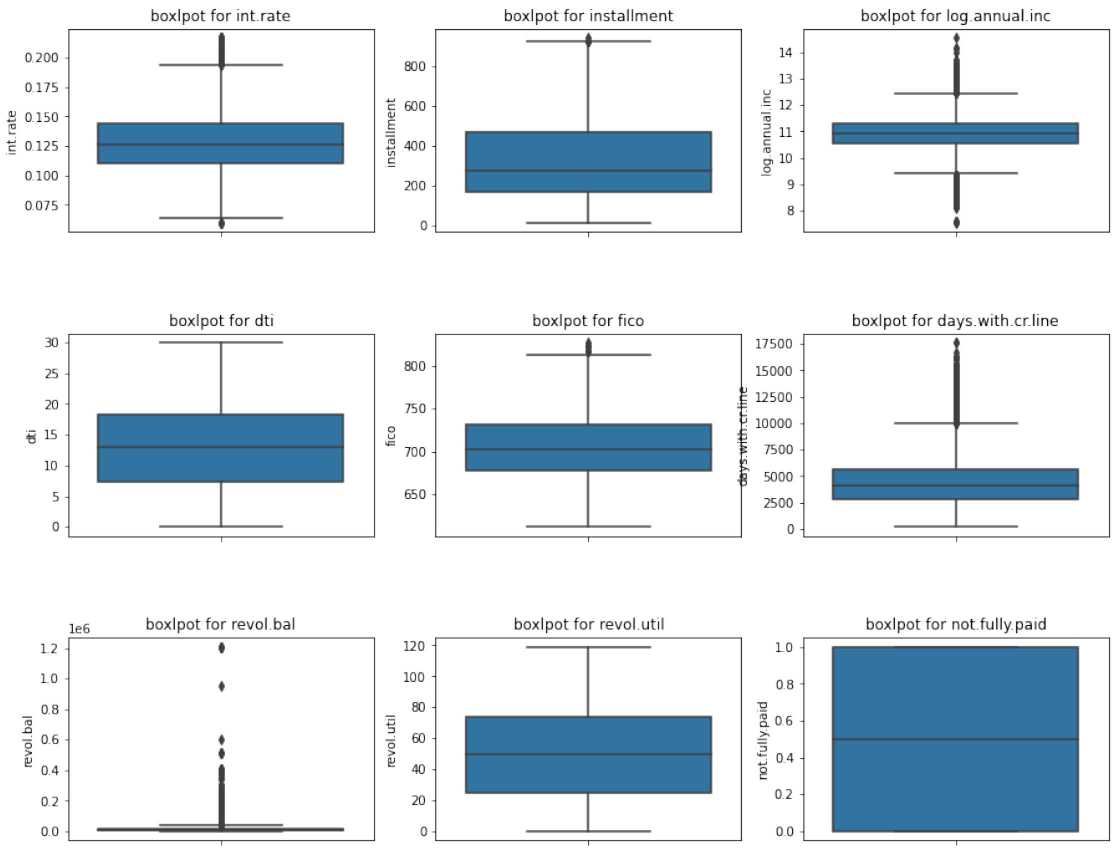
```
num_data.hist(figsize=(30,30),bins=20,legend=True)
plt.show()
```




```
fig,ax= plt.subplots(3,3,figsize=(15,12),sharex=True)
fig.subplots_adjust(hspace=0.5)
for i,cols in enumerate(num_data):
  axx = plt.subplot(3,3,i+1)
  stats.probplot(num_data[cols],plot = axx)
  axx.set_title(f"Probability Plot for {cols}")
plt.show()
```




```
#  it can be seen that the variables such as revol.bal,
days.with.cr.line, installment, fico, and revol.util
# they may contain outliers because the values in these variables do
not fall well around the best fit line.

fig,ax = plt.subplots(3,3,figsize=(15,12),sharex=True)
fig.subplots_adjust(hspace=0.5)

for i,j in enumerate(num_data):
  ax = plt.subplot(3,3,i+1)
  sns.boxplot(y=num_data[j])
  ax.set_title(f'boxlpot for {j}')
plt.show()
```

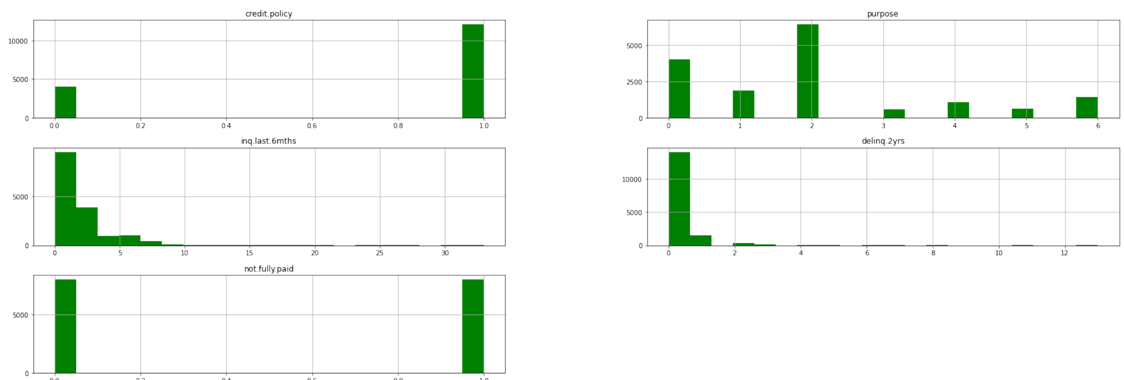



```
# From the graphs above, it can be seen that the outliers exist in the variables
# such as the following: int.rate, installment, log.annual.inc, fico, days.with.cr.line and revol.bal. These outliers will be handles later.

# Converting categorical feature into numerical feature
cat_data = cat_data.copy()
le = preprocessing.LabelEncoder()
cat_data["purpose"] =
le.fit_transform(cat_data["purpose"].astype(str))
cat_data.head()
```

```
      credit.policy  purpose  inq.last.6mths  delinq.2yrs
not.fully.paid
8008              0        6               5            2
1
7050              1        2               0            0
0
6386              1        2               5            0
0
8744              0        1               8            1
1
8357              0        0               5            1
1
```

```
cat_data.describe()

       credit.policy       purpose  inq.last.6mths   delinq.2yrs  \
count   16090.000000  16090.000000    16090.000000  16090.000000
mean        0.747296      2.029770        1.869733      0.170230
std         0.434576      1.788958        2.521743      0.536373
min         0.000000      0.000000        0.000000      0.000000
25%         0.000000      1.000000        0.000000      0.000000
50%         1.000000      2.000000        1.000000      0.000000
75%         1.000000      2.000000        3.000000      0.000000
max         1.000000      6.000000       33.000000     13.000000

       not.fully.paid
count    16090.000000
mean         0.500000
std          0.500016
min          0.000000
25%          0.000000
50%          0.500000
75%          1.000000
max          1.000000
```

```
cat_data.hist(figsize=(30,10),bins=20,color='g')
plt.rcParams['font.size'] = '20'
plt.show()
```




```
# It can be seen that most of the categorical data is positively
skewed.
# Most clients satisfied the credit policy.
# Most clients decided to take the loan for purposes of loan
consolidation

le = preprocessing.LabelEncoder()
df["purpose"] = le.fit_transform(df["purpose"].astype(str))
df.head()

      credit.policy  purpose  int.rate  installment  log.annual.inc
dti  \
8008              0        6    0.1280       503.97       11.002100
```

```
22.44
7050              1        2    0.1311       398.19       11.238489
20.42
6386              1        2    0.1148       478.05       11.277203
6.88
8744              0        1    0.1064       244.27       10.596635
2.07
8357              0        0    0.1324       169.05       10.239960
19.54

      fico  days.with.cr.line  revol.bal  revol.util
inq.last.6mths  \
8008   687        4859.041667         23         1.3               5

7050   692        4267.041667      12764        44.3               0

6386   752        7592.000000      18003        35.0               5

8744   702       11730.000000        599         3.2               8

8357   667        3960.041667      13542        56.4               5


      delinq.2yrs  pub.rec  not.fully.paid
8008            2        0               1
7050            0        1               0
6386            0        0               0
8744            1        1               1
8357            1        0               1
```

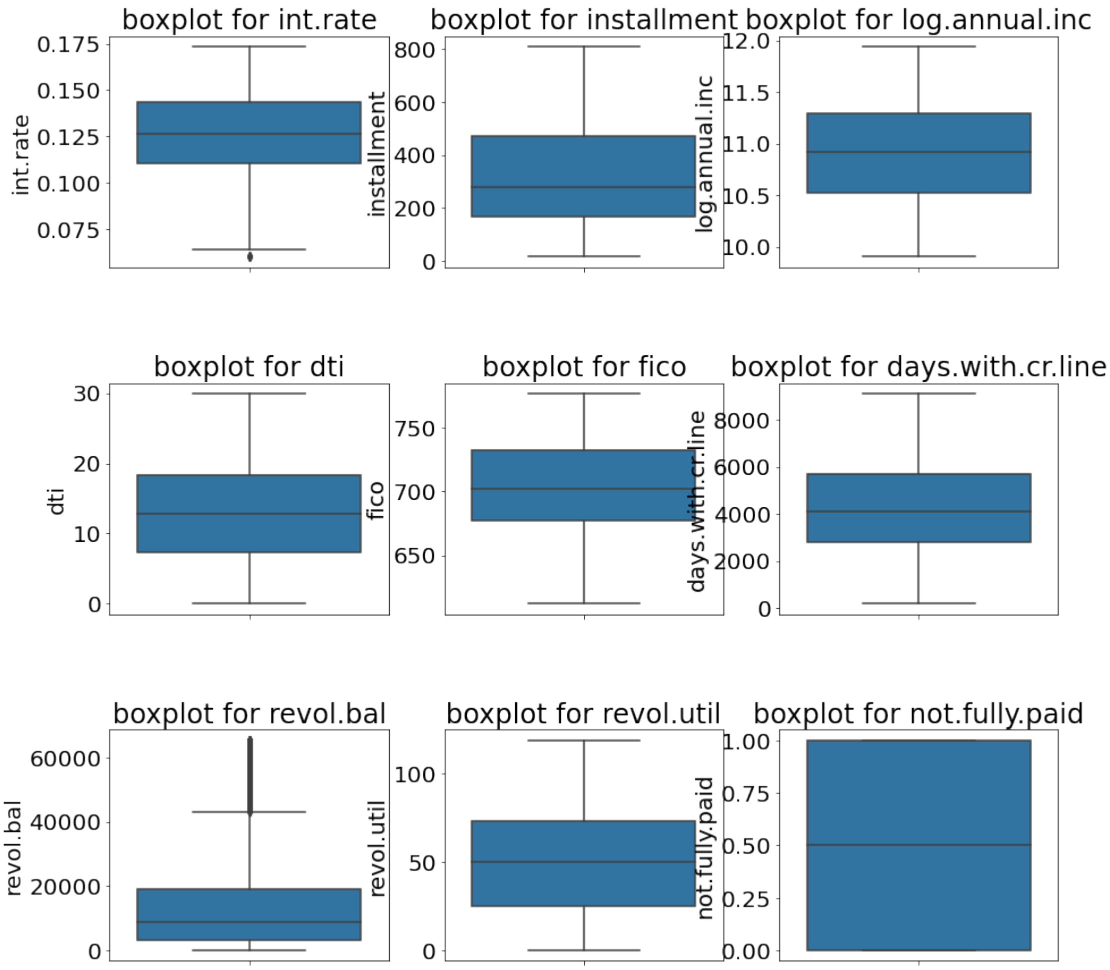
```
df.isnull().sum()
```

```
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
```

```
num_data.describe()
```

```
           int.rate   installment  log.annual.inc          dti
fico  \
count  16090.000000  16090.000000    16090.000000  16090.000000
16090.000000
mean       0.126717    331.233010       10.918552     12.779421
705.695152
std        0.026932    216.046425        0.639732      6.968271
36.941948
min        0.060000     15.670000        7.547502      0.000000
612.000000
25%        0.110300    167.120000       10.524064      7.280000
677.000000
50%        0.126100    277.450000       10.915088     12.890000
702.000000
75%        0.143800    469.780000       11.289832     18.240000
732.000000
max        0.216400    940.140000       14.528354     29.960000
827.000000

       days.with.cr.line     revol.bal    revol.util  not.fully.paid
count       16090.000000  1.609000e+04  16090.000000    16090.000000
mean         4494.871600  1.921108e+04     48.959622        0.500000
std          2474.362329  4.398412e+04     29.142480        0.500016
min           178.958333  0.000000e+00      0.000000        0.000000
25%          2790.000000  3.163250e+03     24.900000        0.000000
50%          4110.000000  8.764000e+03     49.800000        0.500000
75%          5670.041667  1.914400e+04     73.500000        1.000000
max         17639.958330  1.207359e+06    119.000000        1.000000
```

```python
# Upper bounded outliers
for var in ['int.rate' ,'installment','fico', 'days.with.cr.line',
'revol.bal', 'not.fully.paid']:
  df[var].clip(upper = df[var].quantile(0.95),inplace=True)

# Lower and Upper bounded outliers
for var in ['log.annual.inc']:
  df[var] = df[var].clip(upper = df[var].quantile(0.95),lower =
df[var].quantile(0.05))

fig,ax = plt.subplots(3,3,figsize=(15,15))
plt.subplots_adjust(hspace=0.5)

for i,j in enumerate(df[num_data.columns]):
  ax = plt.subplot(3,3,i+1)
  sns.boxplot(y=df[j])
  ax.set_title(f'boxplot for {j}')
plt.show()
```




```python
# Check for skewness in the numerical features
vars_skewed = df[num_data.columns].apply(lambda x:
skew(x)).sort_values(ascending=False)
vars_skewed
```

```
revol.bal            1.709400
installment          0.781616
days.with.cr.line    0.470738
fico                 0.362407
dti                  0.017470
log.annual.inc       0.006915
not.fully.paid       0.000000
revol.util          -0.034960
int.rate            -0.098591
dtype: float64
```

```python
vars_skewed = vars_skewed[vars_skewed>0.3]
vars_skewed
```

```
revol.bal            1.709400
installment          0.781616
days.with.cr.line    0.470738
```

```
fico                   0.362407
dtype: float64
```

```python
for i in  vars_skewed.index:
  df[i] = np.log1p(df[i])

vars_skewed= df[num_data.columns].apply(lambda x:
skew(x)).sort_values(ascending=False)
vars_skewed
```

```
fico                   0.273172
dti                    0.017470
log.annual.inc         0.006915
not.fully.paid         0.000000
revol.util            -0.034960
int.rate              -0.098591
installment           -0.586084
days.with.cr.line     -1.146639
revol.bal             -2.284327
dtype: float64
```

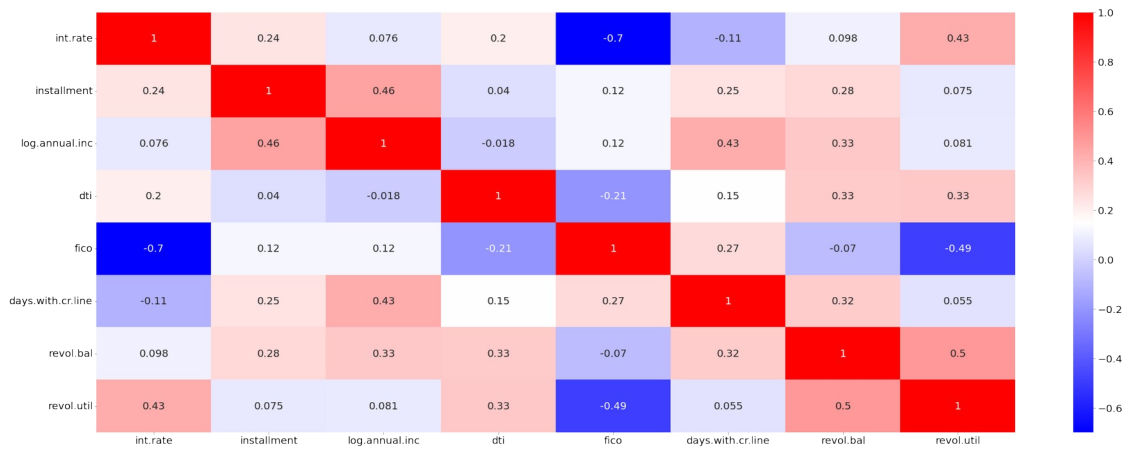
```python
# for i in cat_data.columns:
#   print(df[i].value_counts(),'\n')

x_num = df[num_data.columns.drop('not.fully.paid')]
x_num
```

```
      int.rate  installment  log.annual.inc    dti      fico  \
8008    0.1280     6.224499       11.002100  22.44  6.533789
7050    0.1311     5.989437       11.238489  20.42  6.541030
6386    0.1148     6.171805       11.277203   6.88  6.624065
8744    0.1064     5.502360       10.596635   2.07  6.555357
8357    0.1324     5.136093       10.239960  19.54  6.504288
...        ...          ...             ...    ...       ...
1388    0.1209     6.396329       11.112448  19.22  6.555357
5621    0.1287     5.376481       11.167261  11.09  6.533789
2714    0.1347     6.011463       10.149097   9.20  6.583409
3009    0.1347     5.607235       10.550931  12.59  6.533789
8697    0.1576     4.173618        9.903688  25.07  6.504288

      days.with.cr.line  revol.bal  revol.util
8008           8.488802   3.178054         1.3
7050           8.358910   9.454462        44.3
6386           8.934982   9.798349        35.0
8744           9.118333   6.396930         3.2
8357           8.284262   9.513625        56.4
...                 ...        ...         ...
1388           8.560444  10.038805        64.5
5621           8.804025   9.414750        40.0
```

```
2714           9.118333   9.339437        94.8
3009           8.137968   8.620832        60.9
8697           7.186113   8.596374        35.6

[16090 rows x 8 columns]
```

```
y = df[['not.fully.paid']]
```

```
plt.figure(figsize=(40,15))
sns.heatmap(x_num.corr(),annot=True,cmap='bwr')
plt.show()
```




```
matrix =x_num.corr().unstack()
sorted_pairs = matrix.sort_values()
```

```
sorted_pairs[abs(sorted_pairs)>0.7]
```

```
int.rate           int.rate           1.0
days.with.cr.line  days.with.cr.line  1.0
fico               fico               1.0
dti                dti                1.0
log.annual.inc     log.annual.inc     1.0
installment        installment        1.0
revol.bal          revol.bal          1.0
revol.util         revol.util         1.0
dtype: float64
```

```
from sklearn.svm import SVR
estimator = SVR(kernel='linear')
rfe = RFE(estimator,n_features_to_select=5,step = 1)
rfe= rfe.fit(x_num,y.values)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/
validation.py:993: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
```

```
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993
: DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993
: DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.8/dist-packages/sklearn/utils/validation.py:993
: DataConversionWarning: A column-vector y was passed when a 1d array
was expected. Please change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
```

```
list(zip(x_num.columns ,rfe.support_,rfe.ranking_))
```

```
[('int.rate', True, 1),
 ('installment', True, 1),
 ('log.annual.inc', True, 1),
 ('dti', False, 3),
 ('fico', True, 1),
 ('days.with.cr.line', True, 1),
 ('revol.bal', False, 2),
 ('revol.util', False, 4)]
```

```
selected_cols = x_num.columns[rfe.support_]
selected_cols
```

```
Index(['int.rate', 'installment', 'log.annual.inc', 'fico',
       'days.with.cr.line'],
      dtype='object')
```

```
best_num_cols = df[selected_cols]
```

```
# select best categrical columns
```

```
cat_vars = df[cat_data.columns].drop('not.fully.paid',axis=1)
cat_vars.head()
```

	credit.policy	purpose	inq.last.6mths	delinq.2yrs
8008	0	6	5	2
7050	1	2	0	0
6386	1	2	5	0
8744	0	1	8	1
8357	0	0	5	1

```
f_p_values = chi2(cat_vars,df['not.fully.paid'])
f_p_values
```

```
(array([ 155.64071856,  116.07241495, 1634.36191996,    7.05403432]),
 array([1.01432920e-35, 4.58254427e-27, 0.00000000e+00, 7.90869323e-
03]))
```

```
# wseeing what value belongs to which feature
pd.Series(f_p_values[1],cat_vars.columns).sort_values(ascending=True)
```

```
inq.last.6mths    0.000000e+00
credit.policy     1.014329e-35
purpose           4.582544e-27
delinq.2yrs       7.908693e-03
dtype: float64
```

```
# all four categorical columns have p value less than 0.05 . so
consider all of them
```

```
x = df[['int.rate', 'installment', 'log.annual.inc', 'dti', 'fico',
'inq.last.6mths','credit.policy', 'purpose','delinq.2yrs']]
x.head()
```

```
      int.rate  installment  log.annual.inc    dti      fico
inq.last.6mths  \
8008    0.1280     6.224499       11.002100  22.44  6.533789
5
7050    0.1311     5.989437       11.238489  20.42  6.541030
0
6386    0.1148     6.171805       11.277203   6.88  6.624065
5
8744    0.1064     5.502360       10.596635   2.07  6.555357
8
8357    0.1324     5.136093       10.239960  19.54  6.504288
5


      credit.policy  purpose  delinq.2yrs
8008              0        6            2
7050              1        2            0
6386              1        2            0
8744              0        1            1
8357              0        0            1
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
0.2, random_state = 42)
```

```
# Scale the data
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
model  = keras.Sequential(
    [
        keras.layers.Dense(
        256, activation="relu", input_shape=[9]),
```

```
        keras.layers.Dense(256, activation="relu"),
        keras.layers.Dropout(0.3),
        keras.layers.Dense(256, activation="relu"),
        keras.layers.Dropout(0.3),
        keras.layers.Dense(1, activation="sigmoid"),
    ]
)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 256)               2560

 dense_5 (Dense)             (None, 256)               65792

 dropout_2 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 256)               65792

 dropout_3 (Dropout)         (None, 256)               0

 dense_7 (Dense)             (None, 1)                 257

=================================================================
Total params: 134,401
Trainable params: 134,401
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer = 'Adam', loss = 'binary_crossentropy',
metrics = ['binary_accuracy'])

early_stopping = keras.callbacks.EarlyStopping(patience=10,
min_delta=0.001, restore_best_weights=True)
history = model.fit(
    x_train, y_train,
    validation_data=(x_test, y_test),
    batch_size=256,
    epochs=100,
    callbacks=[early_stopping],
    verbose=1,
)
```

```
Epoch 1/100
51/51 [==============================] - 1s 21ms/step - loss: 0.4797 -
binary_accuracy: 0.7547 - val_loss: 0.5214 - val_binary_accuracy:
0.7390
Epoch 2/100
51/51 [==============================] - 1s 14ms/step - loss: 0.4791 -
```

```
binary_accuracy: 0.7571 - val_loss: 0.5165 - val_binary_accuracy:
0.7362
Epoch 3/100
51/51 [==============================] - 1s 13ms/step - loss: 0.4688 -
binary_accuracy: 0.7609 - val_loss: 0.5187 - val_binary_accuracy:
0.7523
Epoch 4/100
51/51 [==============================] - 1s 12ms/step - loss: 0.4743 -
binary_accuracy: 0.7557 - val_loss: 0.5071 - val_binary_accuracy:
0.7523
Epoch 5/100
51/51 [==============================] - 1s 12ms/step - loss: 0.4654 -
binary_accuracy: 0.7641 - val_loss: 0.5140 - val_binary_accuracy:
0.7464
Epoch 6/100
51/51 [==============================] - 1s 13ms/step - loss: 0.4545 -
binary_accuracy: 0.7702 - val_loss: 0.5073 - val_binary_accuracy:
0.7483
Epoch 7/100
51/51 [==============================] - 1s 13ms/step - loss: 0.4581 -
binary_accuracy: 0.7651 - val_loss: 0.5123 - val_binary_accuracy:
0.7495
Epoch 8/100
51/51 [==============================] - 1s 13ms/step - loss: 0.4529 -
binary_accuracy: 0.7777 - val_loss: 0.4994 - val_binary_accuracy:
0.7604
Epoch 9/100
51/51 [==============================] - 1s 13ms/step - loss: 0.4440 -
binary_accuracy: 0.7777 - val_loss: 0.5053 - val_binary_accuracy:
0.7592
Epoch 10/100
51/51 [==============================] - 1s 13ms/step - loss: 0.4475 -
binary_accuracy: 0.7770 - val_loss: 0.4913 - val_binary_accuracy:
0.7548
Epoch 11/100
51/51 [==============================] - 1s 13ms/step - loss: 0.4470 -
binary_accuracy: 0.7822 - val_loss: 0.5132 - val_binary_accuracy:
0.7557
Epoch 12/100
51/51 [==============================] - 1s 12ms/step - loss: 0.4378 -
binary_accuracy: 0.7825 - val_loss: 0.5008 - val_binary_accuracy:
0.7632
Epoch 13/100
51/51 [==============================] - 1s 19ms/step - loss: 0.4309 -
binary_accuracy: 0.7885 - val_loss: 0.5036 - val_binary_accuracy:
0.7601
Epoch 14/100
51/51 [==============================] - 2s 37ms/step - loss: 0.4396 -
binary_accuracy: 0.7804 - val_loss: 0.4956 - val_binary_accuracy:
0.7713
```

```
Epoch 15/100
51/51 [==============================] - 1s 22ms/step - loss: 0.4283 -
binary_accuracy: 0.7894 - val_loss: 0.4867 - val_binary_accuracy:
0.7623
Epoch 16/100
51/51 [==============================] - 1s 25ms/step - loss: 0.4269 -
binary_accuracy: 0.7920 - val_loss: 0.4931 - val_binary_accuracy:
0.7707
Epoch 17/100
51/51 [==============================] - 2s 30ms/step - loss: 0.4222 -
binary_accuracy: 0.7930 - val_loss: 0.4842 - val_binary_accuracy:
0.7672
Epoch 18/100
51/51 [==============================] - 1s 20ms/step - loss: 0.4153 -
binary_accuracy: 0.7958 - val_loss: 0.5007 - val_binary_accuracy:
0.7784
Epoch 19/100
51/51 [==============================] - 1s 15ms/step - loss: 0.4152 -
binary_accuracy: 0.7960 - val_loss: 0.4847 - val_binary_accuracy:
0.7787
Epoch 20/100
51/51 [==============================] - 1s 16ms/step - loss: 0.4123 -
binary_accuracy: 0.7997 - val_loss: 0.4902 - val_binary_accuracy:
0.7660
Epoch 21/100
51/51 [==============================] - 1s 27ms/step - loss: 0.4096 -
binary_accuracy: 0.7995 - val_loss: 0.4814 - val_binary_accuracy:
0.7775
Epoch 22/100
51/51 [==============================] - 1s 28ms/step - loss: 0.4090 -
binary_accuracy: 0.8025 - val_loss: 0.4872 - val_binary_accuracy:
0.7759
Epoch 23/100
51/51 [==============================] - 1s 28ms/step - loss: 0.4083 -
binary_accuracy: 0.7990 - val_loss: 0.4944 - val_binary_accuracy:
0.7797
Epoch 24/100
51/51 [==============================] - 2s 42ms/step - loss: 0.4048 -
binary_accuracy: 0.8034 - val_loss: 0.4883 - val_binary_accuracy:
0.7747
Epoch 25/100
51/51 [==============================] - 1s 28ms/step - loss: 0.3976 -
binary_accuracy: 0.8083 - val_loss: 0.4966 - val_binary_accuracy:
0.7800
Epoch 26/100
51/51 [==============================] - 1s 15ms/step - loss: 0.3977 -
binary_accuracy: 0.8082 - val_loss: 0.4838 - val_binary_accuracy:
0.7871
Epoch 27/100
51/51 [==============================] - 1s 20ms/step - loss: 0.3960 -
```

```
binary_accuracy: 0.8063 - val_loss: 0.4694 - val_binary_accuracy:
0.7915
Epoch 28/100
51/51 [==============================] - 1s 18ms/step - loss: 0.3939 -
binary_accuracy: 0.8117 - val_loss: 0.4900 - val_binary_accuracy:
0.7778
Epoch 29/100
51/51 [==============================] - 1s 24ms/step - loss: 0.3916 -
binary_accuracy: 0.8101 - val_loss: 0.4831 - val_binary_accuracy:
0.7843
Epoch 30/100
51/51 [==============================] - 1s 17ms/step - loss: 0.3858 -
binary_accuracy: 0.8159 - val_loss: 0.4820 - val_binary_accuracy:
0.7915
Epoch 31/100
51/51 [==============================] - 1s 17ms/step - loss: 0.3834 -
binary_accuracy: 0.8188 - val_loss: 0.4752 - val_binary_accuracy:
0.7955
Epoch 32/100
51/51 [==============================] - 1s 18ms/step - loss: 0.3818 -
binary_accuracy: 0.8167 - val_loss: 0.4830 - val_binary_accuracy:
0.7819
Epoch 33/100
51/51 [==============================] - 1s 15ms/step - loss: 0.3853 -
binary_accuracy: 0.8181 - val_loss: 0.4826 - val_binary_accuracy:
0.7899
Epoch 34/100
51/51 [==============================] - 1s 16ms/step - loss: 0.3746 -
binary_accuracy: 0.8252 - val_loss: 0.4737 - val_binary_accuracy:
0.7884
Epoch 35/100
51/51 [==============================] - 1s 18ms/step - loss: 0.3761 -
binary_accuracy: 0.8222 - val_loss: 0.4975 - val_binary_accuracy:
0.7825
Epoch 36/100
51/51 [==============================] - 1s 17ms/step - loss: 0.3829 -
binary_accuracy: 0.8176 - val_loss: 0.4798 - val_binary_accuracy:
0.7946
Epoch 37/100
51/51 [==============================] - 1s 23ms/step - loss: 0.3764 -
binary_accuracy: 0.8198 - val_loss: 0.4674 - val_binary_accuracy:
0.7993
Epoch 38/100
51/51 [==============================] - 1s 27ms/step - loss: 0.3679 -
binary_accuracy: 0.8276 - val_loss: 0.4702 - val_binary_accuracy:
0.7965
Epoch 39/100
51/51 [==============================] - 1s 21ms/step - loss: 0.3648 -
binary_accuracy: 0.8268 - val_loss: 0.4630 - val_binary_accuracy:
0.8048
```

```
Epoch 40/100
51/51 [==============================] - 1s 21ms/step - loss: 0.3588 -
binary_accuracy: 0.8341 - val_loss: 0.4675 - val_binary_accuracy:
0.7983
Epoch 41/100
51/51 [==============================] - 1s 17ms/step - loss: 0.3657 -
binary_accuracy: 0.8281 - val_loss: 0.4570 - val_binary_accuracy:
0.8014
Epoch 42/100
51/51 [==============================] - 1s 15ms/step - loss: 0.3589 -
binary_accuracy: 0.8327 - val_loss: 0.4610 - val_binary_accuracy:
0.8061
Epoch 43/100
51/51 [==============================] - 1s 21ms/step - loss: 0.3554 -
binary_accuracy: 0.8348 - val_loss: 0.4743 - val_binary_accuracy:
0.8014
Epoch 44/100
51/51 [==============================] - 1s 27ms/step - loss: 0.3567 -
binary_accuracy: 0.8334 - val_loss: 0.4728 - val_binary_accuracy:
0.8033
Epoch 45/100
51/51 [==============================] - 1s 17ms/step - loss: 0.3512 -
binary_accuracy: 0.8335 - val_loss: 0.4707 - val_binary_accuracy:
0.8002
Epoch 46/100
51/51 [==============================] - 1s 19ms/step - loss: 0.3525 -
binary_accuracy: 0.8353 - val_loss: 0.4581 - val_binary_accuracy:
0.8086
Epoch 47/100
51/51 [==============================] - 1s 22ms/step - loss: 0.3504 -
binary_accuracy: 0.8378 - val_loss: 0.4679 - val_binary_accuracy:
0.8123
Epoch 48/100
51/51 [==============================] - 1s 16ms/step - loss: 0.3463 -
binary_accuracy: 0.8376 - val_loss: 0.4731 - val_binary_accuracy:
0.8073
Epoch 49/100
51/51 [==============================] - 1s 17ms/step - loss: 0.3518 -
binary_accuracy: 0.8354 - val_loss: 0.4664 - val_binary_accuracy:
0.8129
Epoch 50/100
51/51 [==============================] - 1s 25ms/step - loss: 0.3402 -
binary_accuracy: 0.8398 - val_loss: 0.4580 - val_binary_accuracy:
0.8173
Epoch 51/100
51/51 [==============================] - 1s 26ms/step - loss: 0.3408 -
binary_accuracy: 0.8419 - val_loss: 0.4532 - val_binary_accuracy:
0.8204
Epoch 52/100
51/51 [==============================] - 1s 24ms/step - loss: 0.3413 -
```

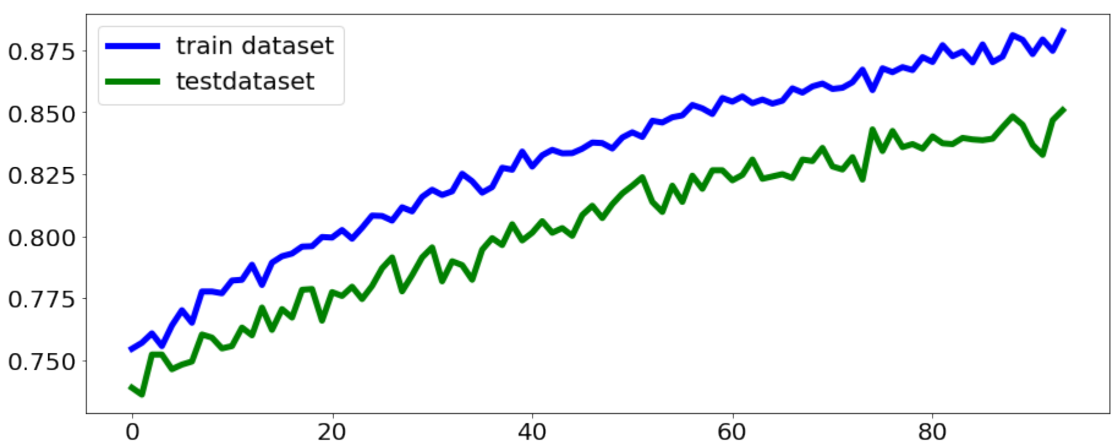
```
binary_accuracy: 0.8400 - val_loss: 0.4548 - val_binary_accuracy:
0.8238
Epoch 53/100
51/51 [==============================] - 1s 24ms/step - loss: 0.3368 -
binary_accuracy: 0.8466 - val_loss: 0.4712 - val_binary_accuracy:
0.8139
Epoch 54/100
51/51 [==============================] - 1s 25ms/step - loss: 0.3348 -
binary_accuracy: 0.8459 - val_loss: 0.4614 - val_binary_accuracy:
0.8098
Epoch 55/100
51/51 [==============================] - 1s 22ms/step - loss: 0.3315 -
binary_accuracy: 0.8480 - val_loss: 0.4558 - val_binary_accuracy:
0.8204
Epoch 56/100
51/51 [==============================] - 1s 19ms/step - loss: 0.3277 -
binary_accuracy: 0.8487 - val_loss: 0.4618 - val_binary_accuracy:
0.8139
Epoch 57/100
51/51 [==============================] - 1s 12ms/step - loss: 0.3249 -
binary_accuracy: 0.8529 - val_loss: 0.4450 - val_binary_accuracy:
0.8244
Epoch 58/100
51/51 [==============================] - 1s 12ms/step - loss: 0.3213 -
binary_accuracy: 0.8515 - val_loss: 0.4637 - val_binary_accuracy:
0.8191
Epoch 59/100
51/51 [==============================] - 1s 13ms/step - loss: 0.3269 -
binary_accuracy: 0.8493 - val_loss: 0.4624 - val_binary_accuracy:
0.8266
Epoch 60/100
51/51 [==============================] - 1s 12ms/step - loss: 0.3141 -
binary_accuracy: 0.8557 - val_loss: 0.4585 - val_binary_accuracy:
0.8266
Epoch 61/100
51/51 [==============================] - 1s 12ms/step - loss: 0.3224 -
binary_accuracy: 0.8543 - val_loss: 0.4775 - val_binary_accuracy:
0.8226
Epoch 62/100
51/51 [==============================] - 1s 13ms/step - loss: 0.3195 -
binary_accuracy: 0.8564 - val_loss: 0.4435 - val_binary_accuracy:
0.8247
Epoch 63/100
51/51 [==============================] - 1s 13ms/step - loss: 0.3206 -
binary_accuracy: 0.8536 - val_loss: 0.4523 - val_binary_accuracy:
0.8310
Epoch 64/100
51/51 [==============================] - 1s 20ms/step - loss: 0.3185 -
binary_accuracy: 0.8551 - val_loss: 0.4626 - val_binary_accuracy:
0.8232
```

```
Epoch 65/100
51/51 [==============================] - 1s 29ms/step - loss: 0.3177 -
binary_accuracy: 0.8535 - val_loss: 0.4614 - val_binary_accuracy:
0.8241
Epoch 66/100
51/51 [==============================] - 1s 27ms/step - loss: 0.3107 -
binary_accuracy: 0.8546 - val_loss: 0.4572 - val_binary_accuracy:
0.8250
Epoch 67/100
51/51 [==============================] - 1s 17ms/step - loss: 0.3166 -
binary_accuracy: 0.8596 - val_loss: 0.4464 - val_binary_accuracy:
0.8235
Epoch 68/100
51/51 [==============================] - 1s 12ms/step - loss: 0.3145 -
binary_accuracy: 0.8579 - val_loss: 0.4441 - val_binary_accuracy:
0.8310
Epoch 69/100
51/51 [==============================] - 1s 12ms/step - loss: 0.3075 -
binary_accuracy: 0.8604 - val_loss: 0.4414 - val_binary_accuracy:
0.8303
Epoch 70/100
51/51 [==============================] - 1s 13ms/step - loss: 0.3054 -
binary_accuracy: 0.8616 - val_loss: 0.4483 - val_binary_accuracy:
0.8356
Epoch 71/100
51/51 [==============================] - 1s 13ms/step - loss: 0.3121 -
binary_accuracy: 0.8594 - val_loss: 0.4595 - val_binary_accuracy:
0.8282
Epoch 72/100
51/51 [==============================] - 1s 13ms/step - loss: 0.3124 -
binary_accuracy: 0.8599 - val_loss: 0.4484 - val_binary_accuracy:
0.8269
Epoch 73/100
51/51 [==============================] - 1s 12ms/step - loss: 0.3036 -
binary_accuracy: 0.8621 - val_loss: 0.4516 - val_binary_accuracy:
0.8319
Epoch 74/100
51/51 [==============================] - 1s 13ms/step - loss: 0.2926 -
binary_accuracy: 0.8672 - val_loss: 0.4719 - val_binary_accuracy:
0.8229
Epoch 75/100
51/51 [==============================] - 1s 13ms/step - loss: 0.3078 -
binary_accuracy: 0.8589 - val_loss: 0.4390 - val_binary_accuracy:
0.8431
Epoch 76/100
51/51 [==============================] - 1s 13ms/step - loss: 0.3013 -
binary_accuracy: 0.8677 - val_loss: 0.4459 - val_binary_accuracy:
0.8344
Epoch 77/100
51/51 [==============================] - 1s 12ms/step - loss: 0.2995 -
```

```
binary_accuracy: 0.8661 - val_loss: 0.4467 - val_binary_accuracy:
0.8424
Epoch 78/100
51/51 [==============================] - 1s 13ms/step - loss: 0.2962 -
binary_accuracy: 0.8682 - val_loss: 0.4447 - val_binary_accuracy:
0.8359
Epoch 79/100
51/51 [==============================] - 1s 12ms/step - loss: 0.2970 -
binary_accuracy: 0.8670 - val_loss: 0.4522 - val_binary_accuracy:
0.8372
Epoch 80/100
51/51 [==============================] - 1s 12ms/step - loss: 0.2921 -
binary_accuracy: 0.8722 - val_loss: 0.4510 - val_binary_accuracy:
0.8353
Epoch 81/100
51/51 [==============================] - 1s 12ms/step - loss: 0.2912 -
binary_accuracy: 0.8703 - val_loss: 0.4361 - val_binary_accuracy:
0.8403
Epoch 82/100
51/51 [==============================] - 1s 19ms/step - loss: 0.2828 -
binary_accuracy: 0.8770 - val_loss: 0.4403 - val_binary_accuracy:
0.8375
Epoch 83/100
51/51 [==============================] - 1s 19ms/step - loss: 0.2873 -
binary_accuracy: 0.8726 - val_loss: 0.4479 - val_binary_accuracy:
0.8372
Epoch 84/100
51/51 [==============================] - 1s 20ms/step - loss: 0.2888 -
binary_accuracy: 0.8745 - val_loss: 0.4321 - val_binary_accuracy:
0.8397
Epoch 85/100
51/51 [==============================] - 1s 14ms/step - loss: 0.2878 -
binary_accuracy: 0.8701 - val_loss: 0.4457 - val_binary_accuracy:
0.8390
Epoch 86/100
51/51 [==============================] - 1s 13ms/step - loss: 0.2810 -
binary_accuracy: 0.8773 - val_loss: 0.4603 - val_binary_accuracy:
0.8387
Epoch 87/100
51/51 [==============================] - 1s 13ms/step - loss: 0.2886 -
binary_accuracy: 0.8702 - val_loss: 0.4653 - val_binary_accuracy:
0.8393
Epoch 88/100
51/51 [==============================] - 1s 13ms/step - loss: 0.2878 -
binary_accuracy: 0.8724 - val_loss: 0.4552 - val_binary_accuracy:
0.8440
Epoch 89/100
51/51 [==============================] - 1s 13ms/step - loss: 0.2735 -
binary_accuracy: 0.8811 - val_loss: 0.4572 - val_binary_accuracy:
0.8484
```

```
Epoch 90/100
51/51 [==============================] - 1s 13ms/step - loss: 0.2729 - binary_accuracy: 0.8792 - val_loss: 0.4317 - val_binary_accuracy: 0.8449
Epoch 91/100
51/51 [==============================] - 1s 13ms/step - loss: 0.2821 - binary_accuracy: 0.8734 - val_loss: 0.4666 - val_binary_accuracy: 0.8369
Epoch 92/100
51/51 [==============================] - 1s 12ms/step - loss: 0.2735 - binary_accuracy: 0.8794 - val_loss: 0.4583 - val_binary_accuracy: 0.8328
Epoch 93/100
51/51 [==============================] - 1s 12ms/step - loss: 0.2807 - binary_accuracy: 0.8748 - val_loss: 0.4468 - val_binary_accuracy: 0.8468
Epoch 94/100
51/51 [==============================] - 1s 12ms/step - loss: 0.2718 - binary_accuracy: 0.8826 - val_loss: 0.4351 - val_binary_accuracy: 0.8508
```

```python
plt.figure(figsize=(15,6))
plt.plot(history.history['binary_accuracy'],color='b',label = 'train dataset',linewidth=5)
plt.plot(history.history['val_binary_accuracy'],color='g',label='testdataset',linewidth=5)
plt.legend()
plt.show()
```



got accuracy around 88% on train set and 85% on test set