Lending Club Loan Data Analysis

July 21, 2023

Beacon Nicolas Mkhabele Project

[3]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```python
data = pd.read_csv("loan_data.csv")
```

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

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

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

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

[6]:
```python
#understanding the dataset
data.shape
```

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

[7]:
```python
data.isnull().sum()
```

[7]:
```
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
```

[8]:
```
#Transforming categorical values into numerical values (discrete)
data.sample(10)
```

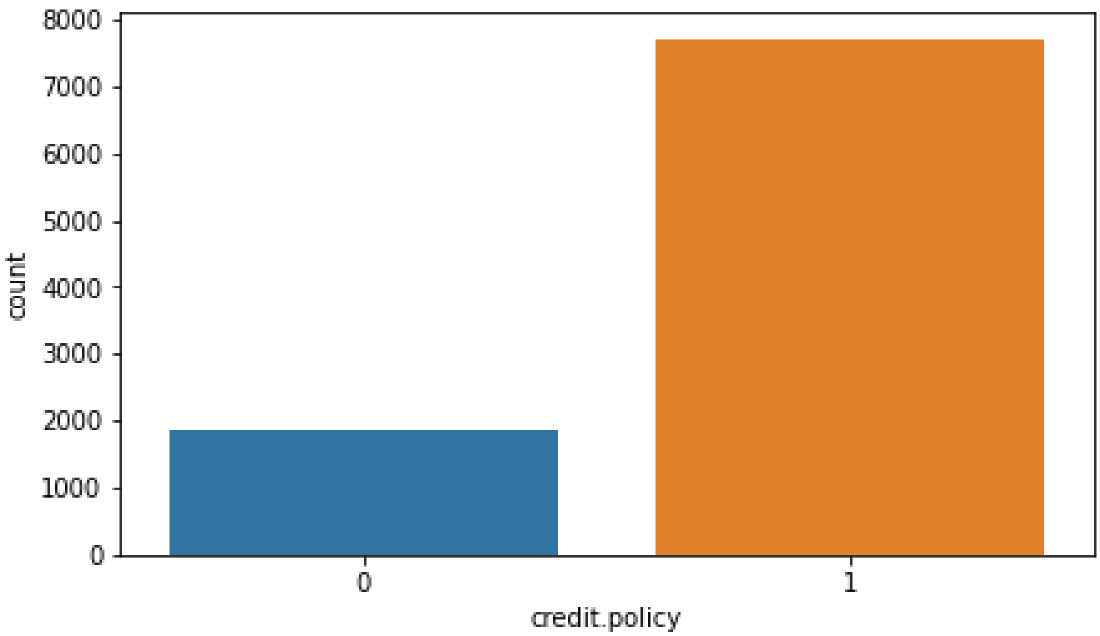
[8]:
```
      credit.policy             purpose  int.rate  installment  \
4157              1  debt_consolidation    0.1426       514.59
908               1  debt_consolidation    0.1197       292.16
1899              1  debt_consolidation    0.0963       320.95
3951              1         credit_card    0.1568       700.04
5580              1         credit_card    0.1565       131.20
3137              1    home_improvement    0.1284        53.79
5084              1           all_other    0.1218       333.00
4552              1  debt_consolidation    0.1287       504.50
4680              1  debt_consolidation    0.1218       666.00
6094              1  debt_consolidation    0.1114       518.30

      log.annual.inc    dti  fico  days.with.cr.line  revol.bal  revol.util  \
4157       10.858999  19.68   687        3313.000000      22918        90.9
908        10.819778   8.14   697        6840.000000       8243        37.0
1899       11.156251   6.31   737        3480.041667      20423        46.7
3951       11.695247  16.77   677        2280.000000      29799        53.0
5580        9.952278  18.91   697        1170.000000      13881        90.7
3137       10.561008   2.46   682        3180.000000       2076        83.0
5084       11.144814  18.22   712        5820.000000      17604        72.1
4552       11.652687   8.35   702        6089.958333      12152        54.2
4680       10.836950  22.88   732        3990.000000      26803        51.4
6094       11.512925  21.88   747        4620.000000      29860        47.6

      inq.last.6mths  delinq.2yrs  pub.rec  not.fully.paid
4157               1            0        0               0
908                2            0        1               0
1899               2            0        0               1
```

3951	3	0	0	0
5580	0	0	0	0
3137	1	0	0	0
5084	0	0	0	0
4552	2	0	1	0
4680	1	0	0	0
6094	0	0	0	0

[9]:
```
np.unique(data['credit.policy'], return_counts=True)
```

[9]:
```
(array([0, 1]), array([1868, 7710]))
```

[10]:
```
plt.figure(figsize=(7, 4))
sns.countplot(x=data['credit.policy']);
```



[11]:
```
data.dtypes
```

[11]:
```
credit.policy          int64
purpose               object
int.rate             float64
installment          float64
log.annual.inc       float64
dti                  float64
fico                   int64
days.with.cr.line    float64
revol.bal              int64
```

```
revol.util           float64
inq.last.6mths         int64
delinq.2yrs            int64
pub.rec                int64
not.fully.paid         int64
dtype: object
```

[12]:
```python
df_col_info = pd.DataFrame(data.dtypes, columns= ['col_data_type'])

df_col_info_names_changed = df_col_info.reset_index()

df_col_info_names_changed.columns = ['col_names', 'col_data_type']
df_col_info_names_changed
```

[12]:

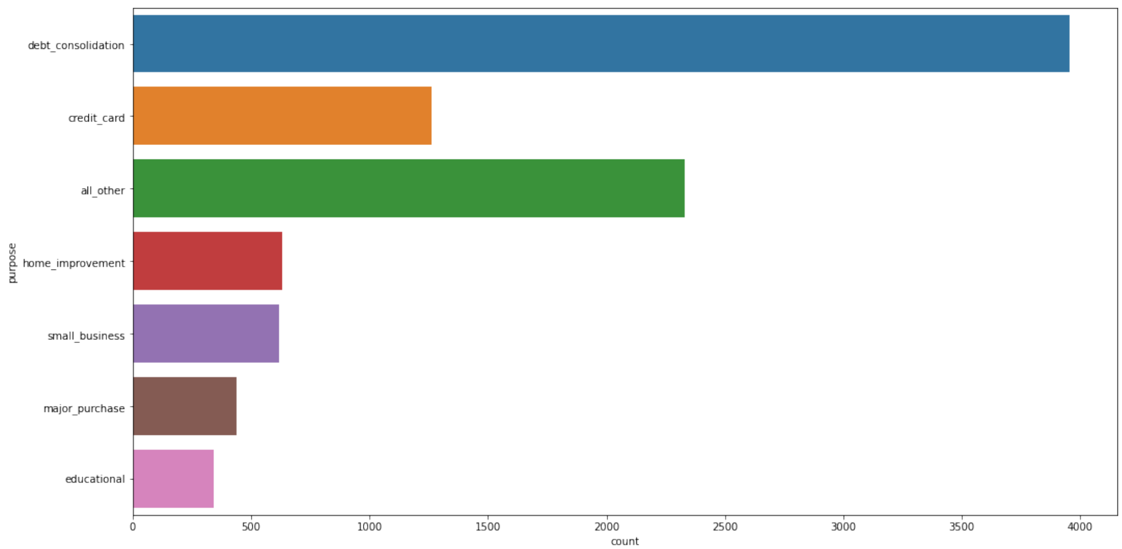
	col_names	col_data_type
0	credit.policy	int64
1	purpose	object
2	int.rate	float64
3	installment	float64
4	log.annual.inc	float64
5	dti	float64
6	fico	int64
7	days.with.cr.line	float64
8	revol.bal	int64
9	revol.util	float64
10	inq.last.6mths	int64
11	delinq.2yrs	int64
12	pub.rec	int64
13	not.fully.paid	int64

[13]:
```python
df_col_info_count = pd.DataFrame(df_col_info_names_changed.
 ↪groupby("col_data_type").count().reset_index())

df_col_info_count.columns= ['col_name', 'dtype']
df_col_info_count
```

[13]:

	col_name	dtype
0	int64	7
1	float64	6
2	object	1

[14]:
```python
plt.figure(figsize=(17, 9))
sns.countplot(y=data['purpose']);
```



[15]:
```
#Null value Analysis
NAs= pd.concat([data.isnull().sum()], axis=1)

NAs[NAs.sum(axis=1)>0]
```

[15]:
```
Empty DataFrame
Columns: [0]
Index: []
```

[16]:
```
pd.set_option('display.float_format', lambda x:'%.4f' % x)
```

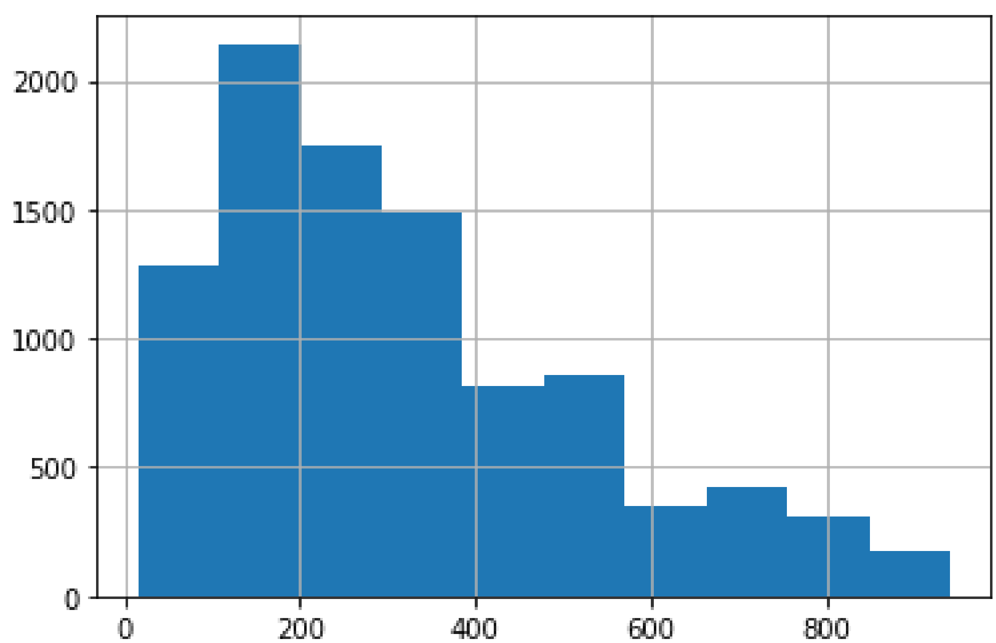
[17]:
```
data.describe().T
```

[17]:
```
                    count       mean        std      min       25%  \
credit.policy     9578.0000     0.8050     0.3962   0.0000    1.0000
int.rate          9578.0000     0.1226     0.0268   0.0600    0.1039
installment       9578.0000   319.0894   207.0713  15.6700  163.7700
log.annual.inc    9578.0000    10.9321     0.6148   7.5475   10.5584
dti               9578.0000    12.6067     6.8840   0.0000    7.2125
fico              9578.0000   710.8463    37.9705 612.0000  682.0000
days.with.cr.line 9578.0000  4560.7672  2496.9304 178.9583 2820.0000
revol.bal         9578.0000 16913.9639 33756.1896   0.0000 3187.0000
revol.util        9578.0000    46.7992    29.0144   0.0000   22.6000
inq.last.6mths    9578.0000     1.5775     2.2002   0.0000    0.0000
delinq.2yrs       9578.0000     0.1637     0.5462   0.0000    0.0000
pub.rec           9578.0000     0.0621     0.2621   0.0000    0.0000
not.fully.paid    9578.0000     0.1601     0.3667   0.0000    0.0000

                        50%        75%         max
```

```
credit.policy         1.0000     1.0000        1.0000
int.rate              0.1221     0.1407        0.2164
installment         268.9500   432.7625      940.1400
log.annual.inc       10.9289    11.2913       14.5284
dti                  12.6650    17.9500       29.9600
fico                707.0000   737.0000      827.0000
days.with.cr.line  4139.9583  5730.0000    17639.9583
revol.bal          8596.0000 18249.5000  1207359.0000
revol.util           46.3000    70.9000      119.0000
inq.last.6mths        1.0000     2.0000       33.0000
delinq.2yrs           0.0000     0.0000       13.0000
pub.rec               0.0000     0.0000        5.0000
not.fully.paid        0.0000     0.0000        1.0000
```

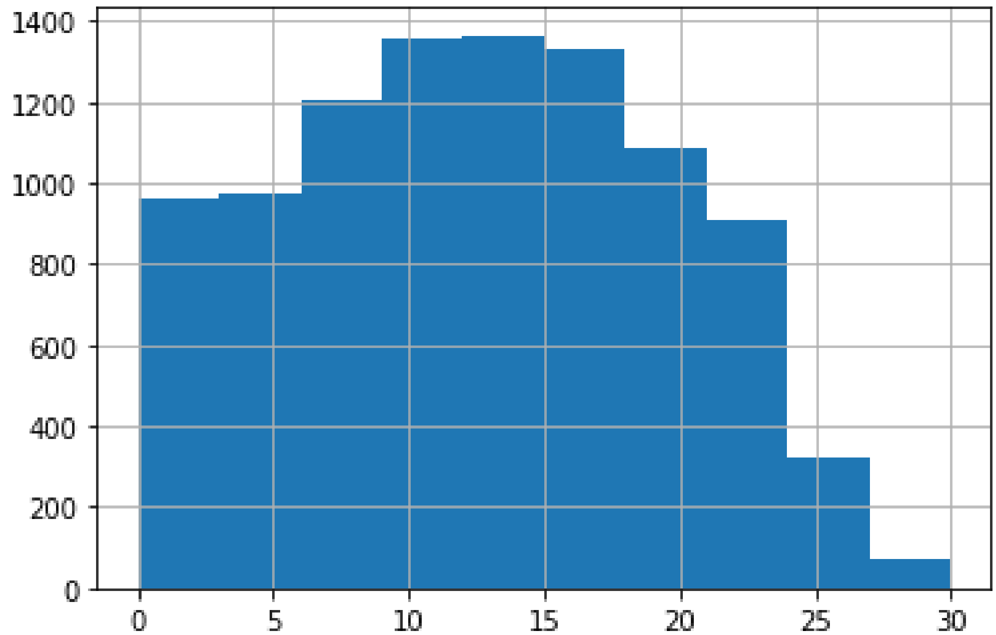
[18]:
```
data.installment.hist()
```

[18]: <AxesSubplot:>



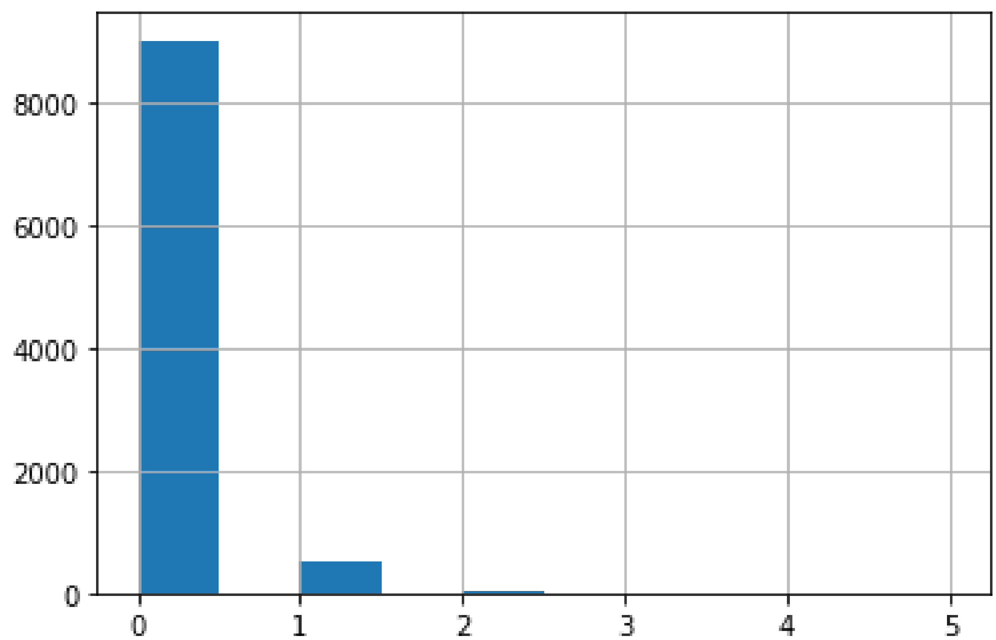
[19]:
```
data.dti.hist()
```

[19]: <AxesSubplot:>




```
[20]: data['pub.rec'].hist()
```

```
[20]: <AxesSubplot:>
```



[21]:
```
# evaluating the usefulness of the records

data.dtypes
```

[21]:
```
credit.policy          int64
purpose               object
int.rate             float64
installment          float64
log.annual.inc       float64
dti                  float64
fico                   int64
days.with.cr.line    float64
revol.bal              int64
revol.util           float64
inq.last.6mths         int64
delinq.2yrs            int64
pub.rec                int64
not.fully.paid         int64
dtype: object
```

[22]:
```
data.columns
```

[22]:
```
Index(['credit.policy', 'purpose', 'int.rate', 'installment', 'log.annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util',
       'inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.paid'],
      dtype='object')
```

[23]:
```
col_to_evaluate = ['credit.policy', 'purpose', 'int.rate', 'installment', 'log.
 ↪annual.inc',
       'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util']
```

[24]:
```
from scipy import stats
```

[25]:
```
# Forming 2 datasets from the original dataset as subsets

class1 = data[(data['credit.policy'] == 0)]
class1.reset_index(inplace= True)

class2 = data[(data['credit.policy'] == 1)]
class2.reset_index(inplace = True)
```

[26]:
```
stats.ttest_ind(class1['int.rate'],class2['int.rate'])[1]
```

[26]:
```
2.1945619717925775e-190
```

Since the predicted value is much lower than 5% then it is usefull for Machine learning model

this means we can create a model for the columns

[27]:
```
for each_col in col_to_evaluate:
    col1 = pd.to_numeric(class1[each_col], errors='coerce')
    col2 = pd.to_numeric(class2[each_col], errors='coerce')

    p_value = stats.ttest_ind(col1, col2, nan_policy='omit')[1]

    if p_value < 0.05:
        # Column is useful
        print("The column '{}' is useful with a p-value of {}".format(each_col,␣
 ↪p_value))
    else:
        # Column is not useful
        print("The column '{}' is not useful with a p-value of {}".
 ↪format(each_col, p_value))
```

```
The column 'credit.policy' is useful with a p-value of 0.0
The column 'purpose' is not useful with a p-value of 0.0
The column 'int.rate' is useful with a p-value of 2.1945619717925775e-190
The column 'installment' is useful with a p-value of 8.620919919293774e-09
The column 'log.annual.inc' is useful with a p-value of 0.0006337324172009995
The column 'dti' is useful with a p-value of 4.945272027510337e-19
The column 'fico' is useful with a p-value of 2.6100416830751396e-271
The column 'days.with.cr.line' is useful with a p-value of 2.627126143480234e-22
The column 'revol.bal' is useful with a p-value of 1.592869483344647e-76
The column 'revol.util' is useful with a p-value of 1.7179159037327666e-24
```

```
/usr/local/lib/python3.7/site-packages/scipy/stats/mstats_basic.py:1050:
RuntimeWarning: divide by zero encountered in true_divide
  denom = ma.sqrt(svar*(1.0/n1 + 1.0/n2))  # n-D computation here!
```

[28]:
```
cols_to_evaluate2 = ['inq.last.6mths', 'delinq.2yrs', 'pub.rec', 'not.fully.
 ↪paid']
```

Categorical usefulness

[29]:
```
from scipy.stats import chi2_contingency
```

[30]:
```
#Tests for the relationships

chi_res = chi2_contingency(pd.crosstab(data['credit.policy'],data['not.fully.
 ↪paid']))
```

[31]:
```
chi_res
```

[31]: (238.3788010698609,
 8.87573133930704e-54,
 1,
 array([[1569.01858426, 298.98141574],
        [6475.98141574, 1234.01858426]]))

[32]:
```
chi2_contingency(pd.crosstab(data['credit.policy'],data['not.fully.paid']))[1]
```

[32]: 8.87573133930704e-54

[33]:
```
for each_col in cols_to_evaluate2 :

    chi2_contingency(pd.crosstab(data['credit.policy'],data['not.fully.
↪paid']))[1]

    if p_value < 0.05:
        # Column is useful
        print("The column '{}' is useful with a p-value of {}".format(each_col,␣
↪p_value))
    else:
        # Column is not useful
        print("The column '{}' is not useful with a p-value of {}".
↪format(each_col, p_value))
```

```
The column 'inq.last.6mths' is useful with a p-value of 1.7179159037327666e-24
The column 'delinq.2yrs' is useful with a p-value of 1.7179159037327666e-24
The column 'pub.rec' is useful with a p-value of 1.7179159037327666e-24
The column 'not.fully.paid' is useful with a p-value of 1.7179159037327666e-24
```

encoding cat columns

[34]:
```
data_dummies= pd.get_dummies(data)
```

[35]:
```
data_dummies.sample(10)
```

[35]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	\
8206	0	0.1425	102.9000	10.9519	22.1700	657	
935	1	0.1008	161.5300	10.2036	12.7100	732	
4740	1	0.1287	571.7700	11.5229	17.2200	712	
7096	1	0.1311	404.9400	10.7013	22.4300	692	
1615	1	0.1367	204.1100	10.7096	22.7800	692	
419	1	0.0775	305.9700	11.0021	9.2000	797	
3106	1	0.1095	327.1400	10.4009	24.8800	727	
8284	0	0.1229	80.0500	9.1695	0.0000	662	
4494	1	0.1357	203.8200	10.9681	9.3700	692	
8671	0	0.1387	218.3400	10.6454	17.3700	657	

	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	\
8206	4949.0417	6710	54.6000	1	0	
935	3150.0000	11837	20.6000	0	0	
4740	7979.9583	26949	52.0000	0	0	
7096	5490.0000	10337	61.9000	0	0	
1615	8490.0417	19270	97.3000	1	0	
419	6270.0000	209	1.9000	0	0	
3106	3726.9583	10462	33.9000	1	0	
8284	539.0417	0	0.0000	5	0	
4494	1410.0000	902	20.0000	1	0	
8671	1710.0000	8312	85.7000	0	1	

	pub.rec	not.fully.paid	purpose_all_other	purpose_credit_card	\
8206	0	0	1	0	
935	0	0	0	0	
4740	0	0	0	0	
7096	0	0	0	0	
1615	1	0	0	1	
419	0	0	0	1	
3106	0	0	0	0	
8284	0	0	1	0	
4494	0	0	1	0	
8671	0	0	0	1	

	purpose_debt_consolidation	purpose_educational	\
8206	0	0	
935	1	0	
4740	1	0	
7096	1	0	
1615	0	0	
419	0	0	
3106	1	0	
8284	0	0	
4494	0	0	
8671	0	0	

	purpose_home_improvement	purpose_major_purchase	purpose_small_business
8206	0	0	0
935	0	0	0
4740	0	0	0
7096	0	0	0
1615	0	0	0
419	0	0	0
3106	0	0	0
8284	0	0	0
4494	0	0	0
8671	0	0	0

[36]:
```
data_dummies.sample(10).T
```

[36]:

	3394	7269	6585	347 \
credit.policy	1.0000	1.0000	1.0000	1.0000
int.rate	0.1126	0.1348	0.1183	0.1141
installment	328.6400	393.5200	795.2200	16.4700
log.annual.inc	11.2252	11.0186	11.7753	9.8782
dti	20.0600	10.9000	15.3800	3.6900
fico	717.0000	682.0000	812.0000	667.0000
days.with.cr.line	4529.9583	4560.0417	11705.0000	8820.0000
revol.bal	3404.0000	3715.0000	346.0000	12229.0000
revol.util	72.4000	26.5000	0.7000	90.6000
inq.last.6mths	2.0000	1.0000	0.0000	0.0000
delinq.2yrs	0.0000	0.0000	0.0000	0.0000
pub.rec	0.0000	0.0000	0.0000	1.0000
not.fully.paid	0.0000	0.0000	0.0000	0.0000
purpose_all_other	0.0000	0.0000	0.0000	1.0000
purpose_credit_card	0.0000	1.0000	0.0000	0.0000
purpose_debt_consolidation	1.0000	0.0000	1.0000	0.0000
purpose_educational	0.0000	0.0000	0.0000	0.0000
purpose_home_improvement	0.0000	0.0000	0.0000	0.0000
purpose_major_purchase	0.0000	0.0000	0.0000	0.0000
purpose_small_business	0.0000	0.0000	0.0000	0.0000

	4837	7371	5518	3755 \
credit.policy	1.0000	1.0000	1.0000	1.0000
int.rate	0.1218	0.1533	0.1322	0.1189
installment	599.4000	870.7100	507.0100	33.1700
log.annual.inc	11.6082	11.2772	10.3810	11.3737
dti	8.2300	13.9600	15.6300	5.5700
fico	722.0000	697.0000	707.0000	697.0000
days.with.cr.line	5429.9583	9721.0417	14879.9583	8519.9583
revol.bal	15533.0000	26018.0000	14489.0000	5742.0000
revol.util	79.3000	43.9000	89.4000	55.2000
inq.last.6mths	1.0000	0.0000	0.0000	0.0000
delinq.2yrs	0.0000	1.0000	0.0000	1.0000
pub.rec	0.0000	0.0000	0.0000	0.0000
not.fully.paid	0.0000	0.0000	0.0000	0.0000
purpose_all_other	0.0000	0.0000	0.0000	1.0000
purpose_credit_card	0.0000	0.0000	0.0000	0.0000
purpose_debt_consolidation	1.0000	1.0000	1.0000	0.0000
purpose_educational	0.0000	0.0000	0.0000	0.0000
purpose_home_improvement	0.0000	0.0000	0.0000	0.0000
purpose_major_purchase	0.0000	0.0000	0.0000	0.0000
purpose_small_business	0.0000	0.0000	0.0000	0.0000

	514	307

```
credit.policy                     1.0000     1.0000
int.rate                          0.1197     0.0743
installment                     199.2000   124.3000
log.annual.inc                   10.9819    11.0429
dti                               6.0000     5.8200
fico                            672.0000   772.0000
days.with.cr.line              4410.0000  5430.0000
revol.bal                      3831.0000   773.0000
revol.util                       34.2000     4.8000
inq.last.6mths                    0.0000     0.0000
delinq.2yrs                       1.0000     0.0000
pub.rec                           0.0000     0.0000
not.fully.paid                    1.0000     0.0000
purpose_all_other                 0.0000     0.0000
purpose_credit_card               0.0000     0.0000
purpose_debt_consolidation        1.0000     0.0000
purpose_educational               0.0000     0.0000
purpose_home_improvement          0.0000     1.0000
purpose_major_purchase            0.0000     0.0000
purpose_small_business            0.0000     0.0000
```

[37]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
```

[38]:
```
x_df = data.drop(['credit.policy'], axis=1)
y_df = data['credit.policy']
```

Building a Model

[39]:
```
data_transf = ColumnTransformer([('std-sclr',StandardScaler(),['credit.policy',␣
↪'purpose', 'int.rate', 'installment', 'log.annual.inc',
      'dti', 'fico', 'days.with.cr.line', 'revol.bal', 'revol.util'])],␣
↪remainder = 'passthrough')
```

0.0.1 Scaling

[40]:
```
x_df_encoded = pd.get_dummies(x_df, columns=['purpose'])

data_transf = StandardScaler()
```

[41]:
```
X_scaled = data_transf.fit_transform(x_df_encoded)
```

[42]:
```
y = y_df.values
```

Using multi-collinearity

[43]:
```
y = data['credit.policy'].values
```

[44]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
```

[45]:
```
X = data.drop(['credit.policy','purpose'],axis = 1)
```

[46]:
```
X.values
```

[46]:
```
array([[1.18900000e-01, 8.29100000e+02, 1.13504065e+01, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.07100000e-01, 2.28220000e+02, 1.10821426e+01, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.35700000e-01, 3.66860000e+02, 1.03734912e+01, …,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       …,
       [1.07100000e-01, 9.78100000e+01, 1.05966347e+01, …,
        0.00000000e+00, 0.00000000e+00, 1.00000000e+00],
       [1.60000000e-01, 3.51580000e+02, 1.08197783e+01, …,
        0.00000000e+00, 0.00000000e+00, 1.00000000e+00],
       [1.39200000e-01, 8.53430000e+02, 1.12644641e+01, …,
        0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

[47]:
```
mod = pd.DataFrame()
mod['VIF_Factor'] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]
mod['feature'] = X.columns
mod = mod.sort_values(by='VIF_Factor', ascending=False).round(5)

print(mod)
```

```
    VIF_Factor          feature
2     377.0961     log.annual.inc
4     278.9056               fico
0      35.2293           int.rate
7       5.6752         revol.util
5       5.2466  days.with.cr.line
3       5.0872                dti
1       4.2039        installment
8       1.6652     inq.last.6mths
6       1.5606          revol.bal
11      1.2471     not.fully.paid
9       1.2000        delinq.2yrs
10      1.1004            pub.rec
```

[48]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size= .25)
```

[49]:
```
ml_model = RandomForestClassifier()

ml_model.fit(X_train,y_train)

y_pred_train = ml_model.predict(X_train)
y_pred = ml_model.predict(X_test)

print("train_ing accuracy score:", metrics.accuracy_score(y_train,␣
 ↪y_pred_train))
print("train_ing accuracy score:", metrics.accuracy_score(y_test, y_pred))
```

```
train_ing accuracy score: 1.0
train_ing accuracy score: 0.9870563674321503
```

[50]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       457
           1       0.99      1.00      0.99      1938

    accuracy                           0.99      2395
   macro avg       0.99      0.97      0.98      2395
weighted avg       0.99      0.99      0.99      2395
```

[51]:
```
metrics.confusion_matrix(y_test, y_pred)
```

[51]:
```
array([[ 428,   29],
       [   2, 1936]])
```

Random Foresting the model

[52]:
```
num_trees_list = [1,2,3,4,5,7,10]
```

[53]:
```
train_ = []
test_ = []

for num_trees in num_trees_list:
    tres_= RandomForestClassifier(n_estimators = num_trees)

    tres_.fit(X_train,y_train)

    train_pred = tres_.predict(X_train)
```

```python
    order = metrics.accuracy_score(y_train, train_pred)

    train_.append(order)
    test_pred = tres_.predict(X_test)

    order = metrics.accuracy_score(y_test, test_pred)

    test_.append(order)
```

[54]:
```python
num_estimators = [3,5,7,9,11,13,15,17,19,21]
max_depths = [2,4,6,8,10,12,14]
min_samples_split = np.linspace(.01,.05, 6)
```

[55]:
```python
parameters = dict(

                max_depths = max_depths,
                min_samples_split = min_samples_split,
                estimatead = num_estimators
        )

print(parameters)
```

```
{'max_depths': [2, 4, 6, 8, 10, 12, 14], 'min_samples_split': array([0.01 ,
0.018, 0.026, 0.034, 0.042, 0.05 ]), 'estimatead': [3, 5, 7, 9, 11, 13, 15, 17,
19, 21]}
```

[56]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
```

[57]:
```python
model = RandomForestClassifier()
```

[58]:
```python
parameters = {
    'n_estimators': [10, 50, 100],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}


Grid = GridSearchCV(
    estimator=model,
    param_grid=parameters,
    scoring='neg_mean_squared_error',
    cv=5,
    verbose=1
)
Grid.fit(X_scaled, y)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
```

[58]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [None, 5, 10],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [10, 50, 100]},
             scoring='neg_mean_squared_error', verbose=1)
```

[59]:
```python
Grid.best_params_
```

[59]:
```
{'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 100}
```

[60]:
```python
ml_model = RandomForestClassifier(
    max_depth=None,
    min_samples_leaf=1,
    min_samples_split=5,
    n_estimators=50
    #Balanced
)


ml_model.fit(X_train,y_train)

y_pred_train = ml_model.predict(X_train)
y_pred = ml_model.predict(X_test)

print("train_ing accuracy score:", metrics.accuracy_score(y_train,␣
 ↪y_pred_train))
print("train_ing accuracy score:", metrics.accuracy_score(y_test, y_pred))
```

```
train_ing accuracy score: 0.9979117360434359
train_ing accuracy score: 0.9870563674321503
```

[61]:
```python
metrics.confusion_matrix(y_test, y_pred)
```

[61]:
```
array([[ 429,   28],
       [   3, 1935]])
```

[66]:
```python
from sklearn.linear_model import LogisticRegression
```

[67]:
```python
ml_model = LogisticRegression()
```

```
ml_model.fit(X_train,y_train)

y_pred_train = ml_model.predict(X_train)
y_pred = ml_model.predict(X_test)

print("train_ing accuracy score:", metrics.accuracy_score(y_train,␣
 ↪y_pred_train))
print("train_ing accuracy score:", metrics.accuracy_score(y_test, y_pred))
```

```
train_ing accuracy score: 0.9050535987748851
train_ing accuracy score: 0.9006263048016702
```

Building a model using deep learning

[68]:
```
from keras.models import Sequential
from keras.layers import Dense
```

[69]:
```
X_train.shape
```

[69]:
```
(7183, 19)
```

[70]:
```
model = Sequential()

model.add(Dense(512, input_dim=19, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

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

[71]:
```
from keras.callbacks import EarlyStopping
```

[72]:
```
E_stop = EarlyStopping(monitor='val_loss', mode ='min',patience= 5, verbose= 1)
```

[73]:
```
%%time

history = model.fit(
                    X_train,
                    y_train,
                    epochs= 100,
                    batch_size=25,
```

```
                     validation_data= (X_test,y_test),
                     verbose=1,
                     callbacks=[E_stop]
)
```

```
Epoch 1/100
288/288 [==============================] - 7s 5ms/step - loss: 0.2868 -
accuracy: 0.8881 - val_loss: 0.2806 - val_accuracy: 0.9027
Epoch 2/100
288/288 [==============================] - 1s 4ms/step - loss: 0.2224 -
accuracy: 0.9120 - val_loss: 0.2127 - val_accuracy: 0.9161
Epoch 3/100
288/288 [==============================] - 1s 4ms/step - loss: 0.1892 -
accuracy: 0.9247 - val_loss: 0.1953 - val_accuracy: 0.9223
Epoch 4/100
288/288 [==============================] - 1s 4ms/step - loss: 0.1690 -
accuracy: 0.9367 - val_loss: 0.1522 - val_accuracy: 0.9424
Epoch 5/100
288/288 [==============================] - 1s 4ms/step - loss: 0.1482 -
accuracy: 0.9415 - val_loss: 0.1393 - val_accuracy: 0.9549
Epoch 6/100
288/288 [==============================] - 1s 4ms/step - loss: 0.1279 -
accuracy: 0.9522 - val_loss: 0.1464 - val_accuracy: 0.9407
Epoch 7/100
288/288 [==============================] - 1s 4ms/step - loss: 0.1159 -
accuracy: 0.9564 - val_loss: 0.1216 - val_accuracy: 0.9541
Epoch 8/100
288/288 [==============================] - 1s 4ms/step - loss: 0.1099 -
accuracy: 0.9589 - val_loss: 0.1168 - val_accuracy: 0.9570
Epoch 9/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0991 -
accuracy: 0.9635 - val_loss: 0.1190 - val_accuracy: 0.9557
Epoch 10/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0948 -
accuracy: 0.9667 - val_loss: 0.1060 - val_accuracy: 0.9599
Epoch 11/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0857 -
accuracy: 0.9719 - val_loss: 0.1090 - val_accuracy: 0.9608
Epoch 12/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0778 -
accuracy: 0.9710 - val_loss: 0.1154 - val_accuracy: 0.9516
Epoch 13/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0767 -
accuracy: 0.9720 - val_loss: 0.1017 - val_accuracy: 0.9670
Epoch 14/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0618 -
accuracy: 0.9765 - val_loss: 0.1019 - val_accuracy: 0.9587
```

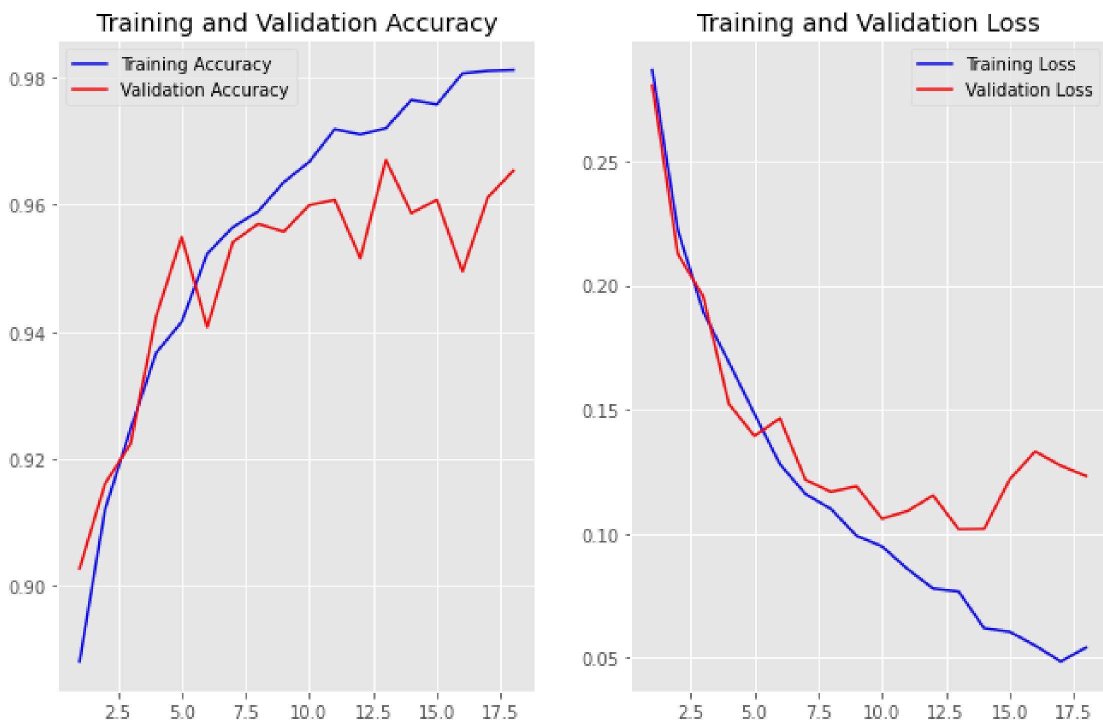
```
Epoch 15/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0604 -
accuracy: 0.9758 - val_loss: 0.1218 - val_accuracy: 0.9608
Epoch 16/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0548 -
accuracy: 0.9806 - val_loss: 0.1331 - val_accuracy: 0.9495
Epoch 17/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0484 -
accuracy: 0.9811 - val_loss: 0.1273 - val_accuracy: 0.9612
Epoch 18/100
288/288 [==============================] - 1s 4ms/step - loss: 0.0541 -
accuracy: 0.9812 - val_loss: 0.1231 - val_accuracy: 0.9653
Epoch 18: early stopping
CPU times: user 35.1 s, sys: 4.9 s, total: 40 s
Wall time: 27.9 s
```

[74]:

```
plt.style.use('ggplot')

def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    x = range(1, len(acc) + 1)

    plt.figure(figsize=(11, 7))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training Accuracy')
    plt.plot(x, val_acc, 'r', label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training Loss')
    plt.plot(x, val_loss, 'r', label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.legend()
```

[75]:

```
plot_history(history)
```



[76]:
```
y_pred = np.squeeze((y_pred > .5 ).astype('int32'))
```

[77]:
```
from sklearn import metrics
```

[78]:
```
metrics.accuracy_score(y_test, y_pred)
```

[78]: 0.9006263048016702

[ ]:
```
```