Lending Club loan Data Analysis Project

July 12, 2023

Lending Club Loan Data Analysis Course-end Project 2

DESCRIPTION

Create a model that predicts whether or not a loan will be default using the historical data.

Problem Statement:

For companies like Lending Club correctly predicting whether or not a loan will be a default is very important. In this project, using the historical data from 2007 to 2015, you have to build a deep learning model to predict the chance of default for future loans. As you will see later this dataset is highly imbalanced and includes a lot of features that make this problem more challenging.

IMPORT RELEVANT LIBRARIES

[2]:
```
# Data loading and data management
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, norm

# Data wrangling and feature engineering
from sklearn.impute import SimpleImputer
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 OrdinalEncoder

# 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
import keras_tuner as kt
```

DATA LOADING AND INSPECTION

[3]:
```
import os
os.listdir('./')

data = pd.read_csv("loan_data.csv")
```

[4]:
```
# Check the first five variables of the dataframe
data.head()
```

[4]:
```
   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
```

[5]:
```
# Check the size of the dataframe
data.shape
```

[5]:
```
(9578, 14)
```

[6]:
```
# See the data types of variables in the dataframe
data.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
```

```
Feature Transformation
```

Transform categorical values into numerical values (discrete)

[7]:
```python
# Check the target variable
data['not.fully.paid'].value_counts()
```

[7]:
```
0    8045
1    1533
Name: not.fully.paid, dtype: int64
```

[8]:
```python
#handling imbalanced dataset
not_fully_paid_0 = data[data['not.fully.paid'] == 0]
not_fully_paid_1 = data[data['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, 14)
not_fully_paid_1 (1533, 14)
```

[9]:
```python
#handling imbalanced data
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)
```

[10]:
```
#imbalanced data handled
df['not.fully.paid'].value_counts()
```

[10]:
```
1    8045
0    8045
Name: not.fully.paid, dtype: int64
```

[11]:
```
# 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"]]
num_data
```

[11]:
```
      int.rate  installment  log.annual.inc    dti  fico  days.with.cr.line  \
9155    0.1979       296.46       11.350407   9.94   692        1800.000000
1379    0.0938       383.73       11.813030   9.91   767       13620.000000
6186    0.0894       111.21       10.778956  15.80   722        3870.041667
7941    0.1565       293.87       10.491274  21.50   642        1830.000000
5126    0.1461       137.91       10.915016   3.16   667        7410.000000
…         …            …              …     …     …                 …
1916    0.1253       127.18       11.002100  20.14   687        4140.041667
8081    0.1122       229.91       11.589887  21.97   692        6629.041667
8587    0.1482        74.35       10.463103  13.68   642        3900.000000
4666    0.0894       327.25       10.304141  19.29   747        8070.000000
7964    0.1312       168.76        9.615805   8.72   647        1530.041667

      revol.bal  revol.util  not.fully.paid
9155       7491        29.0               1
1379       1579         7.2               0
6186      10659        76.4               1
7941       3896        34.2               1
5126       3864        52.9               1
…           …          …              …
1916      27759        71.7               1
8081       7258        42.7               1
8587       4207        79.4               1
4666       5822        64.7               1
7964       5183        46.3               1

[16090 rows x 9 columns]
```

[12]:
```
# Check the features in the numerical data
num_data_features = num_data.columns
num_data_features
```

[12]:
```
Index(['int.rate', 'installment', 'log.annual.inc', 'dti', 'fico',
       'days.with.cr.line', 'revol.bal', 'revol.util', 'not.fully.paid'],
```

```
        dtype='object')
```

[13]:
```
# Separate data to include categorical data only
cat_data = df[["credit.policy", "purpose", "inq.last.6mths", "delinq.2yrs",␣
 ↪"not.fully.paid"]]
cat_data
```

[13]:
```
      credit.policy             purpose  inq.last.6mths  delinq.2yrs  \
9155              0      small_business               6            0
1379              1      small_business               0            0
6186              1           all_other               0            0
7941              0           all_other               3            0
5126              1         educational               2            0
…               …                 …              …            …
1916              1           all_other               1            0
8081              0           all_other               7            0
8587              0  debt_consolidation               4            0
4666              1  debt_consolidation               1            0
7964              0  debt_consolidation               1            1

      not.fully.paid
9155               1
1379               0
6186               1
7941               1
5126               1
…                …
1916               1
8081               1
8587               1
4666               1
7964               1

[16090 rows x 5 columns]
```

[14]:
```
# Check the features in the numerical data
cat_data_features = cat_data.columns
cat_data_features
```

[14]:
```
Index(['credit.policy', 'purpose', 'inq.last.6mths', 'delinq.2yrs',
       'not.fully.paid'],
      dtype='object')
```

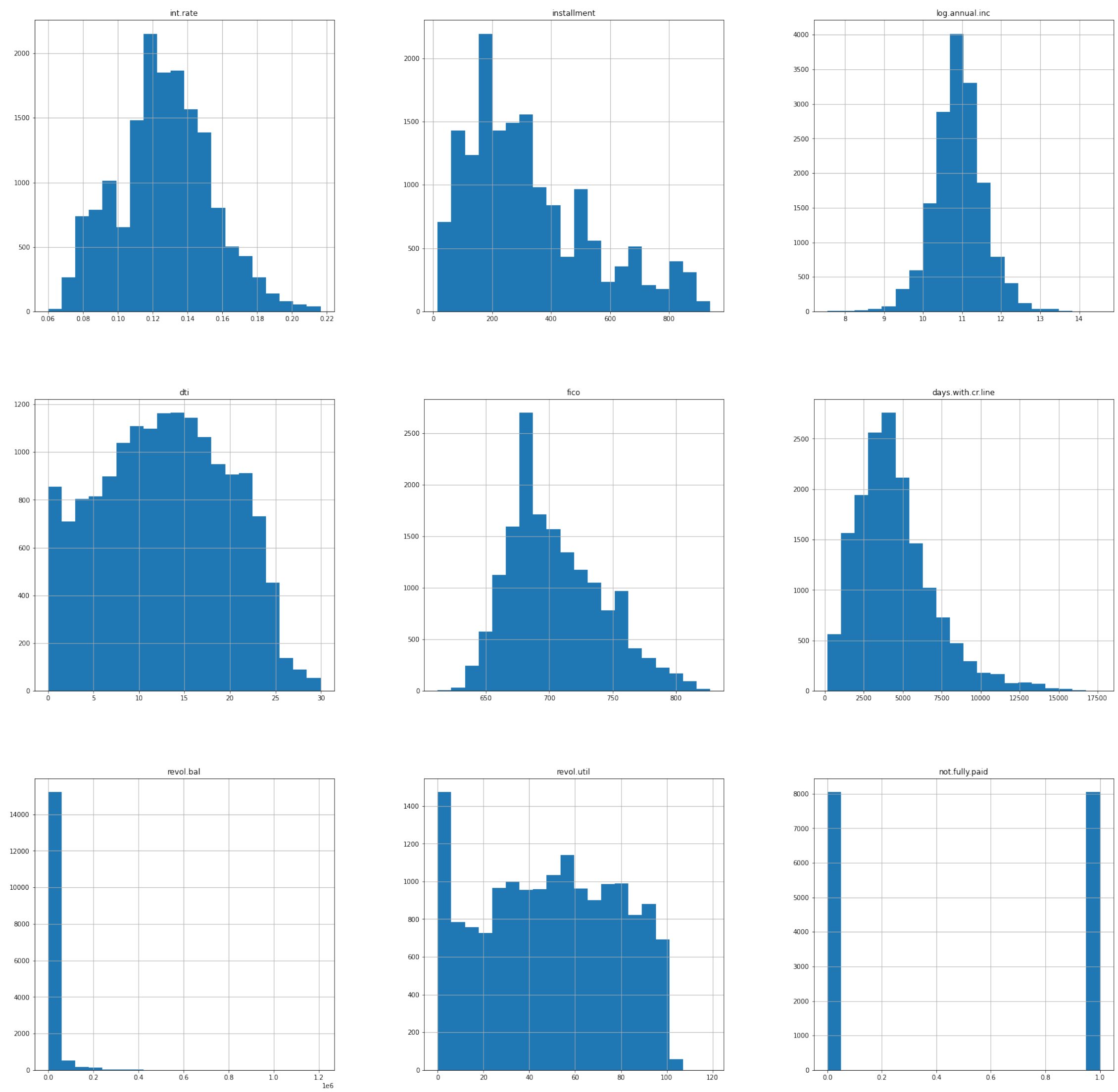
Exploratory data analysis of different factors of the dataset.

DATA EXPLORATION

[15]:
```
# Check the statistics of the numerical data
num_data.describe()
```

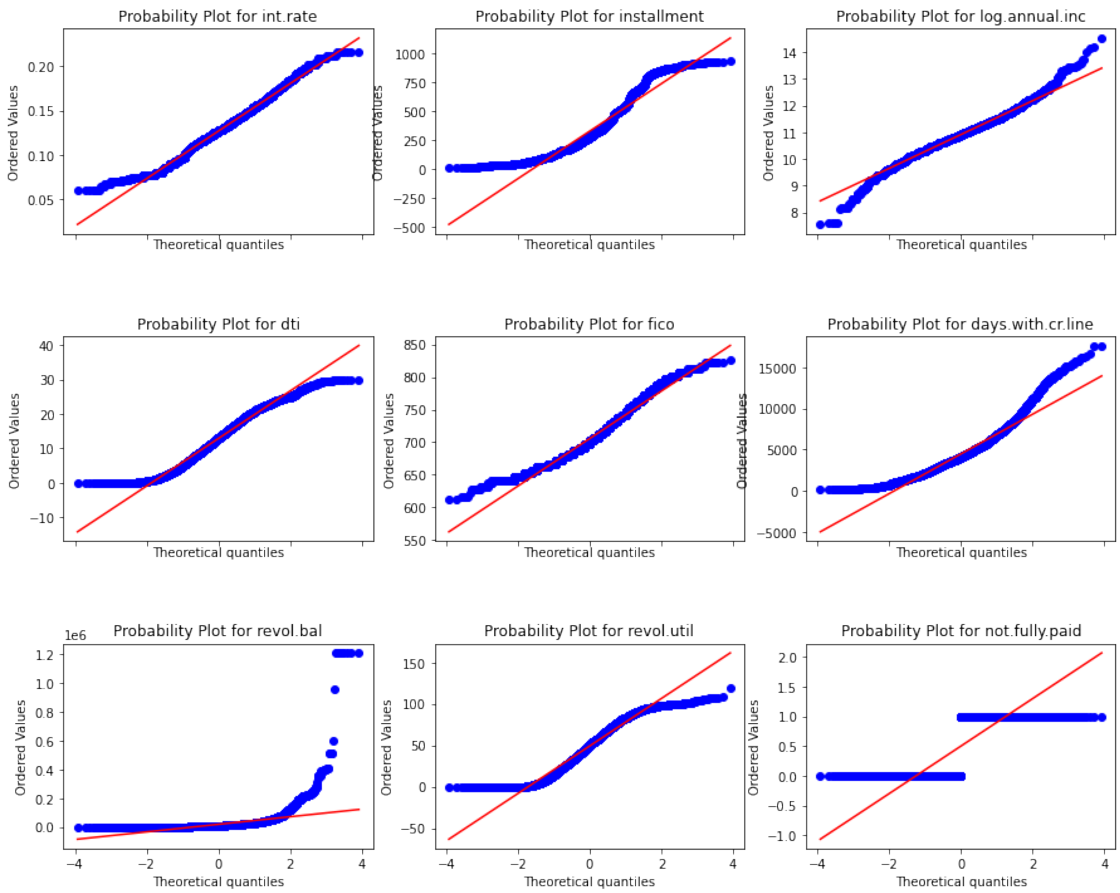
[15]:
```
            int.rate   installment  log.annual.inc           dti          fico  \
count   16090.000000  16090.000000    16090.000000  16090.000000  16090.000000
mean        0.126771    327.576484       10.914929     12.831789    705.463331
std         0.026815    213.318191        0.636466      6.953046     36.905042
min         0.060000     15.670000        7.547502      0.000000    612.000000
25%         0.110300    166.000000       10.518889      7.370000    677.000000
50%         0.126100    276.220000       10.915088     12.950000    702.000000
75%         0.144200    458.460000       11.289782     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         4502.339320  1.923788e+04     49.175818        0.500000
std          2485.443570  4.567750e+04     29.222996        0.500016
min           178.958333  0.000000e+00      0.000000        0.000000
25%          2789.958333  3.084000e+03     25.300000        0.000000
50%          4109.958333  8.748000e+03     50.300000        0.500000
75%          5691.041667  1.942475e+04     73.700000        1.000000
max         17639.958330  1.207359e+06    119.000000        1.000000
```

[16]:
```
# Check the distribution of the numerical continous data
num_data.hist(figsize = (30, 30), bins = 20, legend = False)
plt.show()
```

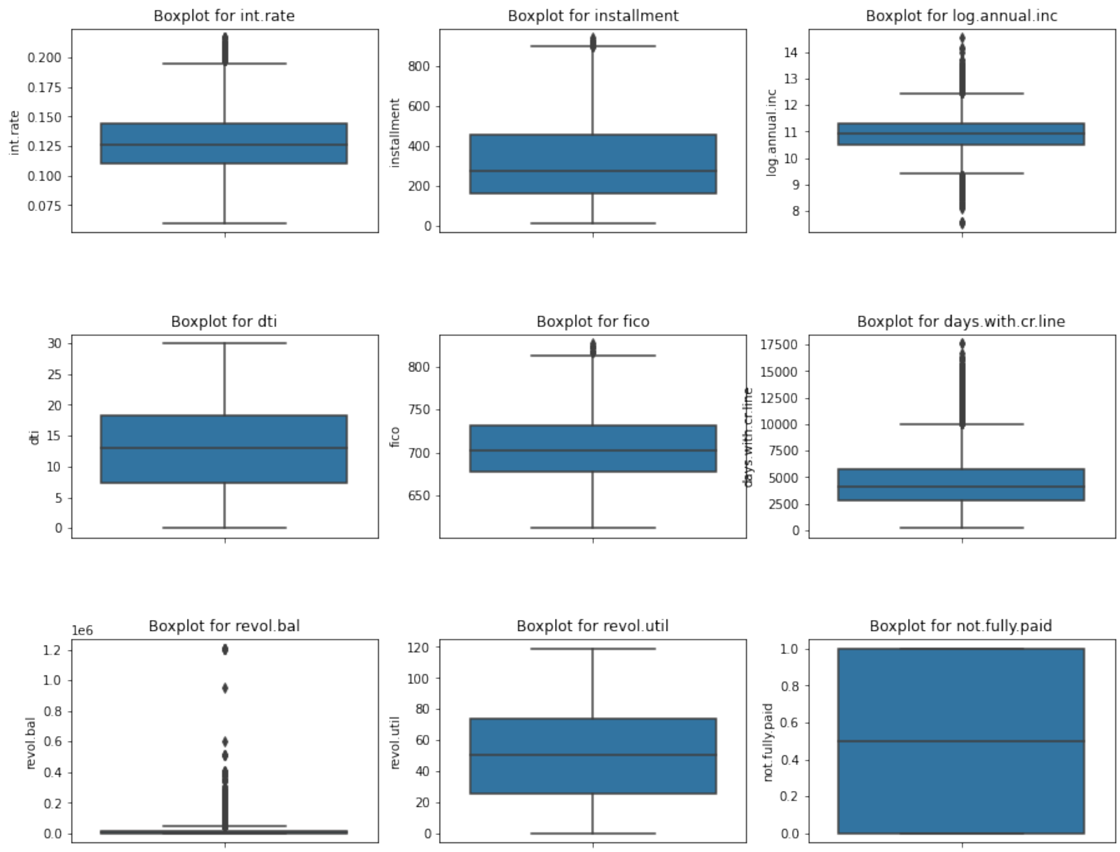


[17]:

```python
# Develop a probability plot to find out how compacted or degress of normalisation the data is.

# Define subplot grid
fig, axs = plt.subplots(nrows = 3, ncols = 3, figsize = (15, 12), sharex = True)
fig.subplots_adjust(hspace = 0.5)


for i, col in enumerate(num_data):
    ax = plt.subplot(3, 3, i+1)
    stats.probplot(num_data[col], plot = ax)
    ax.set_title(f"Probability Plot for {col}")
```



[18]:
```python
# Create plots showing the uncertainity in the data and the outliers.

# Define subplot grid
fig, axs = plt.subplots(nrows = 3, ncols = 3, figsize = (15, 12), sharex = True)
fig.subplots_adjust(hspace = 0.5)


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



Below is an analysis of the categorical data

[19]:
```python
# 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()
```

[19]:

	credit.policy	purpose	inq.last.6mths	delinq.2yrs	not.fully.paid
9155	0	6	6	0	1
1379	1	6	0	0	0
6186	1	0	0	0	1
7941	0	0	3	0	1
5126	1	3	2	0	1

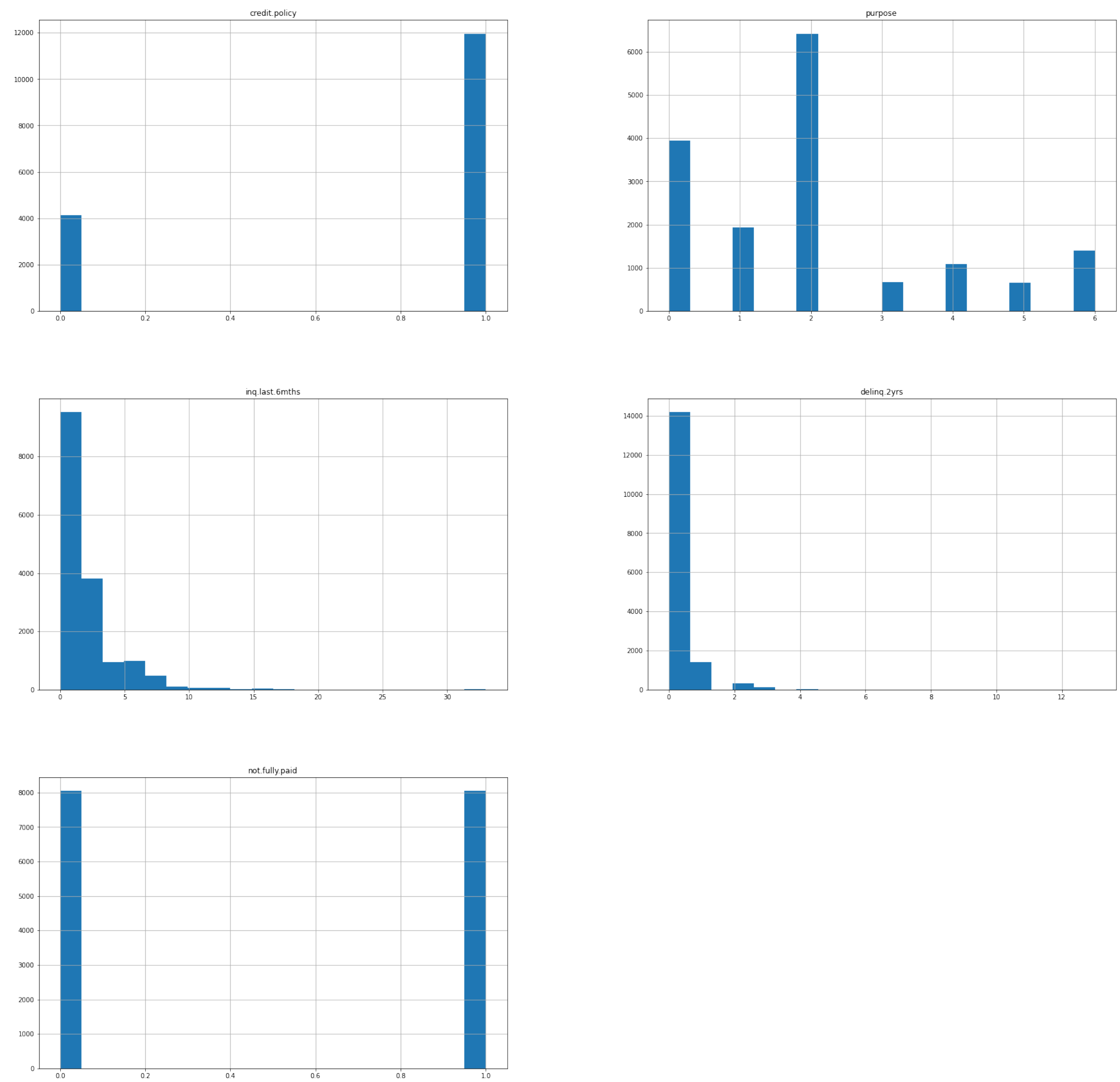
[20]:
```python
# Check the statistics of the numerical data
cat_data.describe()
```

[20]:
```
         credit.policy       purpose  inq.last.6mths   delinq.2yrs  \
count     16090.000000  16090.000000    16090.000000  16090.000000
mean          0.743505      2.034866        1.885208      0.164512
```

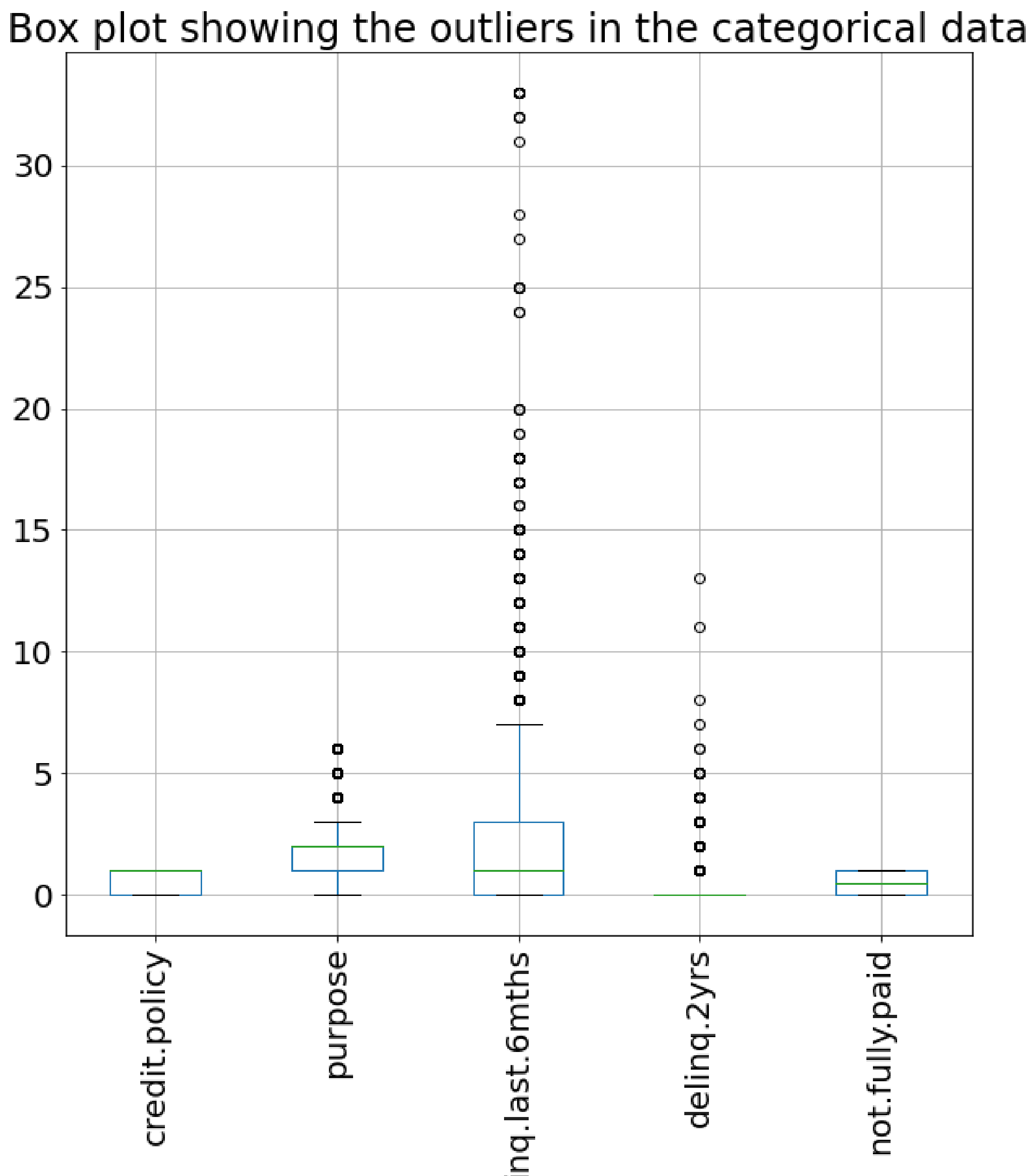
```
std     0.436712   1.778831    2.536954    0.528364
min     0.000000   0.000000    0.000000    0.000000
25%     0.000000   1.000000    0.000000    0.000000
50%     1.000000   2.000000    1.000000    0.000000
75%     1.000000   2.000000    3.000000    0.000000
max     1.000000   6.000000   33.000000   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
```

[21]:

```
# Check the distribution of the categorical data
cat_data.hist(figsize = (30, 30), bins = 20, legend = False)
plt.rcParams["font.size"] = "20"
plt.show()
```



[22]:
```
# Create plots showing the uncertainity in the categorical data and the outliers.
plt.figure(figsize = (10, 10))
cat_data.boxplot()
plt.xticks(rotation = 90)
plt.title("Box plot showing the outliers in the categorical data")
plt.show()
```



DATA WRANGLING

Handling missing values in the data frame

[23]:
```
# Convert the categorical feature in the data set into a numerical feature
le = preprocessing.LabelEncoder()
df["purpose"] = le.fit_transform(df["purpose"].astype(str))
df.head()
```

[23]:
```
      credit.policy  purpose  int.rate  installment  log.annual.inc    dti  \
9155              0        6    0.1979       296.46       11.350407   9.94
1379              1        6    0.0938       383.73       11.813030   9.91
6186              1        0    0.0894       111.21       10.778956  15.80
7941              0        0    0.1565       293.87       10.491274  21.50
5126              1        3    0.1461       137.91       10.915016   3.16

      fico  days.with.cr.line  revol.bal  revol.util  inq.last.6mths  \
9155   692        1800.000000       7491        29.0               6
1379   767       13620.000000       1579         7.2               0
6186   722        3870.041667      10659        76.4               0
7941   642        1830.000000       3896        34.2               3
5126   667        7410.000000       3864        52.9               2

      delinq.2yrs  pub.rec  not.fully.paid
9155            0        0               1
1379            0        0               0
6186            0        0               1
7941            0        0               1
5126            0        0               1
```

[24]:
```python
# Check for missing values in the data frame
df.isnull().sum()
```

[24]:
```
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
```

Handling outliers and skewness in the numerical variable of our data set.

[25]:
```python
# 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)])))
```

[26]:
```
# Determine what the upperbound should be for continuous features in dataframe.
for feat in num_data:
    detect_outlier(feat)
```

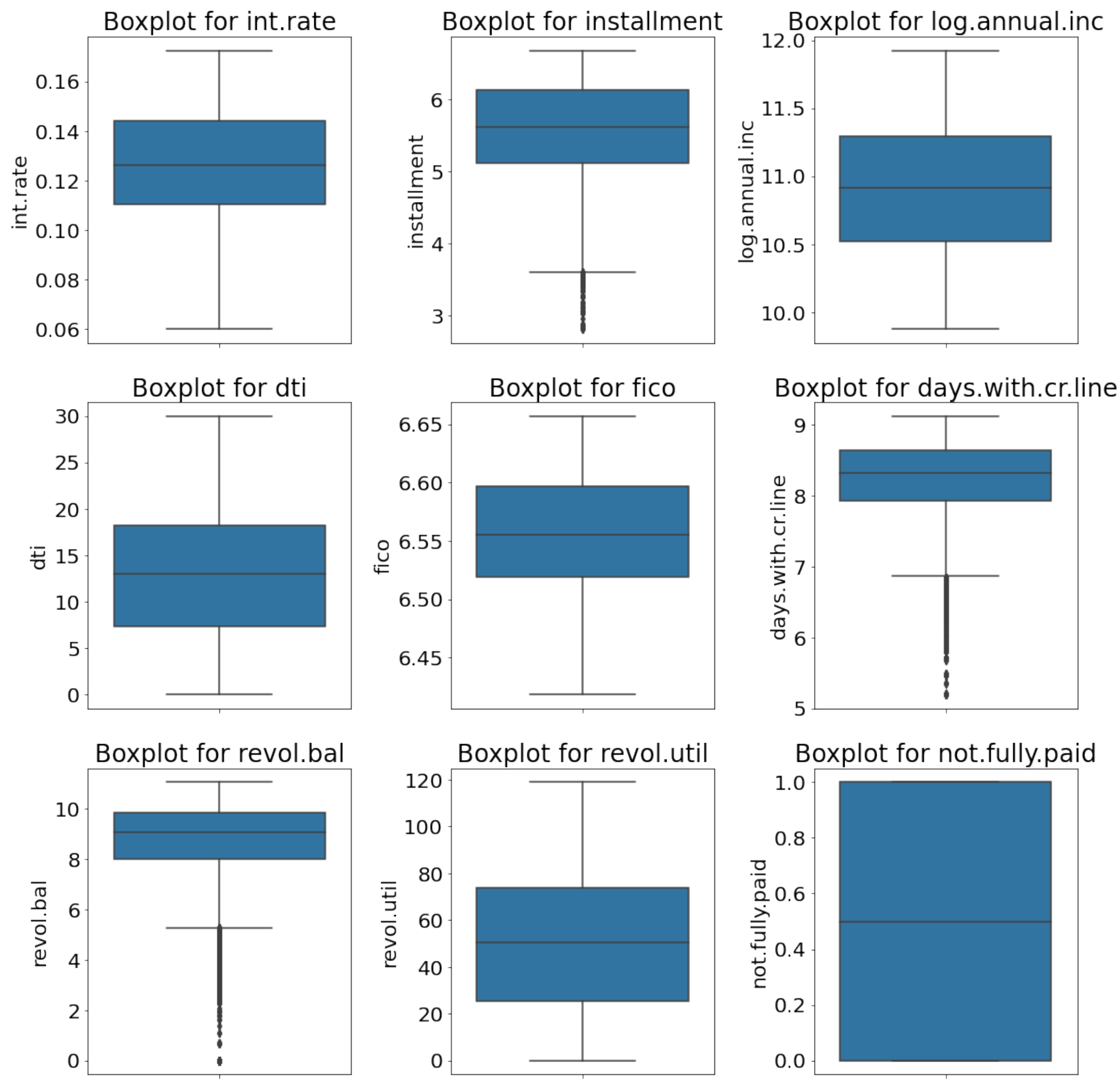
```
Outlier caps for int.rate
  --95p: 0.2 / 789 values exceed that
  --3sd: 0.2 / 42 values exceed that
  --99p: 0.2 / 129 values exceed that

Outlier caps for installment
  --95p: 798.8 / 802 values exceed that
  --3sd: 967.5 / 0 values exceed that
  --99p: 874.7 / 157 values exceed that

Outlier caps for log.annual.inc
  --95p: 11.9 / 804 values exceed that
  --3sd: 12.8 / 150 values exceed that
  --99p: 12.5 / 160 values exceed that

Outlier caps for dti
  --95p: 23.9 / 797 values exceed that
  --3sd: 33.7 / 0 values exceed that
  --99p: 26.9 / 152 values exceed that

Outlier caps for fico
  --95p: 777.0 / 655 values exceed that
```

```
  --3sd: 816.2 / 16 values exceed that
  --99p: 802.0 / 107 values exceed that

Outlier caps for days.with.cr.line
  --95p: 9150.0 / 801 values exceed that
  --3sd: 11958.4 / 240 values exceed that
  --99p: 12930.0 / 157 values exceed that

Outlier caps for revol.bal
  --95p: 64403.3 / 805 values exceed that
  --3sd: 156266.1 / 258 values exceed that
  --99p: 204960.8 / 161 values exceed that

Outlier caps for revol.util
  --95p: 94.9 / 799 values exceed that
  --3sd: 136.8 / 0 values exceed that
  --99p: 99.2 / 154 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
```

[27]:
```python
# Capping features in df to remover outliers in numerical features

# Upper bounded outliers
for var in ['int.rate' ,'installment', 'log.annual.inc', 'fico', 'days.with.cr.
 ↪line', 'revol.bal', 'not.fully.paid']:
    df[var].clip(upper=df[var].quantile(.95), inplace=True)

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

[34]:
```python
# Check for the presence of outliers in the numerical data of the dataframe␣
 ↪again

# Define subplot grid
numerical_df = df[num_data_features]
plt.figure(figsize = (15, 15))

def num_plot(df, a, var):

    ax = plt.subplot(3, 3, a+1)
    sns.boxplot(y = df[var])
    ax.set_title(f"Boxplot for {var}")
```

```
    plt.tight_layout()

for i, col in enumerate(numerical_df):
    num_plot(numerical_df, i, col)
```



Check the skewness in the numerical data of the dataframe

[33]:
```
# Check for skewness in the numerical features
vars_skewed = df[num_data_features].apply(lambda x: skew(x)).
 ↪sort_values(ascending = False)
vars_skewed
```

[33]:
```
fico                    0.287226
dti                     0.002105
```

```
not.fully.paid       0.000000
log.annual.inc      -0.015148
revol.util          -0.045215
int.rate            -0.109951
installment         -0.574942
days.with.cr.line   -1.154591
revol.bal           -2.268189
dtype: float64
```

[30]:
```
# Getting numerical features with skewness higher than 0.3.
high_skew = vars_skewed[abs(vars_skewed) > 0.3]
high_skew
```

[30]:
```
revol.bal            1.688205
installment          0.784034
days.with.cr.line    0.478981
fico                 0.375329
dtype: float64
```

[31]:
```
# Correct the skeness in the numerical features
for feat in high_skew.index:
    df[feat] = np.log1p(df[feat])
```

[32]:
```
# Check for skewness in the numerical data again for the entire data set
vars_skewed = df[num_data_features].apply(lambda x: skew(x)).
 ↪sort_values(ascending = False)
vars_skewed
```

[32]:
```
fico                 0.287226
dti                  0.002105
not.fully.paid       0.000000
log.annual.inc      -0.015148
revol.util          -0.045215
int.rate            -0.109951
installment         -0.574942
days.with.cr.line   -1.154591
revol.bal           -2.268189
dtype: float64
```

Handle outliers and skewness in categorical features in our dataframe

[35]:
```
# Detect outliers in categorical data
for feat in cat_data:
    detect_outlier(feat)
```

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

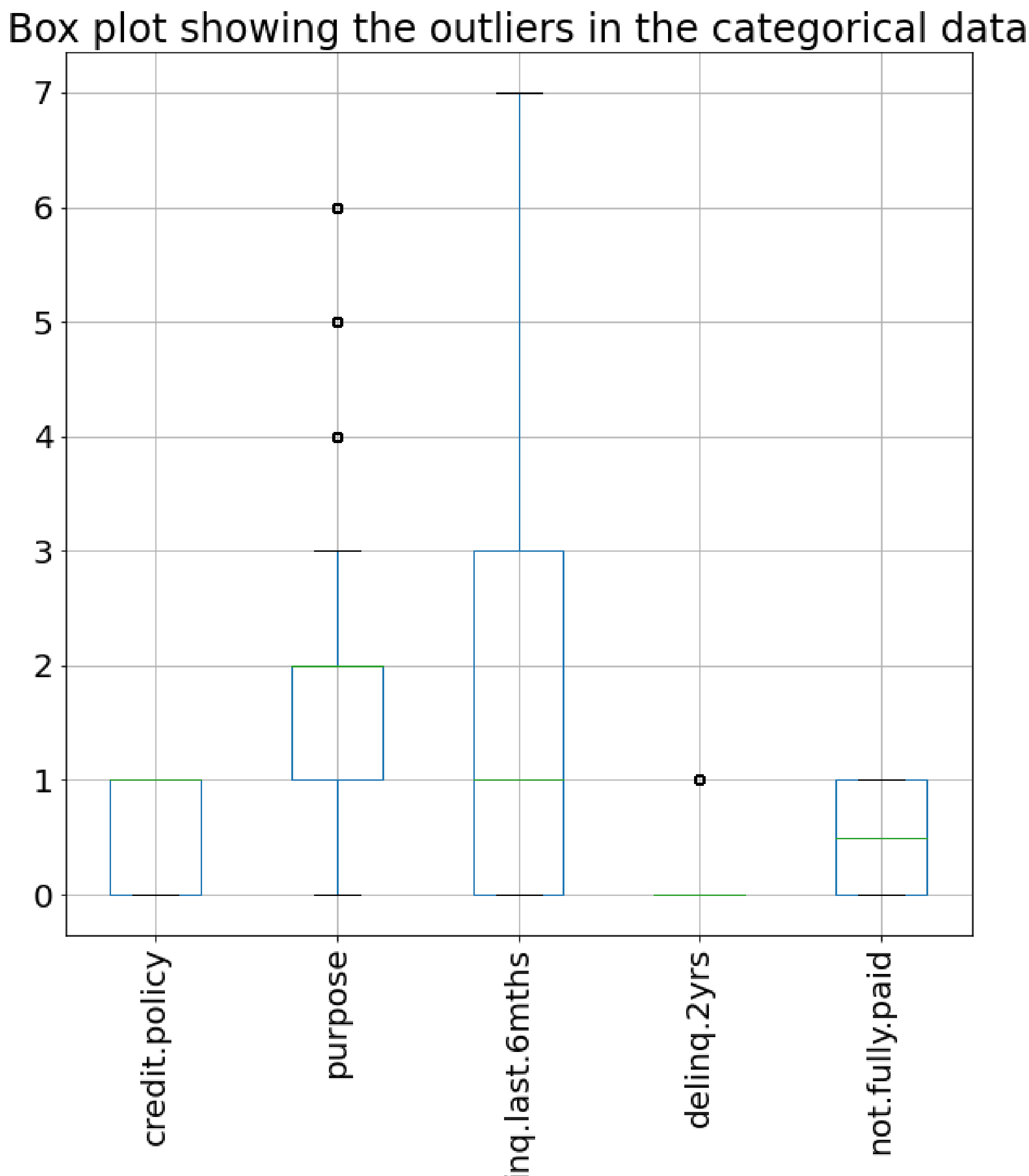
```
  --3sd: 2.1 / 0 values exceed that
  --99p: 1.0 / 0 values exceed that

Outlier caps for purpose
  --95p: 6.0 / 0 values exceed that
  --3sd: 7.4 / 0 values exceed that
  --99p: 6.0 / 0 values exceed that

Outlier caps for inq.last.6mths
  --95p: 7.0 / 559 values exceed that
  --3sd: 9.5 / 226 values exceed that
  --99p: 11.0 / 155 values exceed that

Outlier caps for delinq.2yrs
  --95p: 1.0 / 495 values exceed that
  --3sd: 1.7 / 495 values exceed that
  --99p: 3.0 / 39 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
```

[36]:

```
# Capping features in combined_df to remove outliers in categorical features

# Upper bounded outliers
for cat in ['credit.policy', 'purpose', 'inq.last.6mths', 'delinq.2yrs']:
    df[cat].clip(upper=df[cat].quantile(.95), inplace=True)
```

[37]:

```
# Check for the presence of outliers in the numerical data of the dataframe␣
 ↪again

# Define subplot grid
categorical_df = df[cat_data_features]

plt.figure(figsize = (10, 10))
categorical_df.boxplot()
plt.xticks(rotation = 90)
plt.title("Box plot showing the outliers in the categorical data")
plt.show()
```



Handle skeness in the categorical data of the dataframe

[38]:
```
# Identify the skewness in the categorical data
for cat in cat_data:
    cat_skewed = df[cat].skew()
    print(f"{cat}", cat_skewed)
```

```
credit.policy -1.1153152277973852
purpose 0.8566419597830179
```

```
inq.last.6mths 1.237694849978447
delinq.2yrs 2.352426920685018
not.fully.paid 0.0
```

[39]:
```
# Correct the skewness in categorical features of the dataframe if skewness is␣
↪greater than 0.3.
for cat in cat_data:
    cat_skewed = df[cat].skew()
    if (cat_skewed) > 0.3:
        df[cat] = np.log1p(df[cat])
```

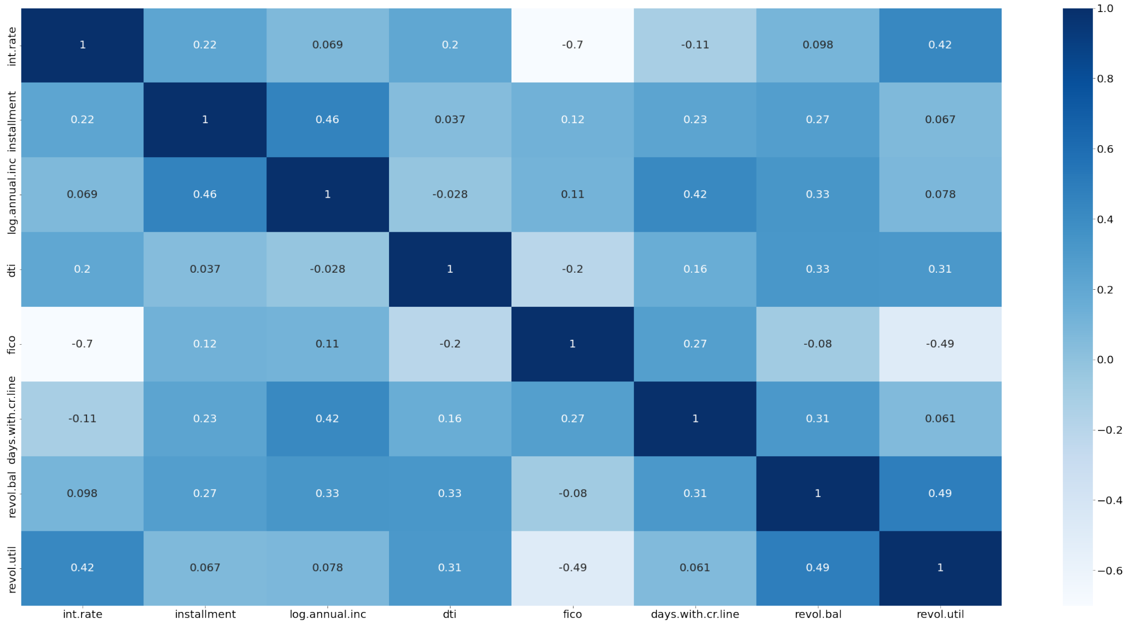
[40]:
```
# Confirm the correction  of the skewness in the categorical data again
for cat in cat_data:
    cat_skewed = df[cat].skew()
    print(f"{cat}", cat_skewed)
```

```
credit.policy -1.1153152277973852
purpose -0.2191819509204393
inq.last.6mths 0.2910423678987724
delinq.2yrs 2.352426920685016
not.fully.paid 0.0
```

FEATURE ENGINEERING

[41]:
```
# Identify the correlations in the numerical data

# Independent variables
X_num = df[num_data_features]
X_num = X_num.drop(['not.fully.paid'], axis = 1)

# Dependent variable
Y = df[['not.fully.paid']]
```

[42]:
```
# Generate a correlation
matrix = X_num.corr()
plt.figure(figsize = [40, 20])
sns.heatmap(matrix, annot = True, cmap = "Blues");
```



[43]:
```
# Select strong correlations among features
cor_pairs = matrix.unstack()
sorted_pairs = cor_pairs.sort_values(kind = 'quicksort')
strong_pairs = sorted_pairs[abs(sorted_pairs) > 0.7]

print(strong_pairs)
```

```
fico               int.rate            -0.701051
int.rate           fico                -0.701051
                   int.rate             1.000000
days.with.cr.line  days.with.cr.line    1.000000
fico               fico                 1.000000
dti                dti                  1.000000
log.annual.inc     log.annual.inc       1.000000
installment        installment          1.000000
revol.bal          revol.bal            1.000000
revol.util         revol.util           1.000000
dtype: float64
```

[44]:
```
def get_redundant_pairs(df):
    '''Get diagonal and lower triangular pairs of correlation matrix'''
    pairs_to_drop = set()
    cols = df.columns
    for i in range(0, df.shape[1]):
        for j in range(0, i+1):
            pairs_to_drop.add((cols[i], cols[j]))
    return pairs_to_drop
```

```
# Get top pairs
def get_top_abs_correlations(df, n=10):
    corr_list = df.abs().unstack()
    labels_to_drop = get_redundant_pairs(df)
    corr_list = corr_list.drop(labels=labels_to_drop).
↪sort_values(ascending=False)
    return corr_list[0:n]
```

[45]:
```
# Get top 10 correlation pairs
print('Top 10 correlation pairs:')
get_top_abs_correlations(matrix, 5)
```

```
Top 10 correlation pairs:
```

[45]:
```
int.rate     fico              0.701051
revol.bal    revol.util        0.491859
fico         revol.util        0.487608
installment  log.annual.inc    0.459022
int.rate     revol.util        0.424882
dtype: float64
```

[46]:
```
# Feature Selection
Y = le.fit_transform(Y)
from sklearn.datasets import make_friedman1
from sklearn.svm import SVR
X_num, Y = make_friedman1(n_samples=9578, n_features=8, random_state=42)
estimator = SVR(kernel="linear")
rfe = RFE(estimator, n_features_to_select=5, step=1)
rfe = rfe.fit(X_num, Y.ravel())
```

```
/usr/local/lib/python3.7/site-packages/sklearn/preprocessing/_label.py:115:
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)
```

[47]:
```
num_cols = df[num_data_features].drop(['not.fully.paid'], axis = 1)
```

[48]:
```
num_cols = num_cols.columns
num_cols
```

[48]:
```
Index(['int.rate', 'installment', 'log.annual.inc', 'dti', 'fico',
       'days.with.cr.line', 'revol.bal', 'revol.util'],
      dtype='object')
```

[49]:
```
# Check the RFE ranking
X_num = pd.DataFrame(X_num, columns = [num_cols])
list(zip(X_num.columns, rfe.support_, rfe.ranking_))
```

[49]:
```
[(('int.rate',), True, 1),
 (('installment',), True, 1),
 (('log.annual.inc',), True, 1),
 (('dti',), True, 1),
 (('fico',), True, 1),
 (('days.with.cr.line',), False, 2),
 (('revol.bal',), False, 3),
 (('revol.util',), False, 4)]
```

[50]:
```
# Columns selected by RFE
cols = X_num.columns[rfe.support_]
cols
```

[50]:
```
MultiIndex([(      'int.rate',),
            (   'installment',),
            ('log.annual.inc',),
            (           'dti',),
            (          'fico',)],
           )
```

[51]:
```
# columns not selected by RFE
X_num.columns[~rfe.support_]
```

[51]:
```
MultiIndex([('days.with.cr.line',),
            (        'revol.bal',),
            (       'revol.util',)],
           )
```

[52]:
```
# Show the selected numerical features
num_vals = df[['int.rate', 'installment', 'log.annual.inc', 'dti', 'fico']]
num_vals.head()
```

[52]:

	int.rate	installment	log.annual.inc	dti	fico
9155	0.1726	5.695280	11.350407	9.94	6.541030
1379	0.0938	5.952542	11.813030	9.91	6.643790
6186	0.0894	4.720372	10.778956	15.80	6.583409
7941	0.1565	5.686535	10.491274	21.50	6.466145
5126	0.1461	4.933826	10.915016	3.16	6.504288

Select the best features in categorical data

[54]:
```
# Collecting the categorical data
cat_vars = df[cat_data_features].drop(['not.fully.paid'], axis = 1)
```

```
cat_vars
```

[54]:

	credit.policy	purpose	inq.last.6mths	delinq.2yrs
9155	0	1.945910	1.945910	0.000000
1379	1	1.945910	0.000000	0.000000
6186	1	0.000000	0.000000	0.000000
7941	0	0.000000	1.386294	0.000000
5126	1	1.386294	1.098612	0.000000
…	…	…	…	…
1916	1	0.000000	0.693147	0.000000
8081	0	0.000000	2.079442	0.000000
8587	0	1.098612	1.609438	0.000000
4666	1	1.098612	0.693147	0.000000
7964	0	1.098612	0.693147	0.693147

[16090 rows x 4 columns]

[55]:
```
# Perform the chi test and determine the f score and the p value
f_p_values = chi2(cat_vars, df['not.fully.paid'])
f_p_values
```

[55]:
```
(array([170.69639722,  13.91438805, 318.1131853 ,   1.67281449]),
 array([5.21276521e-39, 1.91328240e-04, 3.73159133e-71, 1.95881980e-01]))
```

[56]:
```
# Representing the p values in list form
p_values = pd.Series(f_p_values[1])
p_values.index = cat_vars.columns
p_values
```

[56]:
```
credit.policy     5.212765e-39
purpose           1.913282e-04
inq.last.6mths    3.731591e-71
delinq.2yrs       1.958820e-01
dtype: float64
```

[57]:
```
# Sorting the p values in ascending order
p_values.sort_values(ascending = True)
```

[57]:
```
inq.last.6mths    3.731591e-71
credit.policy     5.212765e-39
purpose           1.913282e-04
delinq.2yrs       1.958820e-01
dtype: float64
```

DATA TRAINING

[58]:
```
# Divide the data into features and target variables
X = df[['int.rate', 'installment', 'log.annual.inc', 'dti', 'fico', 'inq.last.
 ↪6mths',
        'credit.policy', 'purpose']]
y = df['not.fully.paid']
```

[60]:
```
# Split the data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
 ↪random_state = 42)
```

[61]:
```
# Scale the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[62]:
```
model  = keras.Sequential(
    [
        keras.layers.Dense(
        256, activation="relu", input_shape=[8]),
        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"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               2304

 dense_1 (Dense)             (None, 256)               65792

 dropout (Dropout)           (None, 256)               0

 dense_2 (Dense)             (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 1)                 257

=================================================================
Total params: 134,145
Trainable params: 134,145
```

```
Non-trainable params: 0
_________________________________________________________________
```

[63]:
```
model.compile(optimizer = 'Adam', loss = 'binary_crossentropy', metrics =␣
↪['binary_accuracy'])
```

[64]:
```
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=1000,
    callbacks=[early_stopping],
    verbose=1,
)
```

```
Epoch 1/1000
51/51 [==============================] - 8s 15ms/step - loss: 0.6472 -
binary_accuracy: 0.6184 - val_loss: 0.6431 - val_binary_accuracy: 0.6243
Epoch 2/1000
51/51 [==============================] - 0s 9ms/step - loss: 0.6361 -
binary_accuracy: 0.6312 - val_loss: 0.6386 - val_binary_accuracy: 0.6277
Epoch 3/1000
51/51 [==============================] - 0s 9ms/step - loss: 0.6305 -
binary_accuracy: 0.6406 - val_loss: 0.6341 - val_binary_accuracy: 0.6231
Epoch 4/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.6269 -
binary_accuracy: 0.6405 - val_loss: 0.6354 - val_binary_accuracy: 0.6224
Epoch 5/1000
51/51 [==============================] - 0s 9ms/step - loss: 0.6233 -
binary_accuracy: 0.6509 - val_loss: 0.6290 - val_binary_accuracy: 0.6305
Epoch 6/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.6178 -
binary_accuracy: 0.6505 - val_loss: 0.6261 - val_binary_accuracy: 0.6355
Epoch 7/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.6146 -
binary_accuracy: 0.6594 - val_loss: 0.6238 - val_binary_accuracy: 0.6318
Epoch 8/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.6078 -
binary_accuracy: 0.6592 - val_loss: 0.6274 - val_binary_accuracy: 0.6370
Epoch 9/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.6017 -
binary_accuracy: 0.6661 - val_loss: 0.6176 - val_binary_accuracy: 0.6367
Epoch 10/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5964 -
binary_accuracy: 0.6720 - val_loss: 0.6230 - val_binary_accuracy: 0.6414
Epoch 11/1000
```

```
51/51 [==============================] - 0s 8ms/step - loss: 0.5926 -
binary_accuracy: 0.6746 - val_loss: 0.6118 - val_binary_accuracy: 0.6479
Epoch 12/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5847 -
binary_accuracy: 0.6821 - val_loss: 0.6048 - val_binary_accuracy: 0.6576
Epoch 13/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5783 -
binary_accuracy: 0.6911 - val_loss: 0.6113 - val_binary_accuracy: 0.6603
Epoch 14/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5752 -
binary_accuracy: 0.6901 - val_loss: 0.5960 - val_binary_accuracy: 0.6653
Epoch 15/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5649 -
binary_accuracy: 0.7013 - val_loss: 0.5881 - val_binary_accuracy: 0.6771
Epoch 16/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.5581 -
binary_accuracy: 0.7056 - val_loss: 0.5835 - val_binary_accuracy: 0.6818
Epoch 17/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5507 -
binary_accuracy: 0.7128 - val_loss: 0.5840 - val_binary_accuracy: 0.6802
Epoch 18/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5455 -
binary_accuracy: 0.7157 - val_loss: 0.5794 - val_binary_accuracy: 0.6787
Epoch 19/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.5396 -
binary_accuracy: 0.7250 - val_loss: 0.5784 - val_binary_accuracy: 0.6846
Epoch 20/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5313 -
binary_accuracy: 0.7282 - val_loss: 0.5683 - val_binary_accuracy: 0.6924
Epoch 21/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5257 -
binary_accuracy: 0.7327 - val_loss: 0.5504 - val_binary_accuracy: 0.7048
Epoch 22/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5170 -
binary_accuracy: 0.7370 - val_loss: 0.5505 - val_binary_accuracy: 0.6986
Epoch 23/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5112 -
binary_accuracy: 0.7408 - val_loss: 0.5493 - val_binary_accuracy: 0.7094
Epoch 24/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5041 -
binary_accuracy: 0.7446 - val_loss: 0.5524 - val_binary_accuracy: 0.7094
Epoch 25/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.5013 -
binary_accuracy: 0.7458 - val_loss: 0.5358 - val_binary_accuracy: 0.7172
Epoch 26/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.4945 -
binary_accuracy: 0.7574 - val_loss: 0.5308 - val_binary_accuracy: 0.7253
Epoch 27/1000
```

```
51/51 [==============================] - 0s 8ms/step - loss: 0.4829 -
binary_accuracy: 0.7584 - val_loss: 0.5185 - val_binary_accuracy: 0.7315
Epoch 28/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.4728 -
binary_accuracy: 0.7719 - val_loss: 0.5146 - val_binary_accuracy: 0.7402
Epoch 29/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.4667 -
binary_accuracy: 0.7676 - val_loss: 0.5286 - val_binary_accuracy: 0.7296
Epoch 30/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.4674 -
binary_accuracy: 0.7685 - val_loss: 0.5097 - val_binary_accuracy: 0.7449
Epoch 31/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.4520 -
binary_accuracy: 0.7809 - val_loss: 0.5061 - val_binary_accuracy: 0.7486
Epoch 32/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.4505 -
binary_accuracy: 0.7829 - val_loss: 0.4983 - val_binary_accuracy: 0.7436
Epoch 33/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.4407 -
binary_accuracy: 0.7894 - val_loss: 0.4922 - val_binary_accuracy: 0.7557
Epoch 34/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.4429 -
binary_accuracy: 0.7909 - val_loss: 0.4910 - val_binary_accuracy: 0.7517
Epoch 35/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.4314 -
binary_accuracy: 0.7980 - val_loss: 0.4854 - val_binary_accuracy: 0.7598
Epoch 36/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.4218 -
binary_accuracy: 0.7985 - val_loss: 0.4851 - val_binary_accuracy: 0.7610
Epoch 37/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.4131 -
binary_accuracy: 0.8041 - val_loss: 0.4794 - val_binary_accuracy: 0.7657
Epoch 38/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.4172 -
binary_accuracy: 0.8057 - val_loss: 0.4595 - val_binary_accuracy: 0.7772
Epoch 39/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.4042 -
binary_accuracy: 0.8098 - val_loss: 0.4707 - val_binary_accuracy: 0.7756
Epoch 40/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.4013 -
binary_accuracy: 0.8094 - val_loss: 0.4652 - val_binary_accuracy: 0.7856
Epoch 41/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.3927 -
binary_accuracy: 0.8181 - val_loss: 0.4561 - val_binary_accuracy: 0.7828
Epoch 42/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3898 -
binary_accuracy: 0.8156 - val_loss: 0.4526 - val_binary_accuracy: 0.7831
Epoch 43/1000
```

```
51/51 [==============================] - 0s 8ms/step - loss: 0.3837 -
binary_accuracy: 0.8223 - val_loss: 0.4501 - val_binary_accuracy: 0.7850
Epoch 44/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3760 -
binary_accuracy: 0.8295 - val_loss: 0.4428 - val_binary_accuracy: 0.7909
Epoch 45/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3783 -
binary_accuracy: 0.8265 - val_loss: 0.4337 - val_binary_accuracy: 0.7999
Epoch 46/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.3669 -
binary_accuracy: 0.8332 - val_loss: 0.4422 - val_binary_accuracy: 0.7993
Epoch 47/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3551 -
binary_accuracy: 0.8389 - val_loss: 0.4400 - val_binary_accuracy: 0.8002
Epoch 48/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.3564 -
binary_accuracy: 0.8379 - val_loss: 0.4416 - val_binary_accuracy: 0.8017
Epoch 49/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.3537 -
binary_accuracy: 0.8428 - val_loss: 0.4392 - val_binary_accuracy: 0.8039
Epoch 50/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3463 -
binary_accuracy: 0.8460 - val_loss: 0.4174 - val_binary_accuracy: 0.8213
Epoch 51/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3486 -
binary_accuracy: 0.8464 - val_loss: 0.4182 - val_binary_accuracy: 0.8089
Epoch 52/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3440 -
binary_accuracy: 0.8466 - val_loss: 0.4104 - val_binary_accuracy: 0.8204
Epoch 53/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3389 -
binary_accuracy: 0.8484 - val_loss: 0.4045 - val_binary_accuracy: 0.8185
Epoch 54/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.3309 -
binary_accuracy: 0.8552 - val_loss: 0.4126 - val_binary_accuracy: 0.8182
Epoch 55/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.3276 -
binary_accuracy: 0.8547 - val_loss: 0.4332 - val_binary_accuracy: 0.8108
Epoch 56/1000
51/51 [==============================] - 0s 9ms/step - loss: 0.3192 -
binary_accuracy: 0.8616 - val_loss: 0.4050 - val_binary_accuracy: 0.8232
Epoch 57/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3153 -
binary_accuracy: 0.8637 - val_loss: 0.4113 - val_binary_accuracy: 0.8198
Epoch 58/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.3072 -
binary_accuracy: 0.8665 - val_loss: 0.4113 - val_binary_accuracy: 0.8182
Epoch 59/1000
```

```
51/51 [==============================] - 0s 8ms/step - loss: 0.3120 -
binary_accuracy: 0.8628 - val_loss: 0.3971 - val_binary_accuracy: 0.8350
Epoch 60/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.3122 -
binary_accuracy: 0.8649 - val_loss: 0.3892 - val_binary_accuracy: 0.8384
Epoch 61/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.3023 -
binary_accuracy: 0.8671 - val_loss: 0.3924 - val_binary_accuracy: 0.8285
Epoch 62/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2977 -
binary_accuracy: 0.8703 - val_loss: 0.4089 - val_binary_accuracy: 0.8334
Epoch 63/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.2909 -
binary_accuracy: 0.8731 - val_loss: 0.3983 - val_binary_accuracy: 0.8297
Epoch 64/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2878 -
binary_accuracy: 0.8743 - val_loss: 0.3934 - val_binary_accuracy: 0.8431
Epoch 65/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2871 -
binary_accuracy: 0.8755 - val_loss: 0.3866 - val_binary_accuracy: 0.8337
Epoch 66/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2763 -
binary_accuracy: 0.8797 - val_loss: 0.3900 - val_binary_accuracy: 0.8347
Epoch 67/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2747 -
binary_accuracy: 0.8772 - val_loss: 0.3940 - val_binary_accuracy: 0.8356
Epoch 68/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2790 -
binary_accuracy: 0.8804 - val_loss: 0.3756 - val_binary_accuracy: 0.8468
Epoch 69/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.2731 -
binary_accuracy: 0.8830 - val_loss: 0.3682 - val_binary_accuracy: 0.8465
Epoch 70/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.2727 -
binary_accuracy: 0.8850 - val_loss: 0.3758 - val_binary_accuracy: 0.8496
Epoch 71/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.2650 -
binary_accuracy: 0.8867 - val_loss: 0.3665 - val_binary_accuracy: 0.8493
Epoch 72/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2581 -
binary_accuracy: 0.8881 - val_loss: 0.3832 - val_binary_accuracy: 0.8524
Epoch 73/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2604 -
binary_accuracy: 0.8865 - val_loss: 0.3786 - val_binary_accuracy: 0.8518
Epoch 74/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2619 -
binary_accuracy: 0.8887 - val_loss: 0.3794 - val_binary_accuracy: 0.8499
Epoch 75/1000
```

```
51/51 [==============================] - 0s 7ms/step - loss: 0.2483 -
binary_accuracy: 0.8947 - val_loss: 0.3684 - val_binary_accuracy: 0.8614
Epoch 76/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2533 -
binary_accuracy: 0.8905 - val_loss: 0.3676 - val_binary_accuracy: 0.8561
Epoch 77/1000
51/51 [==============================] - 0s 7ms/step - loss: 0.2496 -
binary_accuracy: 0.8950 - val_loss: 0.3700 - val_binary_accuracy: 0.8645
Epoch 78/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2466 -
binary_accuracy: 0.8968 - val_loss: 0.3657 - val_binary_accuracy: 0.8661
Epoch 79/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2448 -
binary_accuracy: 0.8961 - val_loss: 0.3757 - val_binary_accuracy: 0.8611
Epoch 80/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2311 -
binary_accuracy: 0.9066 - val_loss: 0.3668 - val_binary_accuracy: 0.8586
Epoch 81/1000
51/51 [==============================] - 0s 8ms/step - loss: 0.2371 -
binary_accuracy: 0.9006 - val_loss: 0.3997 - val_binary_accuracy: 0.8456
```

[65]:
```
predictions =(model.predict(X_test)>0.5).astype("int32")

predictions
```

[65]:
```
array([[1],
       [0],
       [1],
       …,
       [0],
       [1],
       [1]], dtype=int32)
```

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

[66]:
```
0.8492852703542573
```

[ ]:
```

```