Loan default India

Through this dataset we are trying to predict that which users are more likely to default on a loan thye borrowed. This is based in the Indian context and people from other parts of the world might find a few things hard to understand. We are tryig to predict which factors are good at pointing out that which particular person is more likely to commit a fraud.We are going to use classifier algorithms for this purpose.
We are going to be studying the historical data in order to find patterns that indicate a fraud. Then we will apply machine learning techniques to create a model that will identify a fraud.

STEP 1:Setting up the environment

In this part we will load the necessary packages and the dataset. We will have a quick overview of the variables.

In [1]:
```
#importing packages
from pandas import Series;  from numpy.random import randn
from statsmodels.stats.weightstats import ttest_ind
import scipy.stats as stats
from scipy.stats import ttest_ind
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
import seaborn as sb
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from matplotlib import pyplot as plt
from scipy.stats import shapiro
from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from datetime import datetime
from datetime import date
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression as lgr
from mpl_toolkits.mplot3d import Axes3D
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing as preproc
from sklearn import metrics
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, auc, balanced_accuracy_score, confusion_matrix, f1_score, precision
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, IsolationForest, RandomForestClassifier
from sklearn.model_selection import cross_val_score, cross_val_predict, cross_validate, train_test_split
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.svm import SVC
```

In [2]:
```
#downloading the file
ldi= pd.read_excel("C:\\Users\\sujoydutta\\Desktop\\Data analysis\\Datasets for ML\\Classifier\\loan_dataset_in
ldi.head()
```

Out[2]:

	UniqueID	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Current_pincode_ID	Date.of.Birth	Age	...	SE
0	420825	50578	58400	89.55	67	22807	45	1441	1984-01-01	39.271233	...	
1	417566	53278	61360	89.63	67	22807	45	1497	1985-08-24	37.624658	...	
2	539055	52378	60300	88.39	67	22807	45	1495	1977-12-09	45.336986	...	
3	529269	46349	61500	76.42	67	22807	45	1502	1988-06-01	34.852055	...	
4	563215	43594	78256	57.50	67	22744	86	1499	1994-07-14	28.731507	...	

5 rows × 42 columns

In [3]:
```
#examining the dataset
ldi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 233154 entries, 0 to 233153
Data columns (total 42 columns):
 #   Column                               Non-Null Count   Dtype
---  ------                               --------------   -----
 0   UniqueID                             233154 non-null  int64
 1   disbursed_amount                     233154 non-null  int64
 2   asset_cost                           233154 non-null  int64
 3   ltv                                  233154 non-null  float64
 4   branch_id                            233154 non-null  int64
 5   supplier_id                          233154 non-null  int64
 6   manufacturer_id                      233154 non-null  int64
 7   Current_pincode_ID                   233154 non-null  int64
 8   Date.of.Birth                        233154 non-null  datetime64[ns]
 9   Age                                  233154 non-null  float64
 10  Employment.Type                      233154 non-null  object
 11  DisbursalDate                        233154 non-null  datetime64[ns]
 12  State_ID                             233154 non-null  int64
 13  Employee_code_ID                     233154 non-null  int64
 14  MobileNo_Avl_Flag                    233154 non-null  int64
 15  Aadhar_flag                          233154 non-null  int64
 16  PAN_flag                             233154 non-null  int64
 17  VoterID_flag                         233154 non-null  int64
 18  Driving_flag                         233154 non-null  int64
 19  Passport_flag                        233154 non-null  int64
 20  Profile_strength                     233154 non-null  object
 21  PERFORM_CNS.SCORE                    233154 non-null  int64
 22  PERFORM_CNS.SCORE.DESCRIPTION        233154 non-null  object
 23  PRI.NO.OF.ACCTS                      233154 non-null  int64
 24  PRI.ACTIVE.ACCTS                     233154 non-null  int64
 25  PRI.OVERDUE.ACCTS                    233154 non-null  int64
 26  PRI.CURRENT.BALANCE                  233154 non-null  int64
 27  PRI.SANCTIONED.AMOUNT                233154 non-null  int64
 28  PRI.DISBURSED.AMOUNT                 233154 non-null  int64
 29  SEC.NO.OF.ACCTS                      233154 non-null  int64
 30  SEC.ACTIVE.ACCTS                     233154 non-null  int64
 31  SEC.OVERDUE.ACCTS                    233154 non-null  int64
 32  SEC.CURRENT.BALANCE                  233154 non-null  int64
 33  SEC.SANCTIONED.AMOUNT                233154 non-null  int64
 34  SEC.DISBURSED.AMOUNT                 233154 non-null  int64
 35  PRIMARY.INSTAL.AMT                   233154 non-null  int64
 36  SEC.INSTAL.AMT                       233154 non-null  int64
 37  NEW.ACCTS.IN.LAST.SIX.MONTHS         233154 non-null  int64
 38  DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS  233154 non-null  int64
 39  Acc_age_months                       233154 non-null  int64
 40  NO.OF_INQUIRIES                      233154 non-null  int64
 41  loan_default                         233154 non-null  int64
dtypes: datetime64[ns](2), float64(2), int64(35), object(3)
memory usage: 74.7+ MB
```

In [4]:
```
#examining the dataset
ldi.shape
```

Out[4]:
```
(233154, 42)
```

STEP 2:Data cleaning and organizing

In this step, we are going to clean our dataset. We are going to look for null values and replace them with mean and mode. We are going to modify some variables if it is necessary and change datatypes for better analysis. We will also remove outliers from the dataset. Outliers hamper the machine learning algorithms and hence they have to be removed.

In [5]:
```
# column name cleaning

ldi.columns = ldi.columns.str.replace(' ', '')
ldi.columns
```

Out[5]:
```
Index(['UniqueID', 'disbursed_amount', 'asset_cost', 'ltv', 'branch_id',
       'supplier_id', 'manufacturer_id', 'Current_pincode_ID', 'Date.of.Birth',
       'Age', 'Employment.Type', 'DisbursalDate', 'State_ID',
       'Employee_code_ID', 'MobileNo_Avl_Flag', 'Aadhar_flag', 'PAN_flag',
       'VoterID_flag', 'Driving_flag', 'Passport_flag', 'Profile_strength',
       'PERFORM_CNS.SCORE', 'PERFORM_CNS.SCORE.DESCRIPTION', 'PRI.NO.OF.ACCTS',
       'PRI.ACTIVE.ACCTS', 'PRI.OVERDUE.ACCTS', 'PRI.CURRENT.BALANCE',
       'PRI.SANCTIONED.AMOUNT', 'PRI.DISBURSED.AMOUNT', 'SEC.NO.OF.ACCTS',
       'SEC.ACTIVE.ACCTS', 'SEC.OVERDUE.ACCTS', 'SEC.CURRENT.BALANCE',
       'SEC.SANCTIONED.AMOUNT', 'SEC.DISBURSED.AMOUNT', 'PRIMARY.INSTAL.AMT',
       'SEC.INSTAL.AMT', 'NEW.ACCTS.IN.LAST.SIX.MONTHS',
       'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS', 'Acc_age_months',
       'NO.OF_INQUIRIES', 'loan_default'],
      dtype='object')
```

In [6]:
```
#checking null values
ldi.isna().sum(axis=0)
```

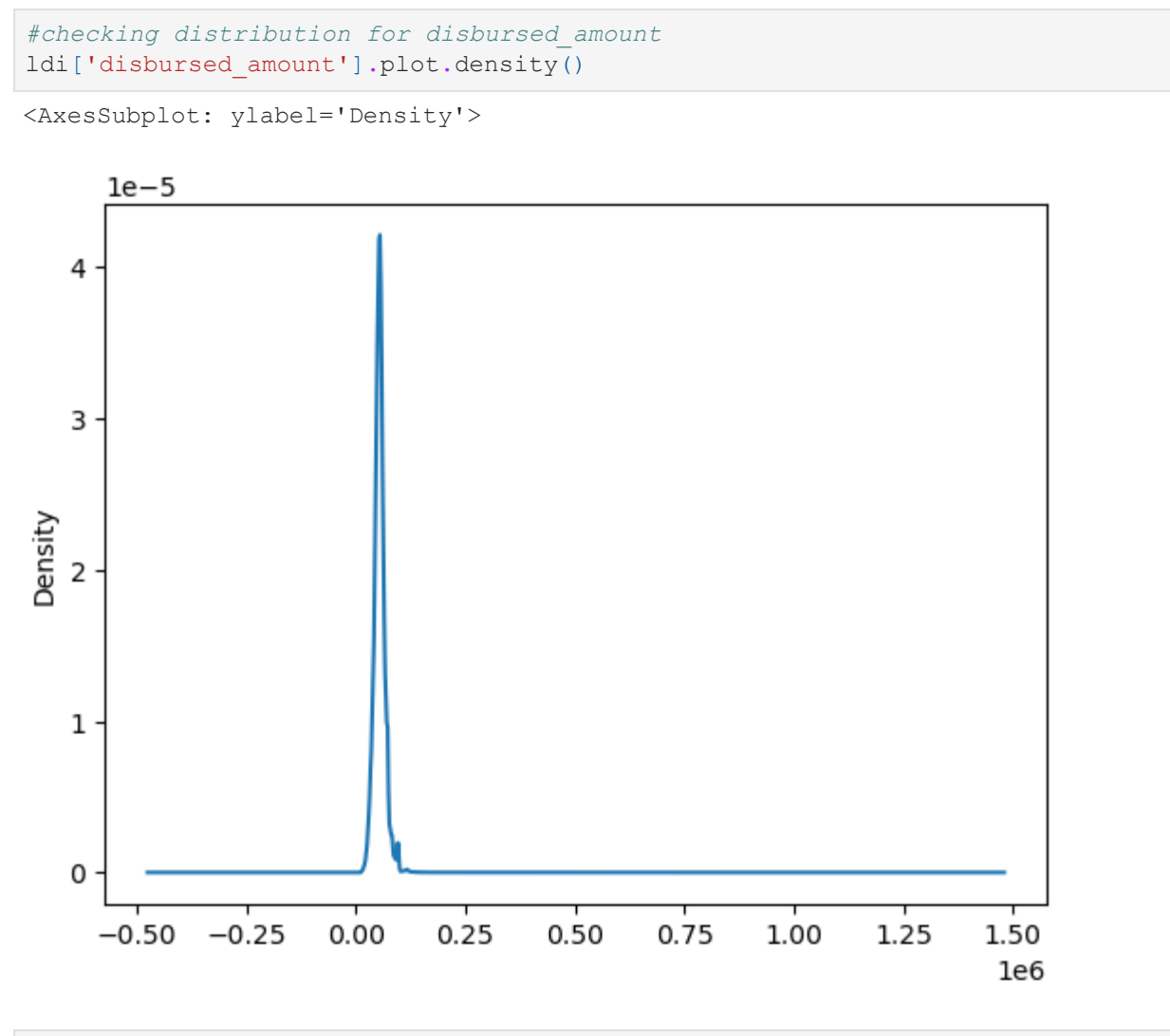
Out[6]:
```
UniqueID                               0
disbursed_amount                       0
asset_cost                             0
ltv                                    0
branch_id                              0
supplier_id                            0
manufacturer_id                        0
Current_pincode_ID                     0
Date.of.Birth                          0
Age                                    0
Employment.Type                        0
DisbursalDate                          0
State_ID                               0
Employee_code_ID                       0
MobileNo_Avl_Flag                      0
Aadhar_flag                            0
PAN_flag                               0
VoterID_flag                           0
Driving_flag                           0
Passport_flag                          0
Profile_strength                       0
PERFORM_CNS.SCORE                      0
PERFORM_CNS.SCORE.DESCRIPTION          0
PRI.NO.OF.ACCTS                        0
PRI.ACTIVE.ACCTS                       0
PRI.OVERDUE.ACCTS                      0
PRI.CURRENT.BALANCE                    0
PRI.SANCTIONED.AMOUNT                  0
PRI.DISBURSED.AMOUNT                   0
SEC.NO.OF.ACCTS                        0
SEC.ACTIVE.ACCTS                       0
SEC.OVERDUE.ACCTS                      0
SEC.CURRENT.BALANCE                    0
SEC.SANCTIONED.AMOUNT                  0
SEC.DISBURSED.AMOUNT                   0
PRIMARY.INSTAL.AMT                     0
SEC.INSTAL.AMT                         0
NEW.ACCTS.IN.LAST.SIX.MONTHS           0
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS    0
Acc_age_months                         0
NO.OF_INQUIRIES                        0
loan_default                           0
dtype: int64
```

Univariate analysis

We are using boxplot,histogram and plot to find the outliers,extreme values and distribution of the numerical variables.

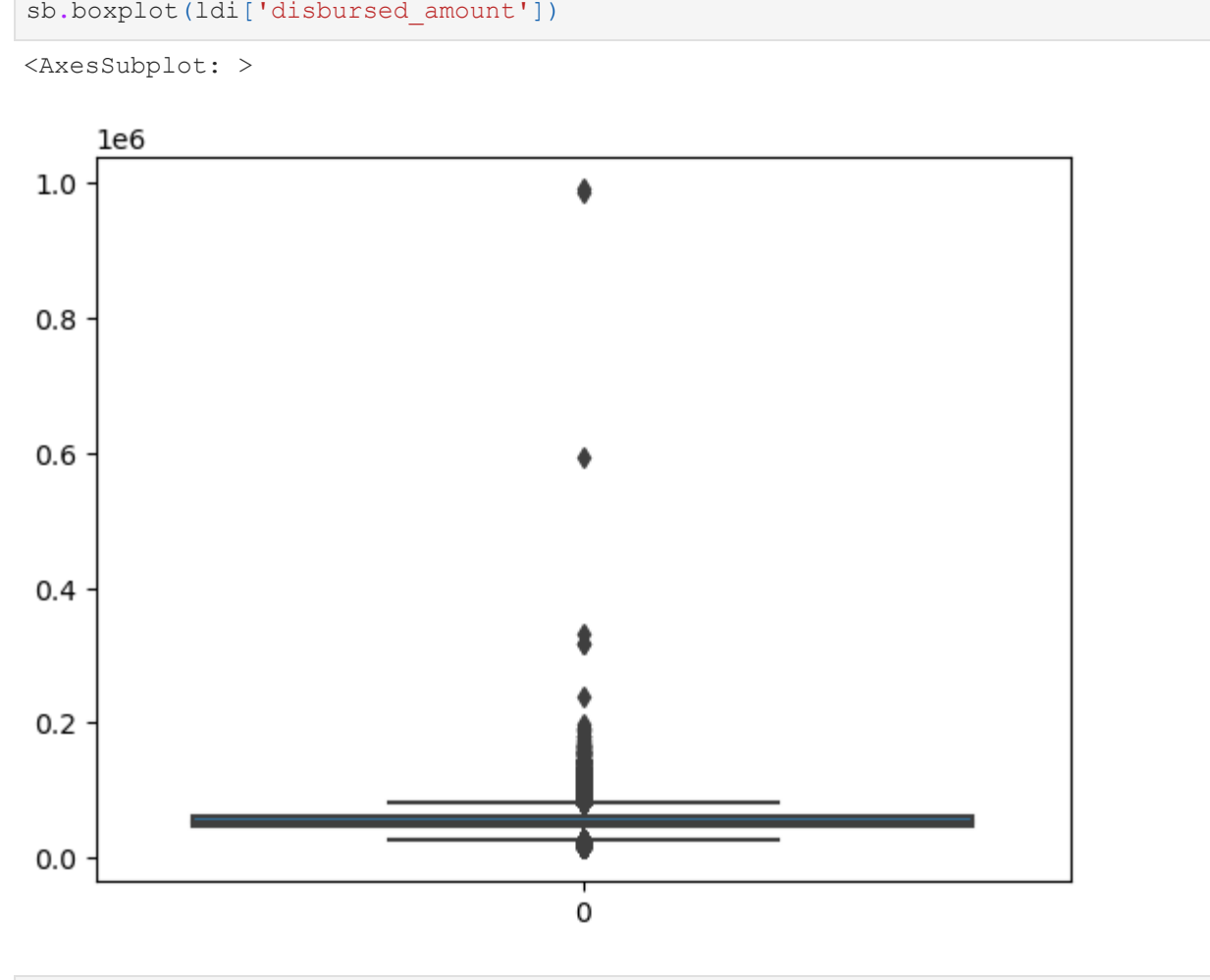
In [7]:
```
#checking distribution for disbursed_amount
ldi['disbursed_amount'].plot.density()
```

Out[7]: <AxesSubplot: ylabel='Density'>



In [8]:
```
#checking outliers using boxplot disbursed_amount
sb.boxplot(ldi['disbursed_amount'])
```

Out[8]: <AxesSubplot: >



In [9]:
```
#outlier treatment
ldi['disbursed_amount'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[9]:
```
0.10    39794.00
0.25    47145.00
0.50    53803.00
0.70    58759.00
0.90    68882.00
0.95    74122.35
0.99    95391.00
Name: disbursed_amount, dtype: float64
```

In [10]:
```
#seeing limits
disbursed_amount_HE=ldi[ldi['disbursed_amount'] > 68882.00].copy()
disbursed_amount_LE=ldi[ldi['disbursed_amount']< 39794.00].copy()
```

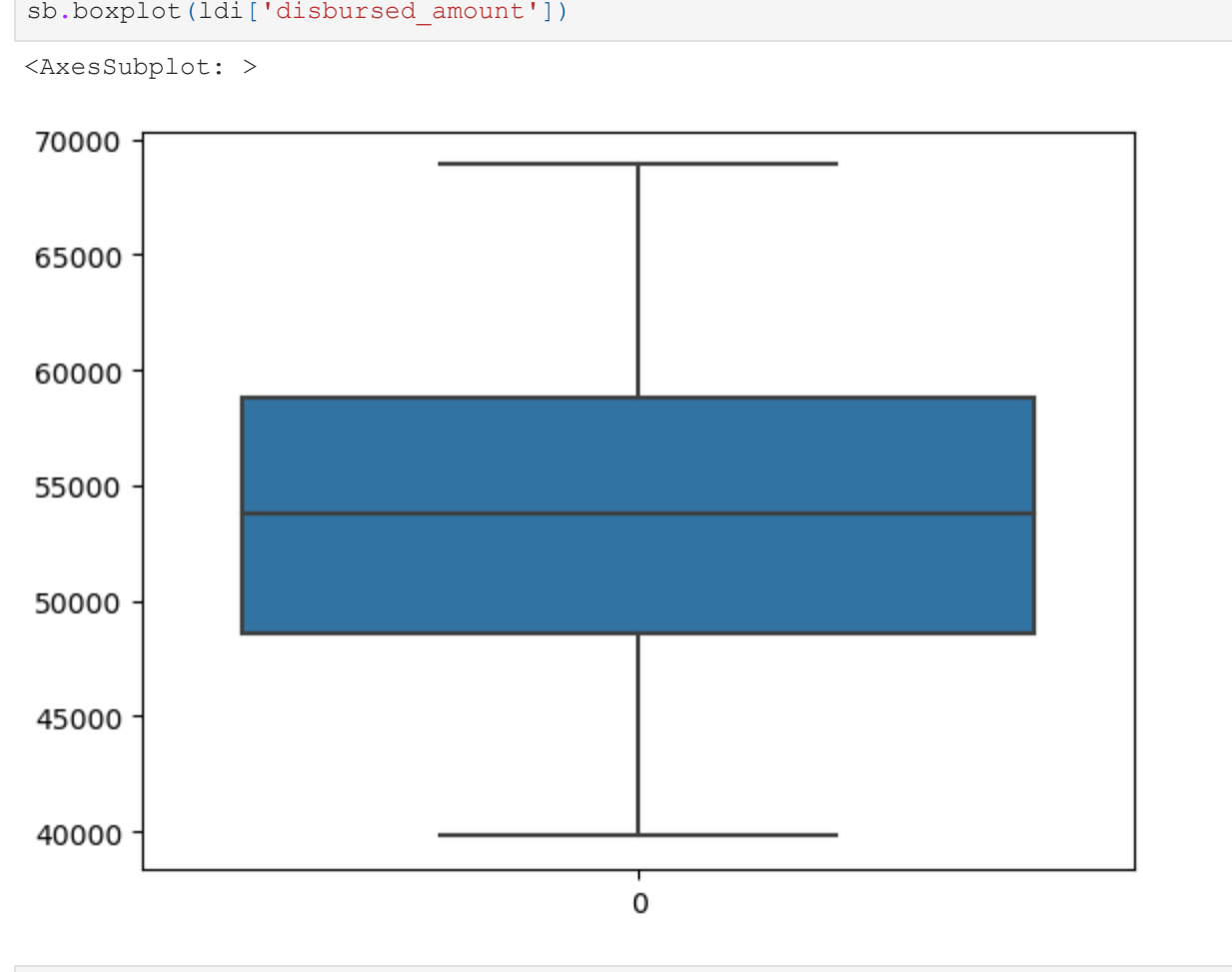
In [11]:
```
#putting them at one place
print(len(disbursed_amount_HE))
print(len(disbursed_amount_LE))
```

```
23293
23315
```

In [12]:
```
#removing outliers
for x in ldi['disbursed_amount']:
    if x > 68882.00:
        ldi['disbursed_amount'].replace(x,np.nan,inplace=True)
for x in ldi['disbursed_amount']:
    if x < 39794.00:
        ldi['disbursed_amount'].replace(x,np.nan,inplace=True)
```

In [13]:
```
#checking outliers gone or not
sb.boxplot(ldi['disbursed_amount'])
```

Out[13]: <AxesSubplot: >

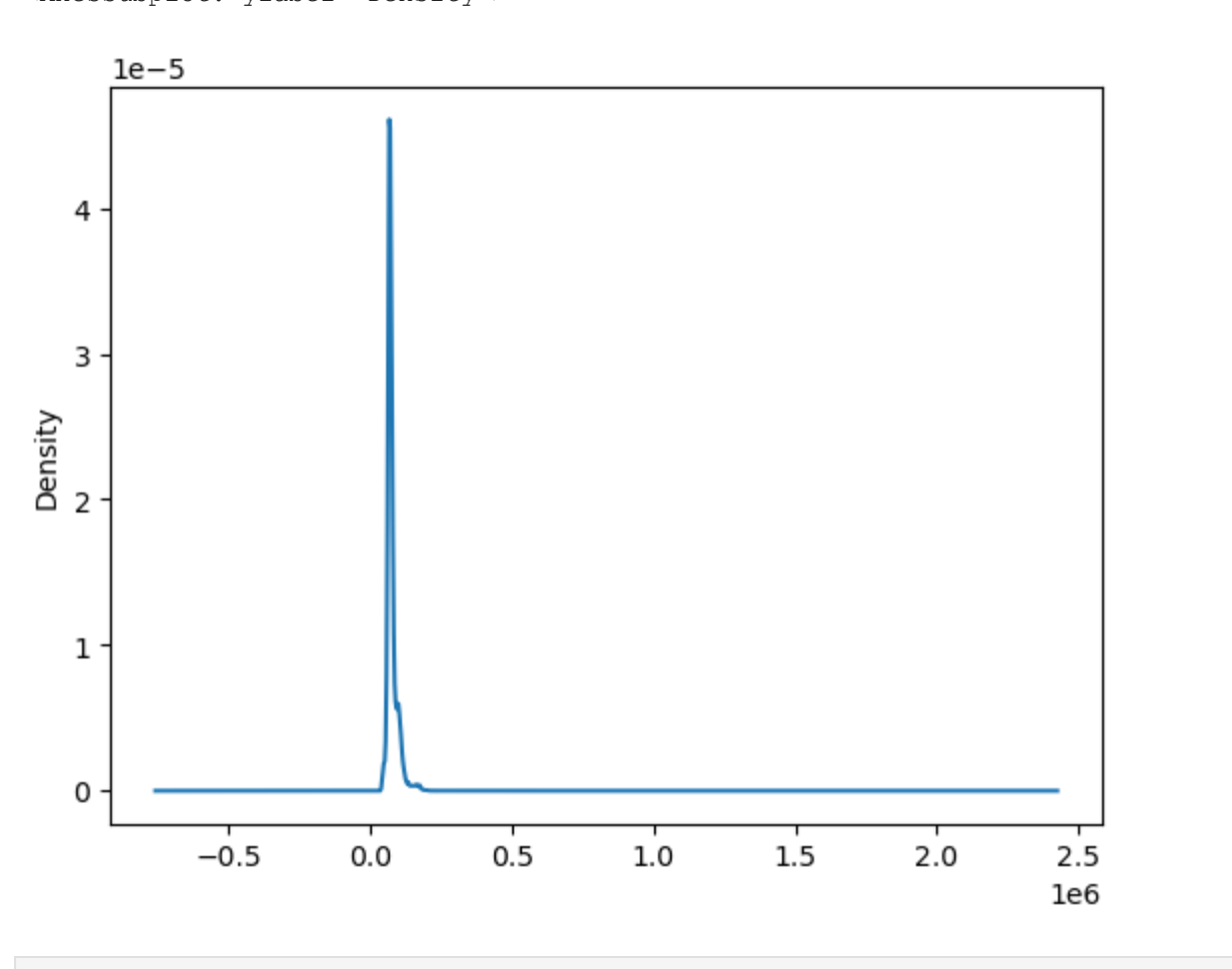


In [14]:
```
#replacing with median values

medda=ldi['disbursed_amount'].median()
print(medda)
ldi['disbursed_amount'].replace(np.nan, medda, inplace=True)
```

```
53803.0
```

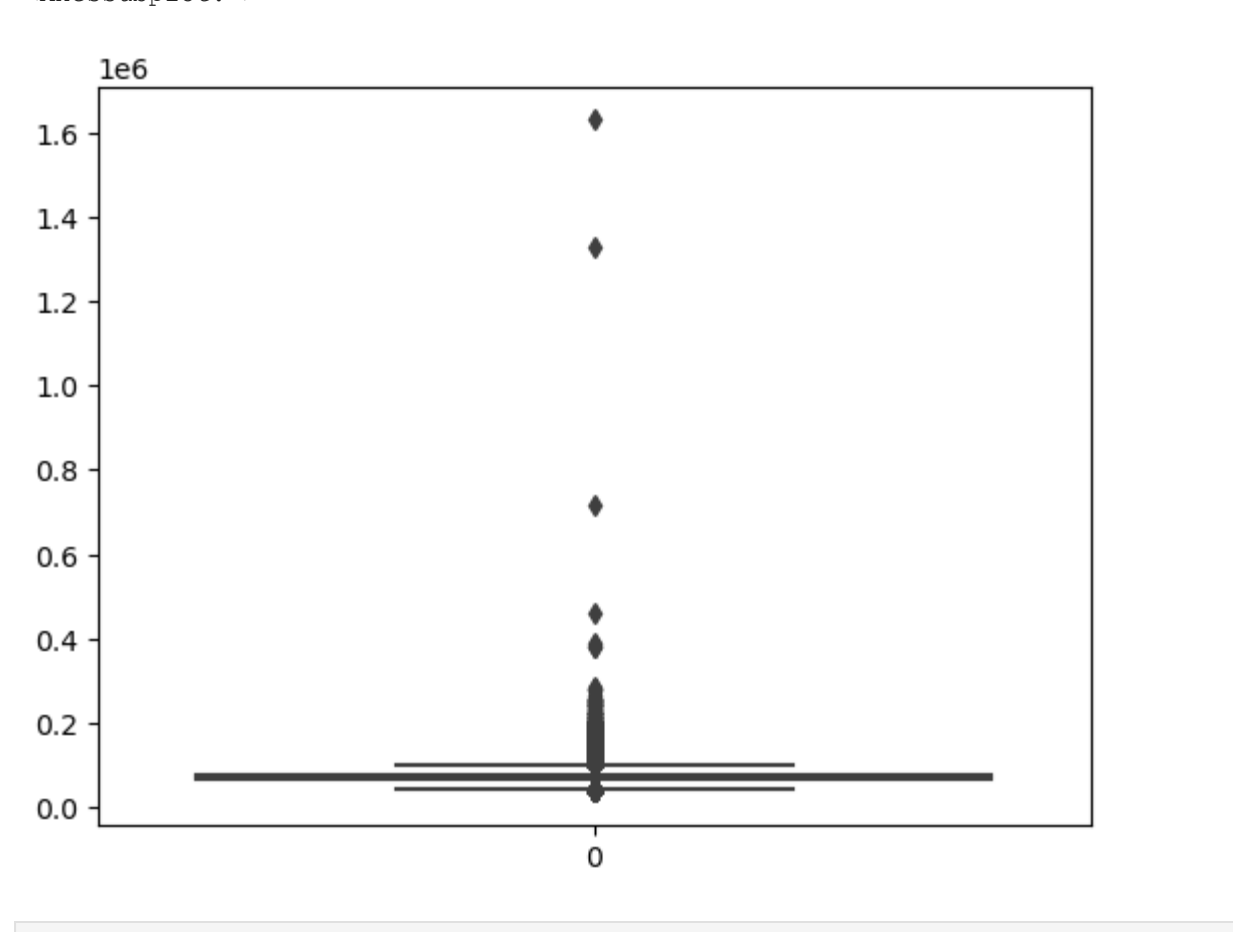
In [15]:
```
#checking distribution for asset_cost
ldi['asset_cost'].plot.density()
```

Out[15]: <AxesSubplot: ylabel='Density'>



In [16]:
```
#checking outliers using boxplot asset_cost
sb.boxplot(ldi['asset_cost'])
```

Out[16]: <AxesSubplot: >



In [17]:
```
#outlier treatment
ldi['asset_cost'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[17]:
```
0.10     61330.0
0.25     65717.0
0.50     70946.0
0.70     76807.0
0.90     99140.0
0.95    109680.0
0.99    156016.0
Name: asset_cost, dtype: float64
```

In [18]:
```
#seeing limits
asset_cost_HE=ldi[ldi['asset_cost'] > 76807.0].copy()
asset_cost_LE=ldi[ldi['asset_cost']< 61330.0].copy()
```

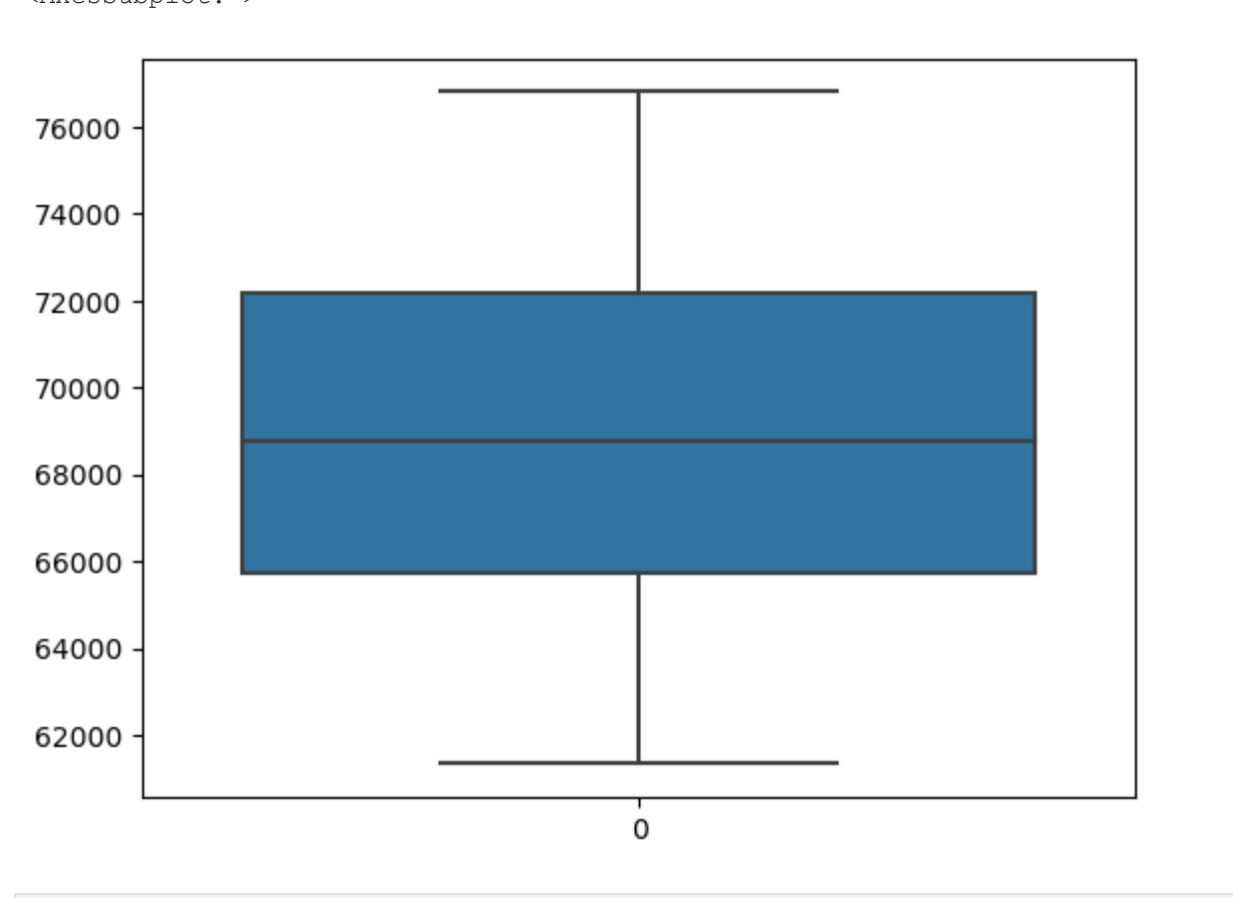
In [19]:
```
#putting them at one place
print(len(asset_cost_HE))
print(len(asset_cost_LE))
```

```
69944
23302
```

In [20]:
```
#removing outliers
for x in ldi['asset_cost']:
    if x > 76807.0:
        ldi['asset_cost'].replace(x,np.nan,inplace=True)
for x in ldi['asset_cost']:
    if x < 61330.0:
        ldi['asset_cost'].replace(x,np.nan,inplace=True)
```

In [21]:
```
#checking outliers gone or not
sb.boxplot(ldi['asset_cost'])
```

Out[21]: <AxesSubplot: >

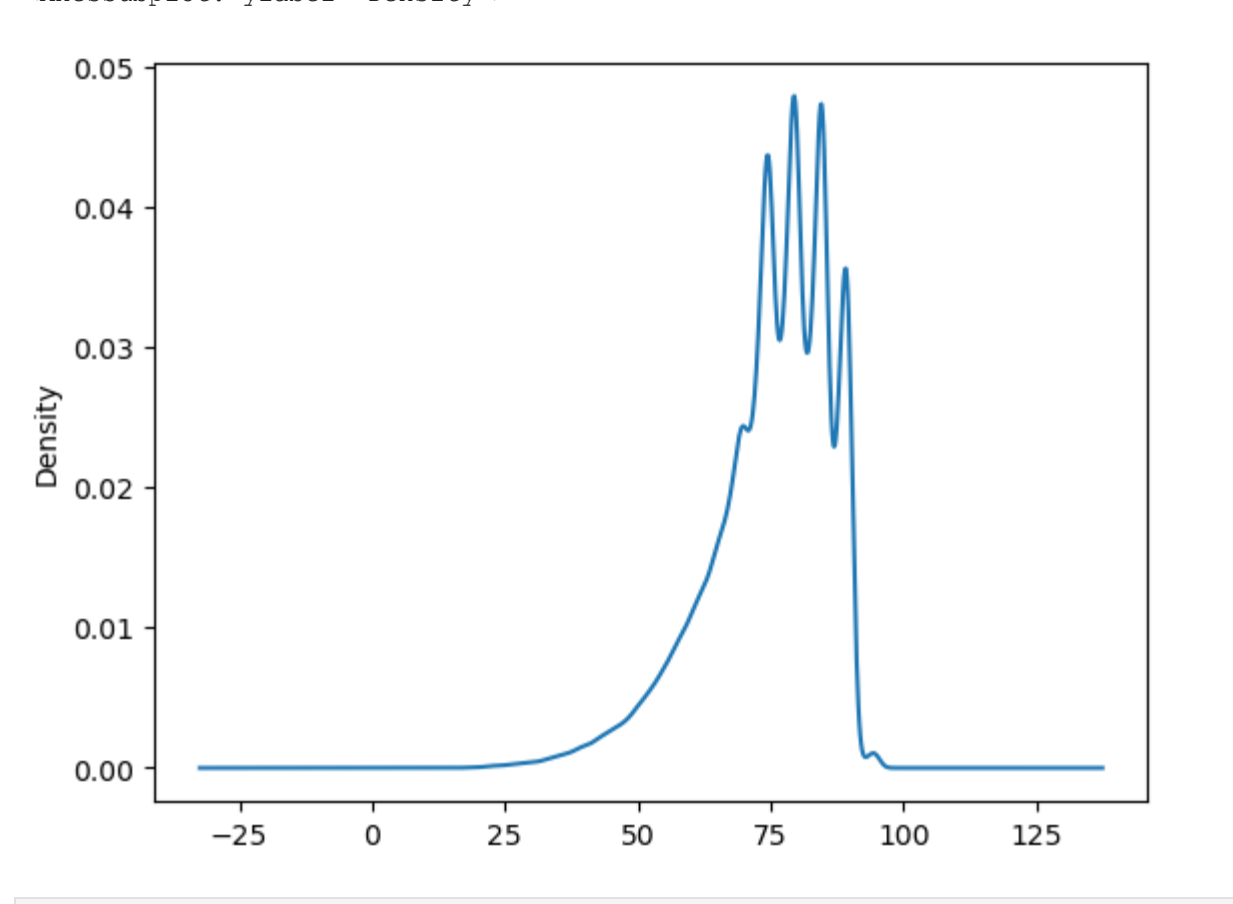


In [22]:
```
#replacing with median values

medac=ldi['asset_cost'].median()
print(medac)
ldi['asset_cost'].replace(np.nan, medac, inplace=True)
```

```
68763.0
```

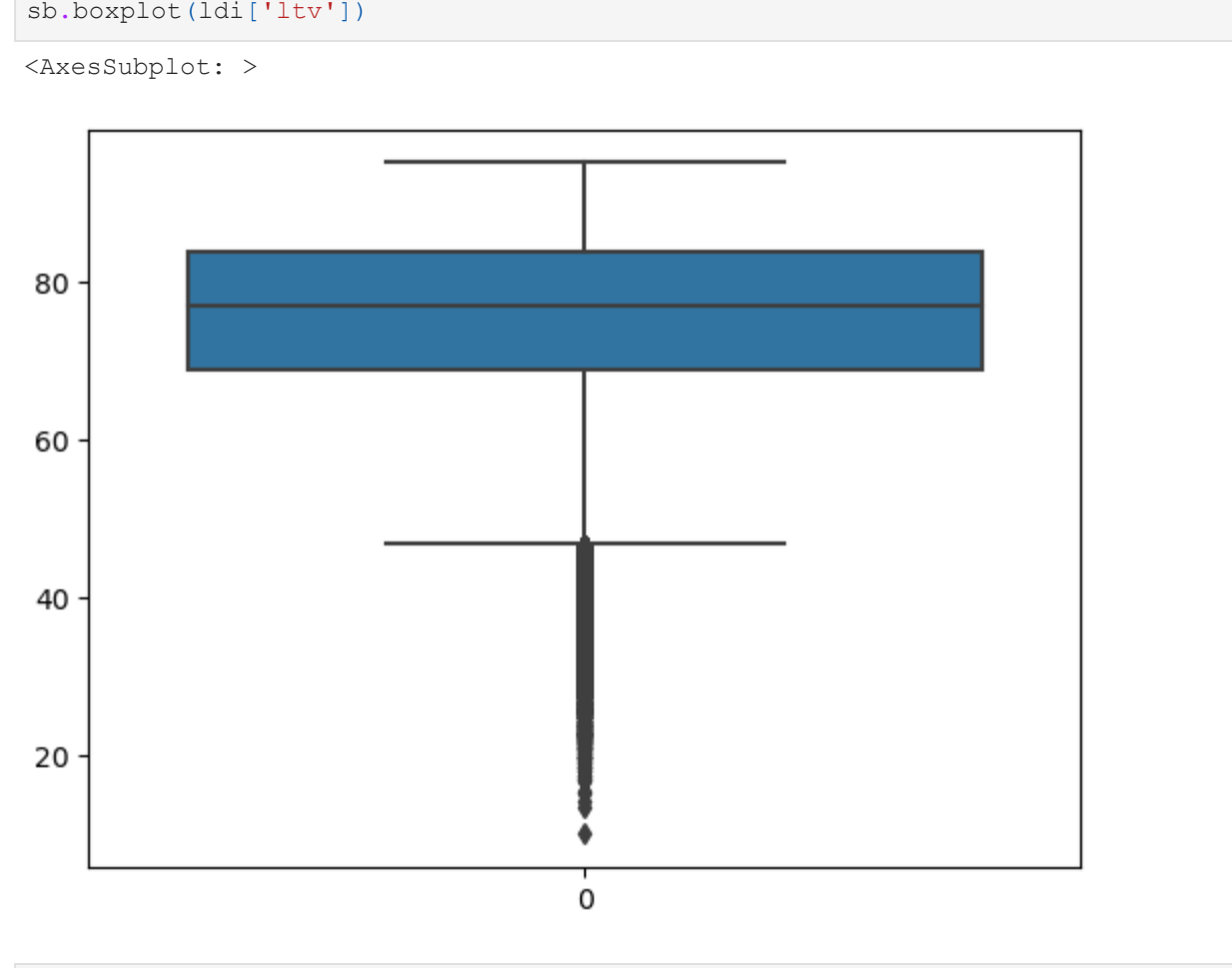
In [23]:
```
#checking distribution for ltv
ldi['ltv'].plot.density()
```

Out[23]: <AxesSubplot: ylabel='Density'>



In [24]:
```
#checking outliers using boxplot ltv
sb.boxplot(ldi['ltv'])
```

Out[24]: <AxesSubplot: >



In [25]:
```
#outlier treatment
ldi['ltv'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[25]:
```
0.10    58.86
0.25    68.88
0.50    76.80
0.70    82.01
0.90    87.95
0.95    89.38
0.99    89.95
Name: ltv, dtype: float64
```

In [26]:
```
#seeing limits
ltv_HE=ldi[ldi['ltv'] > 89.95].copy()
ltv_LE=ldi[ldi['ltv']< 68.88].copy()
```

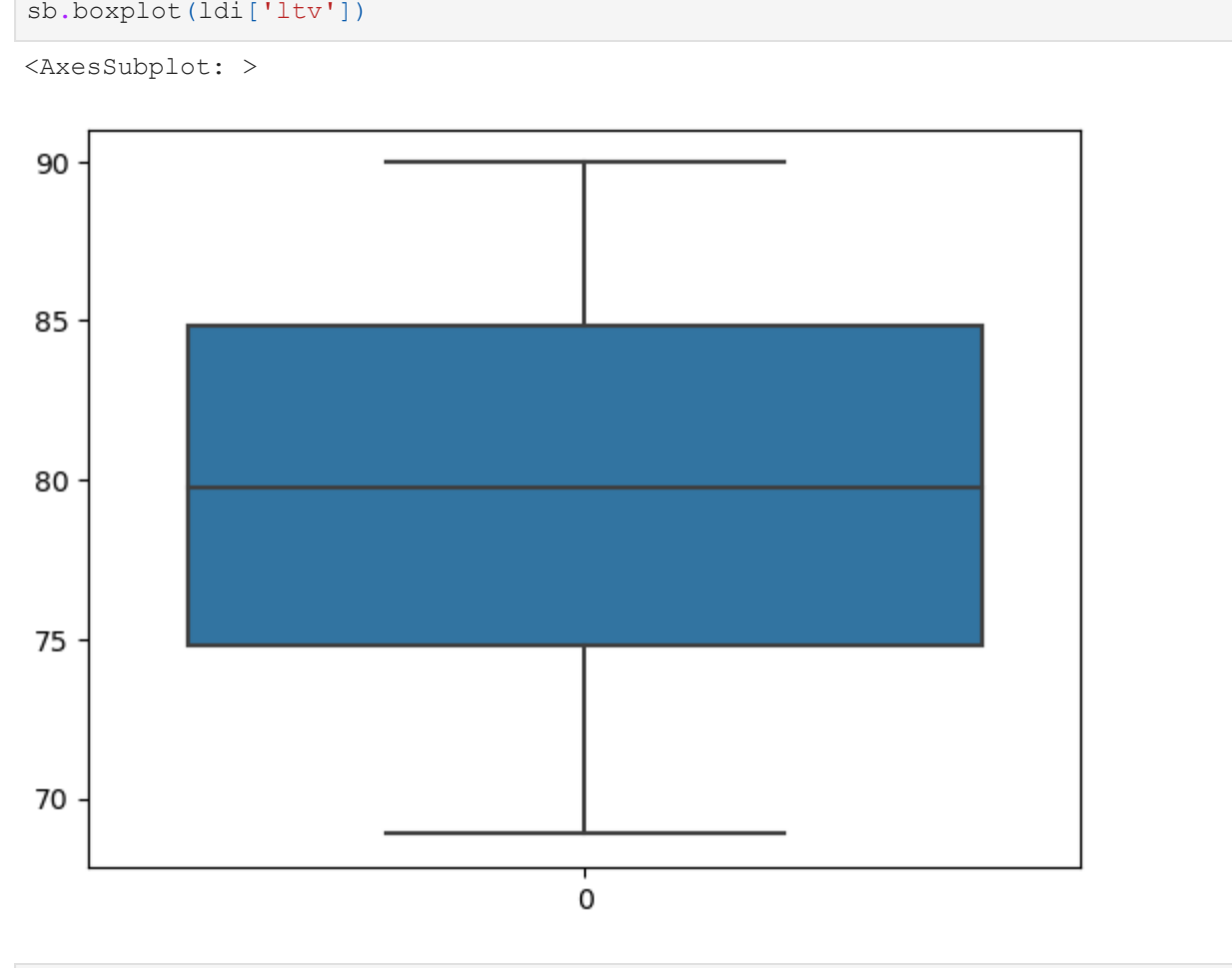
In [27]:
```
#putting them at one place
print(len(ltv_HE))
print(len(ltv_LE))
```

```
2235
58241
```

In [28]:
```
#removing outliers
for x in ldi['ltv']:
    if x > 89.95:
        ldi['ltv'].replace(x,np.nan,inplace=True)
for x in ldi['ltv']:
    if x < 68.88:
        ldi['ltv'].replace(x,np.nan,inplace=True)
```

In [29]:
```
#checking outliers gone or not
sb.boxplot(ldi['ltv'])
```

Out[29]: <AxesSubplot: >

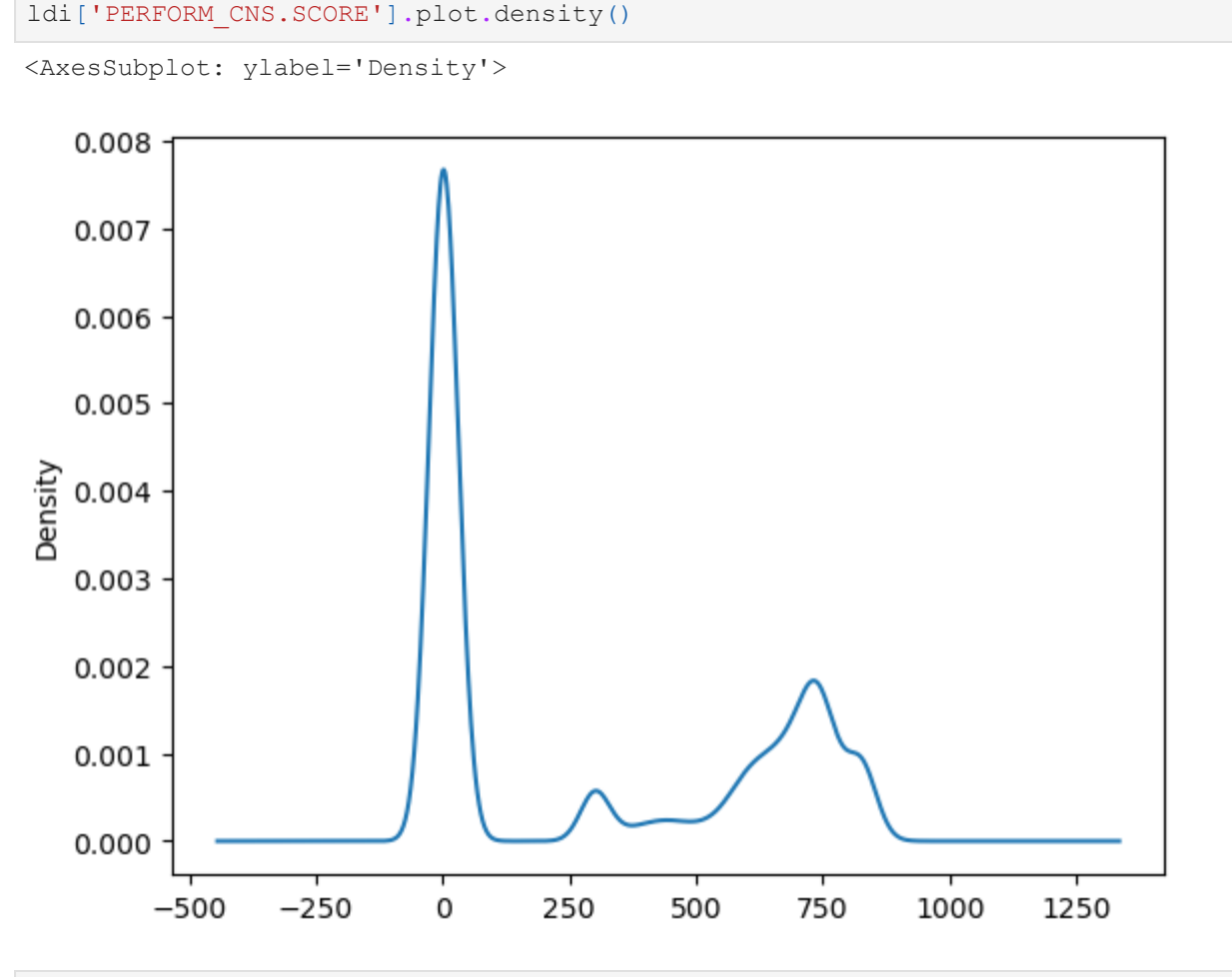


In [30]:
```
#replacing with median values

medltv=ldi['ltv'].median()
print(medda)
ldi['ltv'].replace(np.nan, medltv, inplace=True)
```

```
53803.0
```

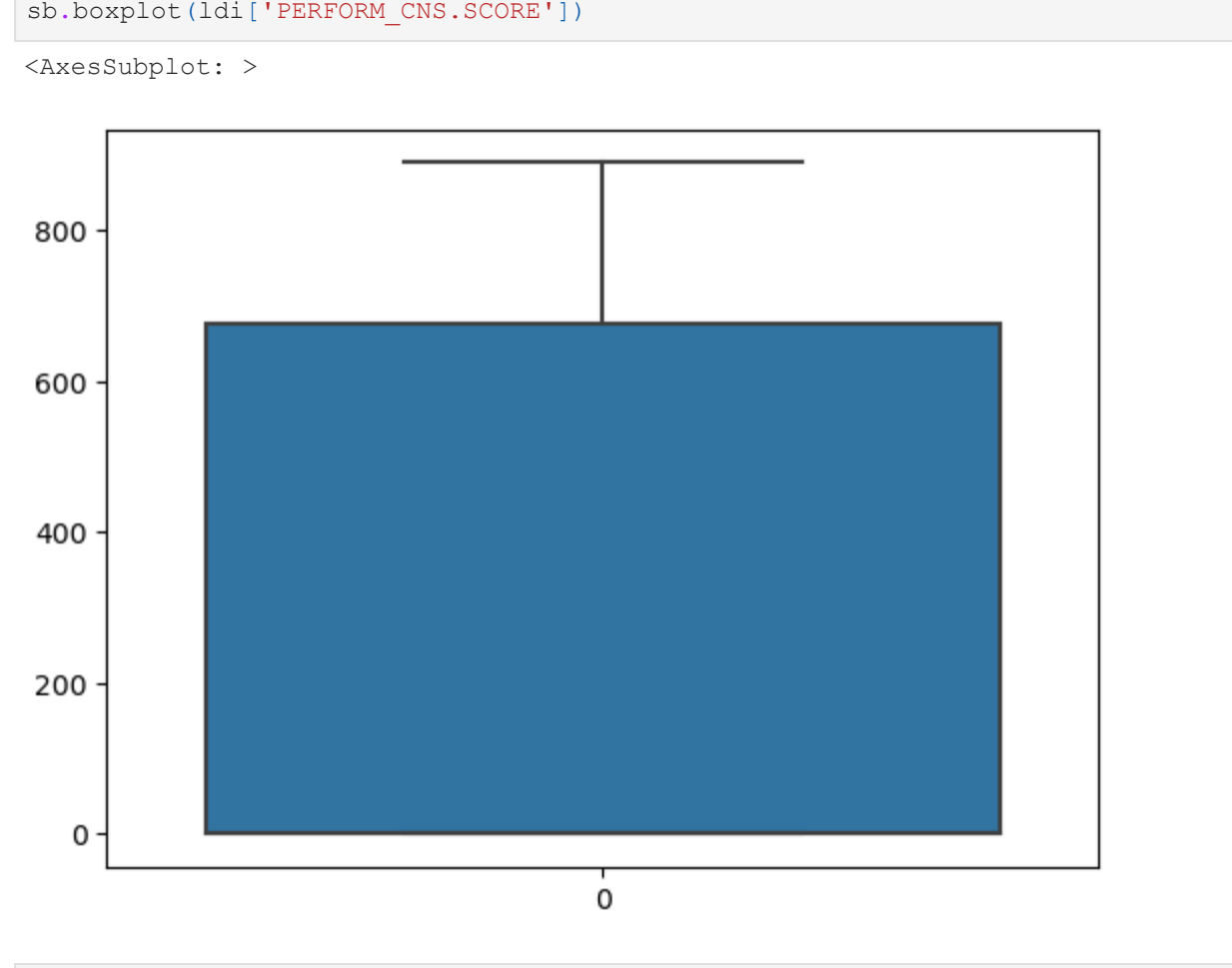
In [31]:
```
#checking distribution for PERFORM_CNS.SCORE
ldi['PERFORM_CNS.SCORE'].plot.density()
```

Out[31]: <AxesSubplot: ylabel='Density'>



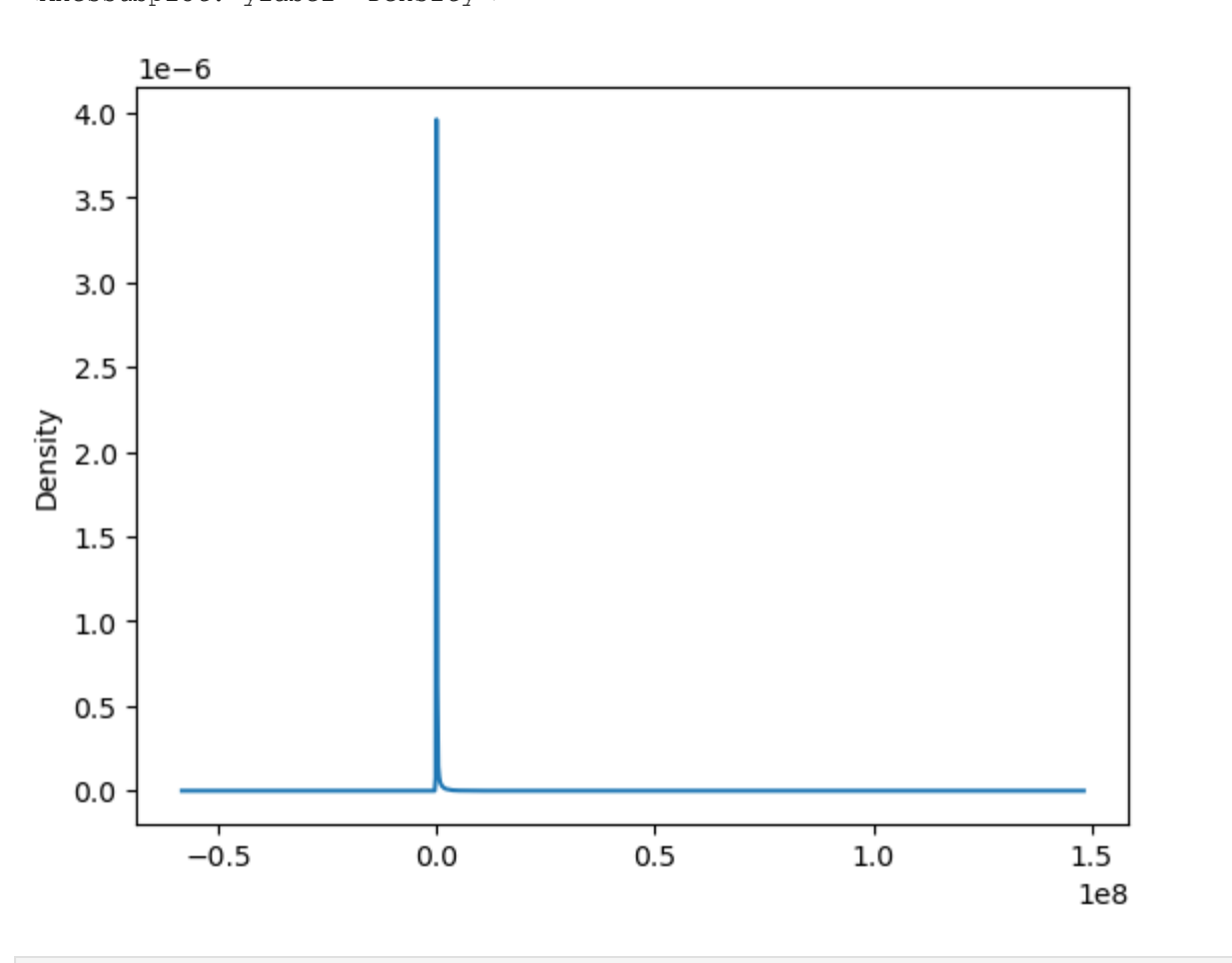
In [32]:
```
#checking outliers using boxplot PERFORM_CNS.SCORE
sb.boxplot(ldi['PERFORM_CNS.SCORE'])
```

Out[32]: <AxesSubplot: >



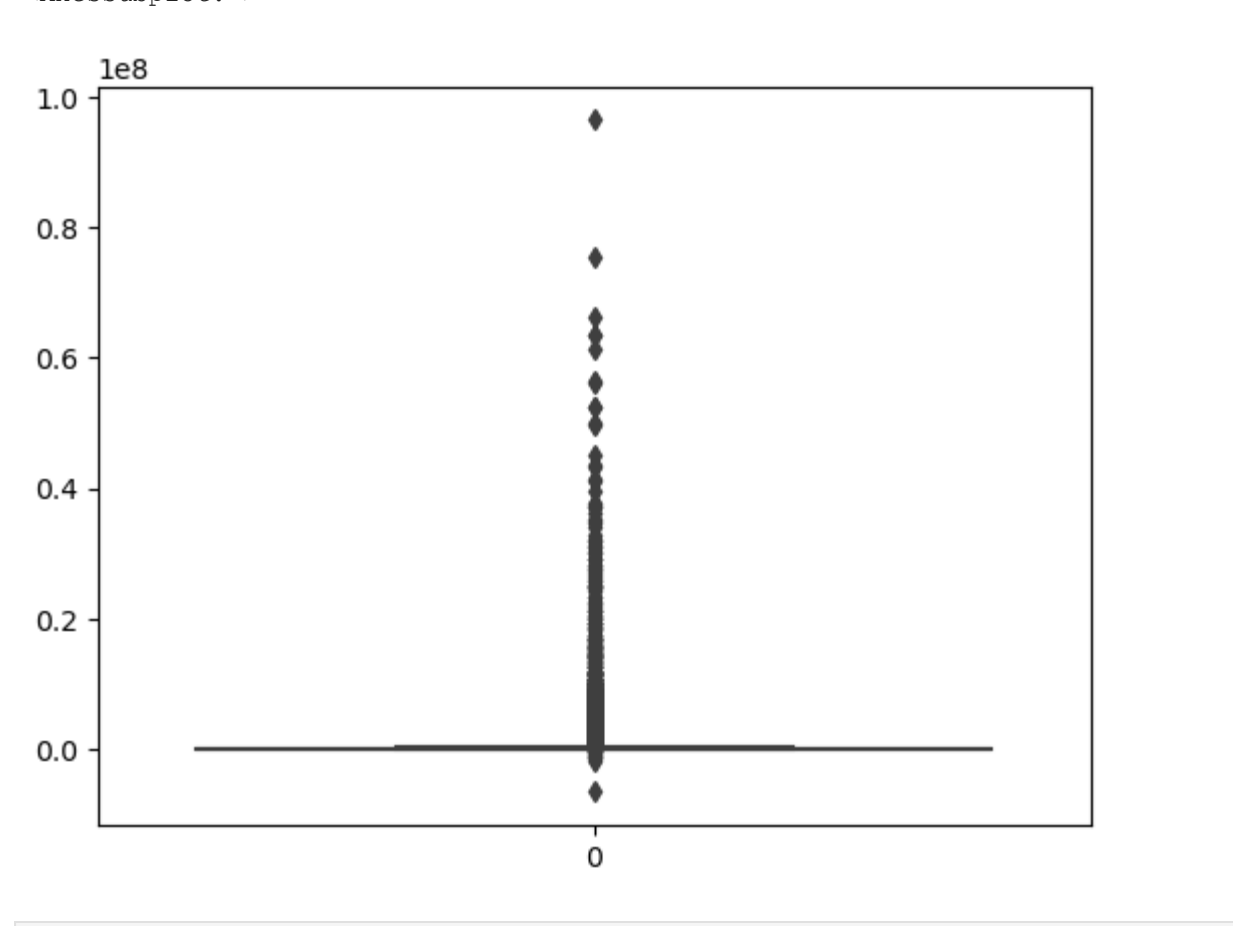
In [33]:
```
#checking distribution for PRI.CURRENT.BALANCE
ldi['PRI.CURRENT.BALANCE'].plot.density()
```

Out[33]: <AxesSubplot: ylabel='Density'>



In [34]:
```
#checking outliers using boxplot PRI.CURRENT.BALANCE
sb.boxplot(ldi['PRI.CURRENT.BALANCE'])
```

Out[34]: <AxesSubplot: >



In [35]:
```
#outlier treatment
ldi['PRI.CURRENT.BALANCE'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[35]:
```
0.10          0.00
0.25          0.00
0.50          0.00
0.70      16524.30
0.90     302356.50
0.95     804263.80
0.99    2917864.33
Name: PRI.CURRENT.BALANCE, dtype: float64
```

In [36]:
```
#seeing limits
pricb_HE=ldi[ldi['PRI.CURRENT.BALANCE'] > 804263.80].copy()
pricb_LE=ldi[ldi['PRI.CURRENT.BALANCE']< 302356.50].copy()
```

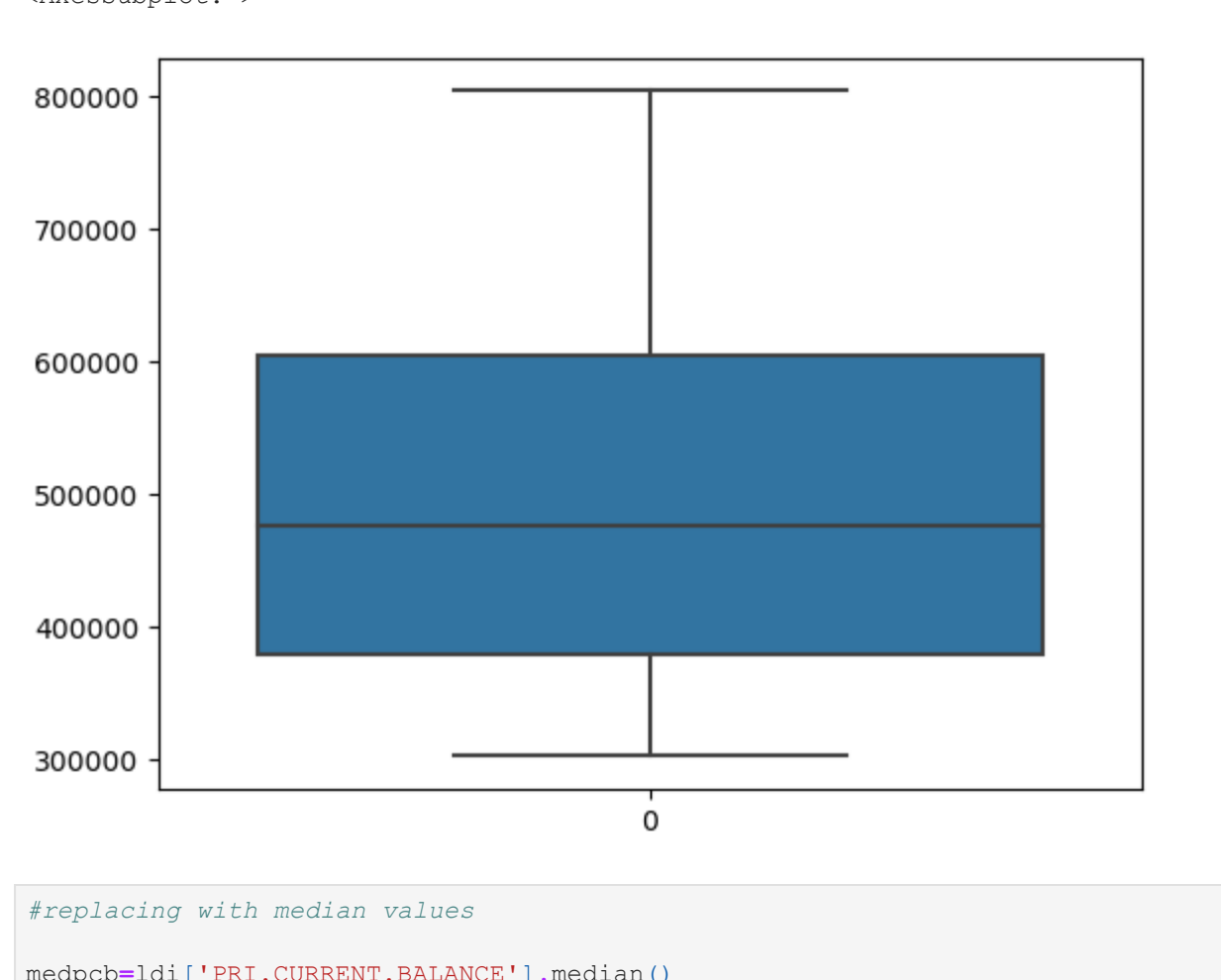
In [37]:
```
#putting them at one place
print(len(pricb_HE))
print(len(disbursed_amount_LE))
```

```
11658
23315
```

In [38]:
```
#removing outliers
for x in ldi['PRI.CURRENT.BALANCE']:
    if x > 804263.80:
        ldi['PRI.CURRENT.BALANCE'].replace(x,np.nan,inplace=True)
for x in ldi['PRI.CURRENT.BALANCE']:
    if x < 302356.50:
        ldi['PRI.CURRENT.BALANCE'].replace(x,np.nan,inplace=True)
```

In [39]:
```
#checking outliers gone or not
sb.boxplot(ldi['PRI.CURRENT.BALANCE'])
```

Out[39]: <AxesSubplot: >

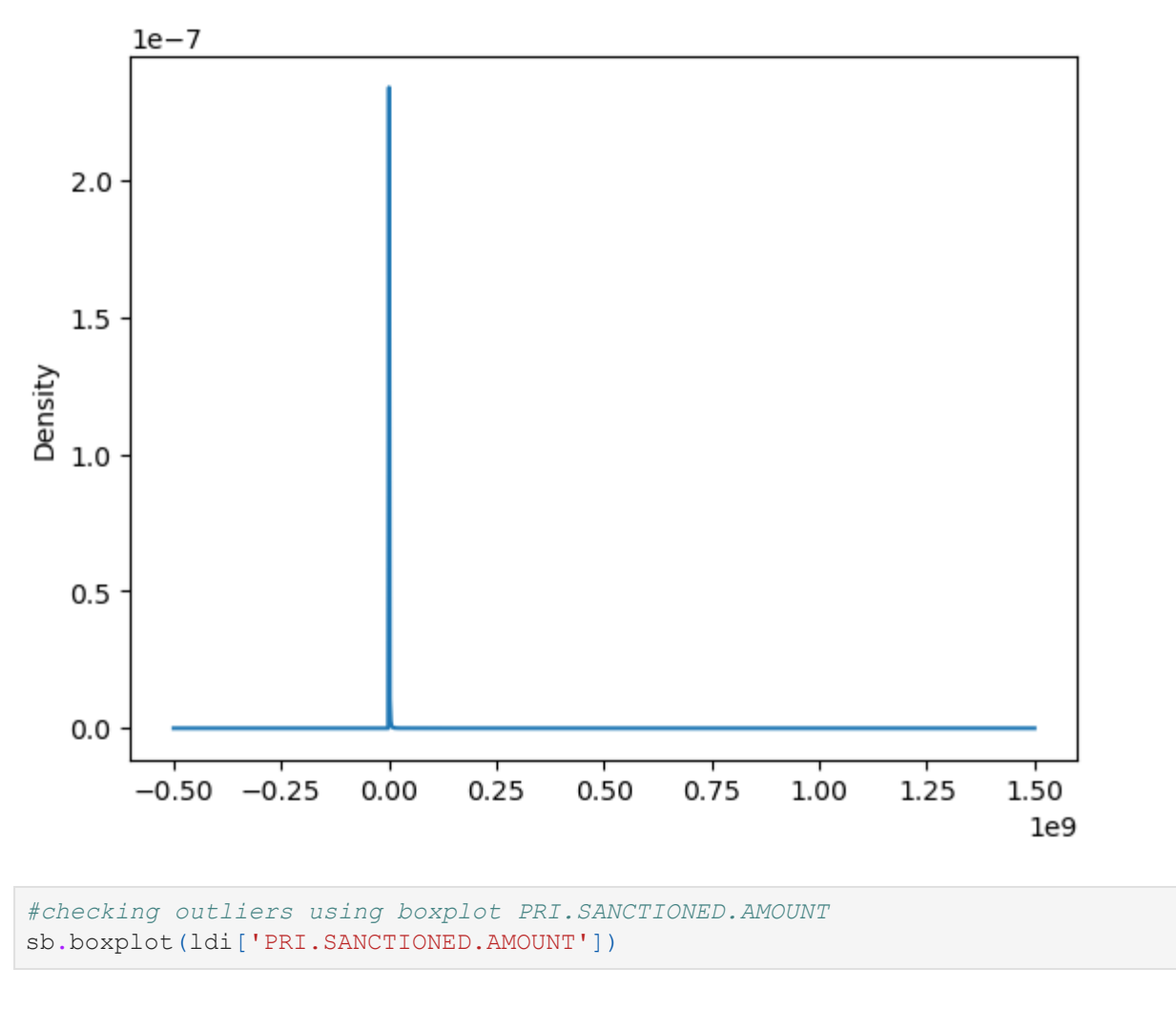


In [40]:
```
#replacing with median values

medpcb=ldi['PRI.CURRENT.BALANCE'].median()
print(medpcb)
ldi['PRI.CURRENT.BALANCE'].replace(np.nan, medpcb, inplace=True)
```

```
476199.0
```

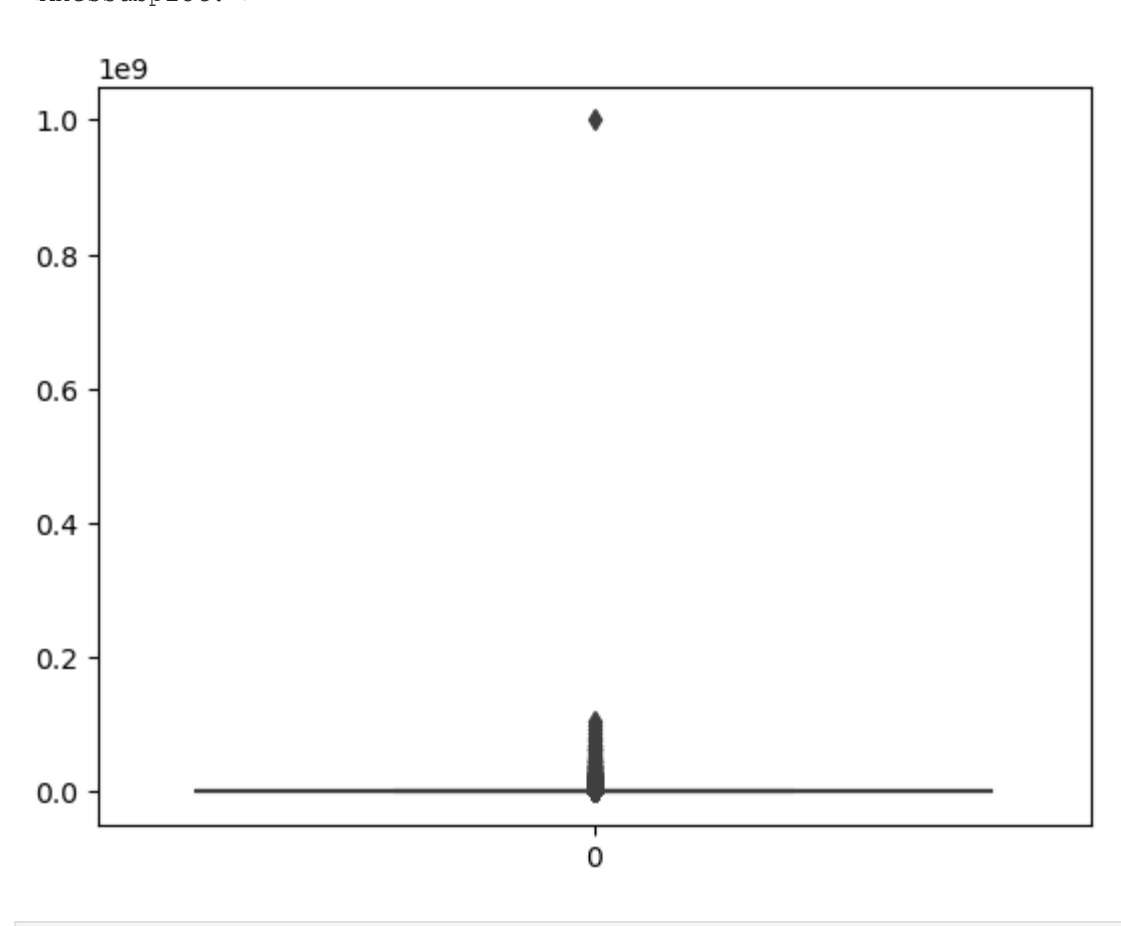
In [41]:
```
#checking distribution for PRI.SANCTIONED.AMOUNT
ldi['PRI.SANCTIONED.AMOUNT'].plot.density()
```

Out[41]: <AxesSubplot: ylabel='Density'>



In [42]:
```
#checking outliers using boxplot PRI.SANCTIONED.AMOUNT
sb.boxplot(ldi['PRI.SANCTIONED.AMOUNT'])
```

Out[42]: <AxesSubplot: >



In [43]:
```
#outlier treatment
ldi['PRI.SANCTIONED.AMOUNT'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[43]:
```
0.10          0.00
0.25          0.00
0.50          0.00
0.70      41486.20
0.90     447157.30
0.95    1030280.05
0.99    3467119.37
Name: PRI.SANCTIONED.AMOUNT, dtype: float64
```

In [44]:
```
#seeing limits
psa_HE=ldi[ldi['PRI.SANCTIONED.AMOUNT'] > 3467119.37].copy()
psa_LE=ldi[ldi['PRI.SANCTIONED.AMOUNT']< 447157.30].copy()
```

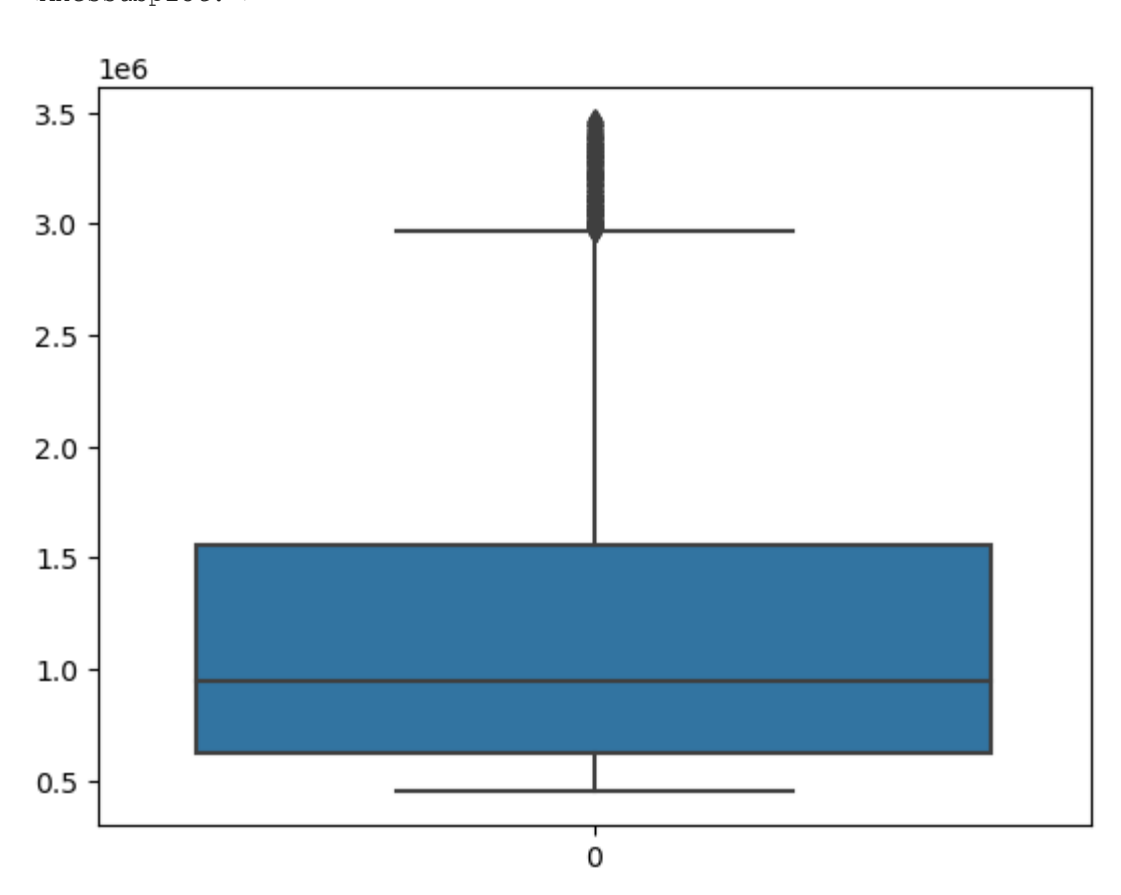
In [45]:
```
#putting them at one place
print(len(psa_HE))
print(len(psa_LE))
```

```
2332
209838
```

In [46]:
```
#removing outliers
for x in ldi['PRI.SANCTIONED.AMOUNT']:
    if x > 3467119.37:
        ldi['PRI.SANCTIONED.AMOUNT'].replace(x,np.nan,inplace=True)
for x in ldi['PRI.SANCTIONED.AMOUNT']:
    if x < 447157.30:
        ldi['PRI.SANCTIONED.AMOUNT'].replace(x,np.nan,inplace=True)
```

In [47]:
```
#checking outliers gone or not
sb.boxplot(ldi['PRI.SANCTIONED.AMOUNT'])
```

Out[47]: <AxesSubplot: >

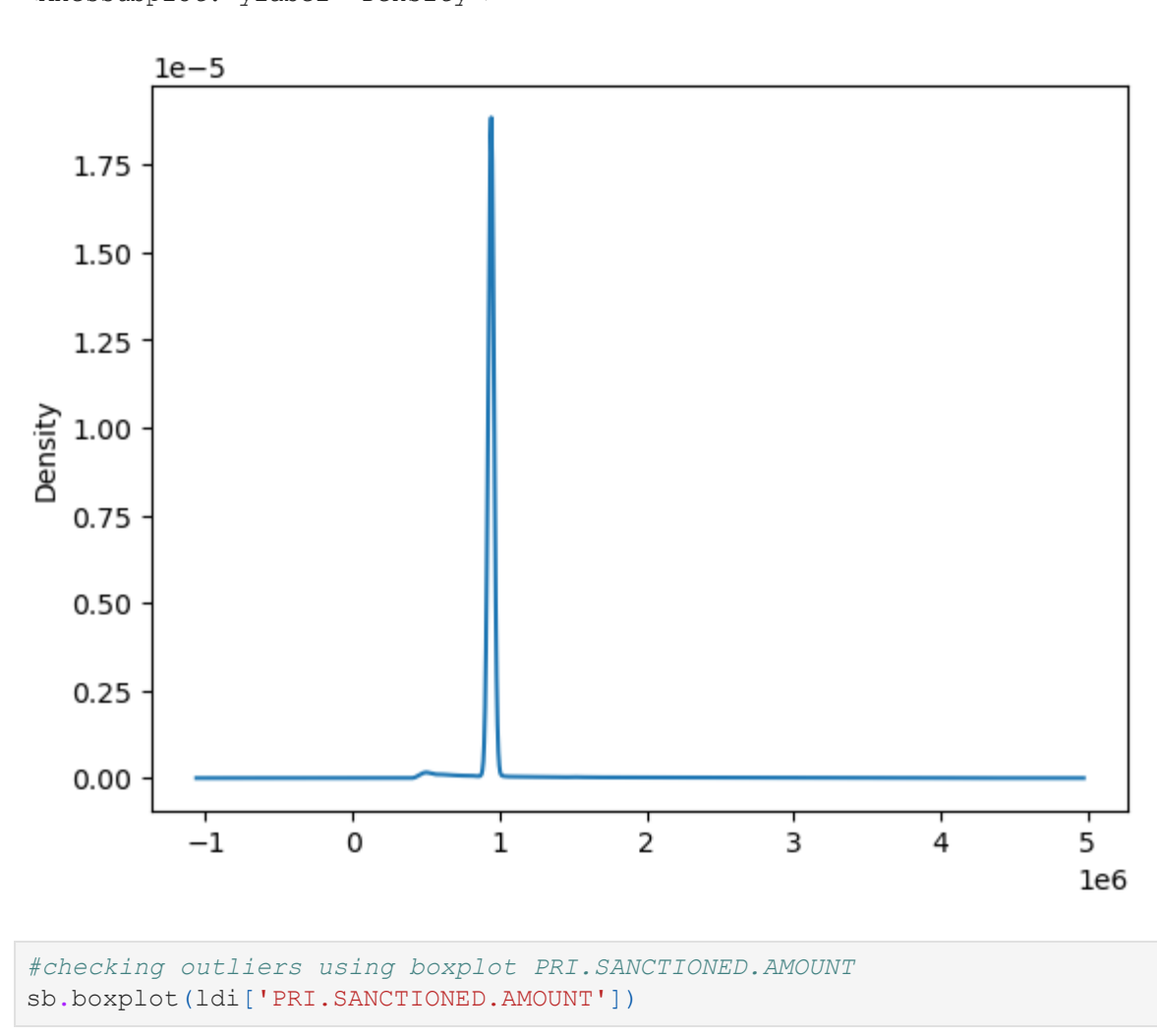


In [48]:
```
#replacing with median values

medpsa=ldi['PRI.SANCTIONED.AMOUNT'].median()
print(medpsa)
ldi['PRI.SANCTIONED.AMOUNT'].replace(np.nan, medpsa, inplace=True)
```

943000.0

In [49]:
```
#checking distribution for PRI.SANCTIONED.AMOUNT
ldi['PRI.SANCTIONED.AMOUNT'].plot.density()
```

Out[49]: <AxesSubplot: ylabel='Density'>



In [50]:
```
#checking outliers using boxplot PRI.SANCTIONED.AMOUNT
sb.boxplot(ldi['PRI.SANCTIONED.AMOUNT'])
```

Out[50]: <AxesSubplot: >

In [51]:
```
#outlier treatment
ldi['PRI.SANCTIONED.AMOUNT'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[51]:
```
0.10     943000.0
0.25     943000.0
0.50     943000.0
0.70     943000.0
0.90     943000.0
0.95     943000.0
0.99    2250000.0
Name: PRI.SANCTIONED.AMOUNT, dtype: float64
```

In [52]:
```
#seeing limits
psa_HE=ldi[ldi['PRI.SANCTIONED.AMOUNT'] > 2250000.0].copy()
psa_LE=ldi[ldi['PRI.SANCTIONED.AMOUNT']<  943000.0].copy()
```

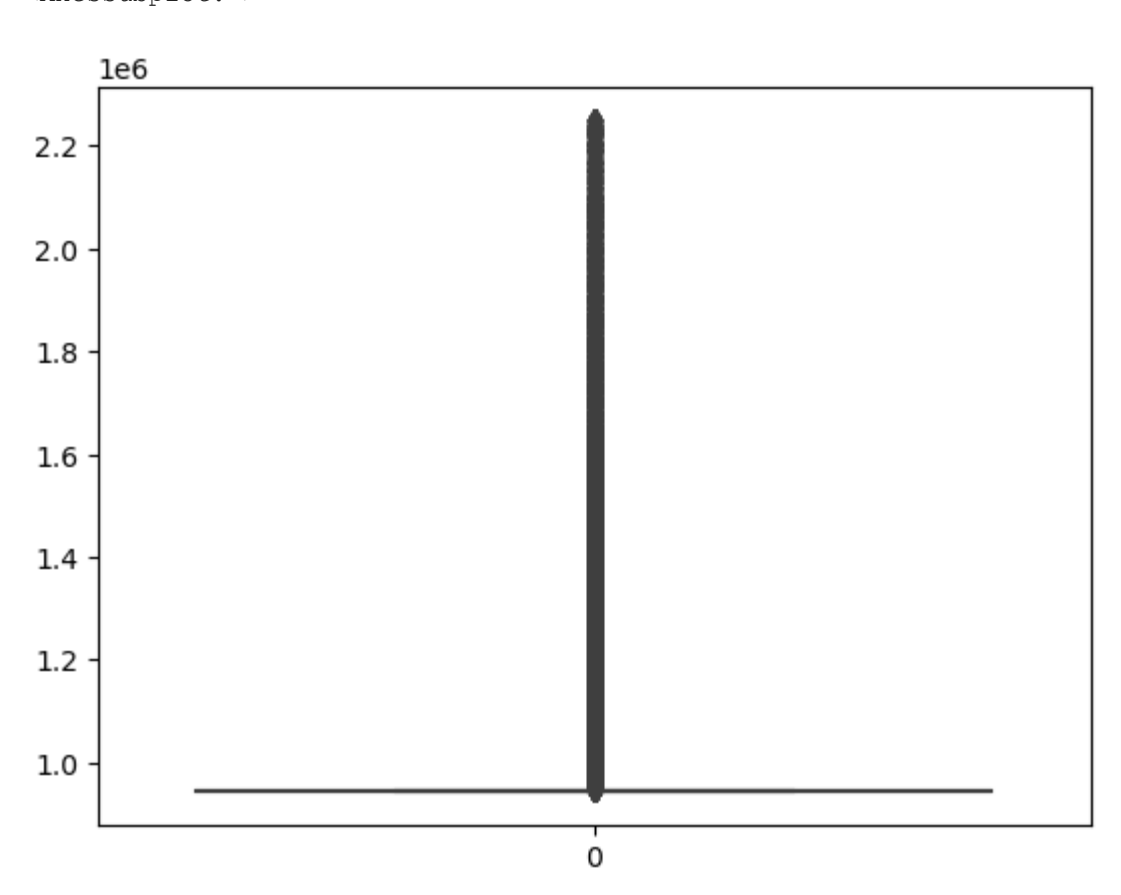
In [53]:
```
#putting them at one place
print(len(psa_HE))
print(len(psa_LE))
```

```
2323
10488
```

In [54]:
```
#removing outliers
for x in ldi['PRI.SANCTIONED.AMOUNT']:
    if x > 2250000.0:
        ldi['PRI.SANCTIONED.AMOUNT'].replace(x,np.nan,inplace=True)
for x in ldi['PRI.SANCTIONED.AMOUNT']:
    if x < 943000.0:
        ldi['PRI.SANCTIONED.AMOUNT'].replace(x,np.nan,inplace=True)
```

In [55]:
```
#checking outliers gone or not
sb.boxplot(ldi['PRI.SANCTIONED.AMOUNT'])
```

Out[55]: <AxesSubplot: >

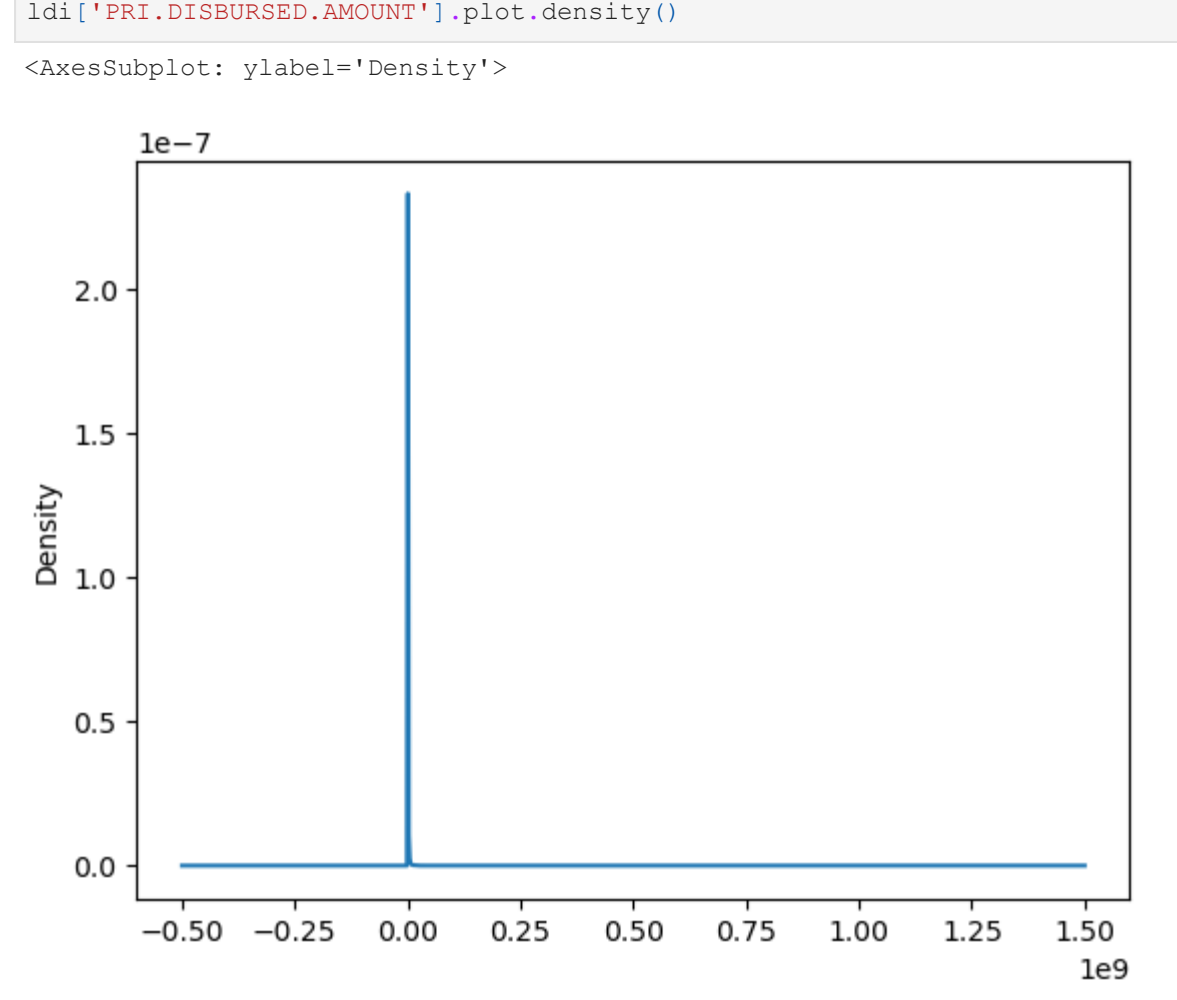


In [56]:
```
#replacing with median values

medpsa=ldi['PRI.SANCTIONED.AMOUNT'].median()
print(medpsa)
ldi['PRI.SANCTIONED.AMOUNT'].replace(np.nan, medpsa, inplace=True)
```

943000.0

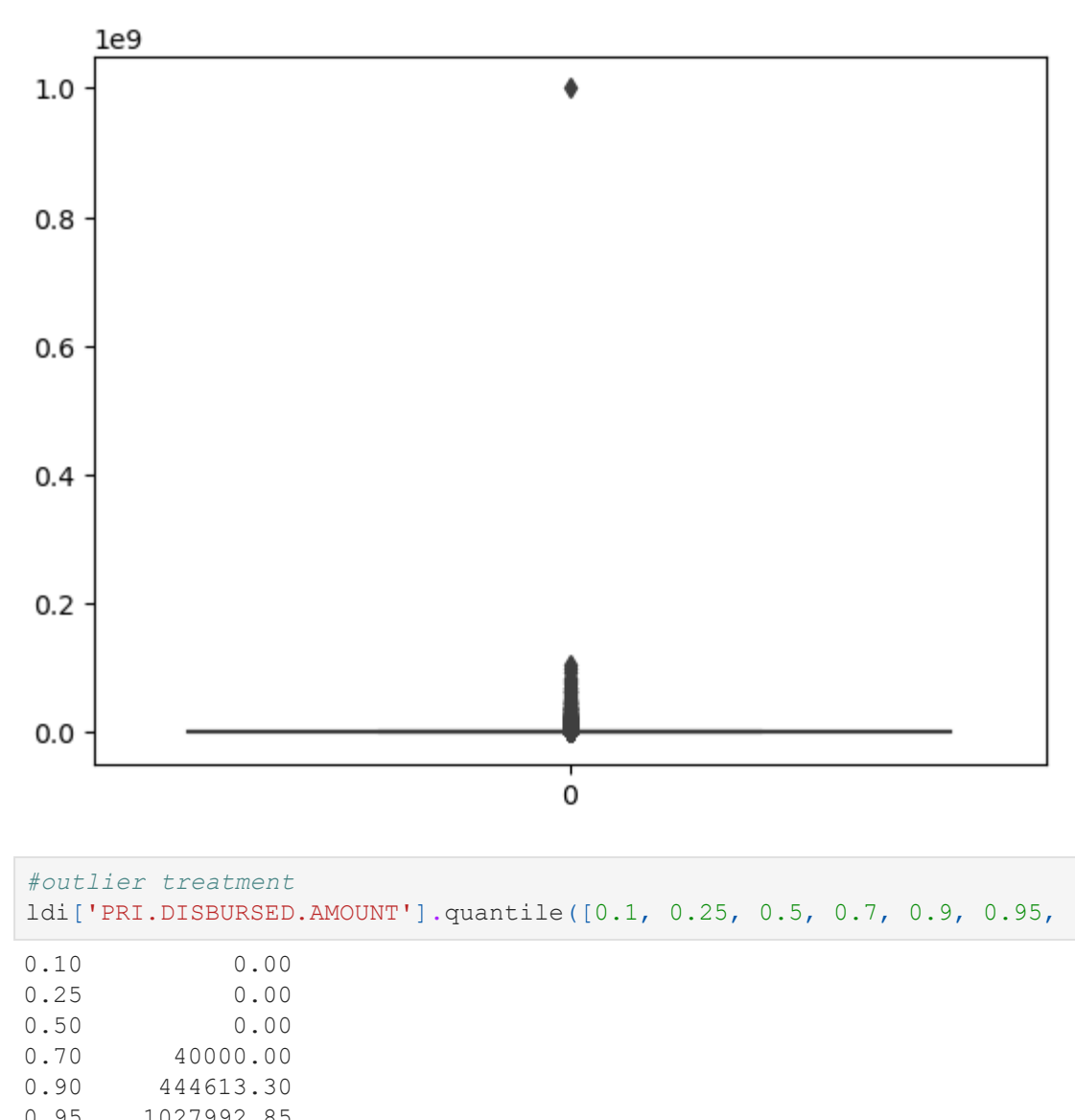
In [57]:
```
#checking distribution for PRI.DISBURSED.AMOUNT
ldi['PRI.DISBURSED.AMOUNT'].plot.density()
```

Out[57]: <AxesSubplot: ylabel='Density'>



In [58]:
```
#checking outliers using boxplot PRI.DISBURSED.AMOUNT
sb.boxplot(ldi['PRI.DISBURSED.AMOUNT'])
```

Out[58]: <AxesSubplot: >



In [59]:
```
#outlier treatment
ldi['PRI.DISBURSED.AMOUNT'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[59]:
```
0.10          0.00
0.25          0.00
0.50          0.00
0.70      40000.00
0.90     444613.30
0.95    1027992.85
0.99    3457573.88
Name: PRI.DISBURSED.AMOUNT, dtype: float64
```

In [60]:
```
#seeing limits
pda_HE=ldi[ldi['PRI.DISBURSED.AMOUNT'] > 3457573.88].copy()
pda_LE=ldi[ldi['PRI.DISBURSED.AMOUNT']< 444613.30].copy()
```

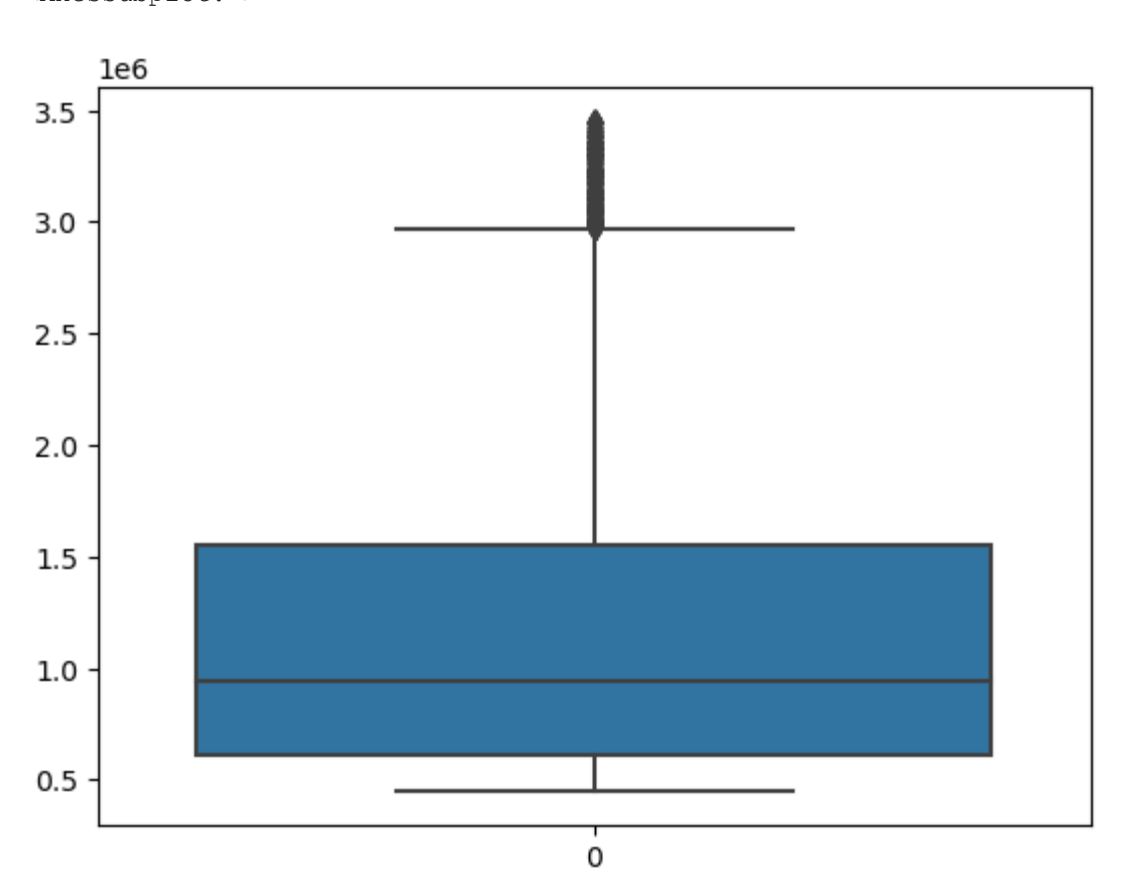
In [61]:
```
#putting them at one place
print(len(pda_HE))
print(len(pda_LE))
```

```
2332
209838
```

In [62]:
```
#removing outliers
for x in ldi['PRI.DISBURSED.AMOUNT']:
    if x >3457573.88:
        ldi['PRI.DISBURSED.AMOUNT'].replace(x,np.nan,inplace=True)
for x in ldi['PRI.DISBURSED.AMOUNT']:
    if x < 444613.30:
        ldi['PRI.DISBURSED.AMOUNT'].replace(x,np.nan,inplace=True)
```

In [63]:
```
#checking outliers gone or not
sb.boxplot(ldi['PRI.DISBURSED.AMOUNT'])
```

Out[63]: <AxesSubplot: >

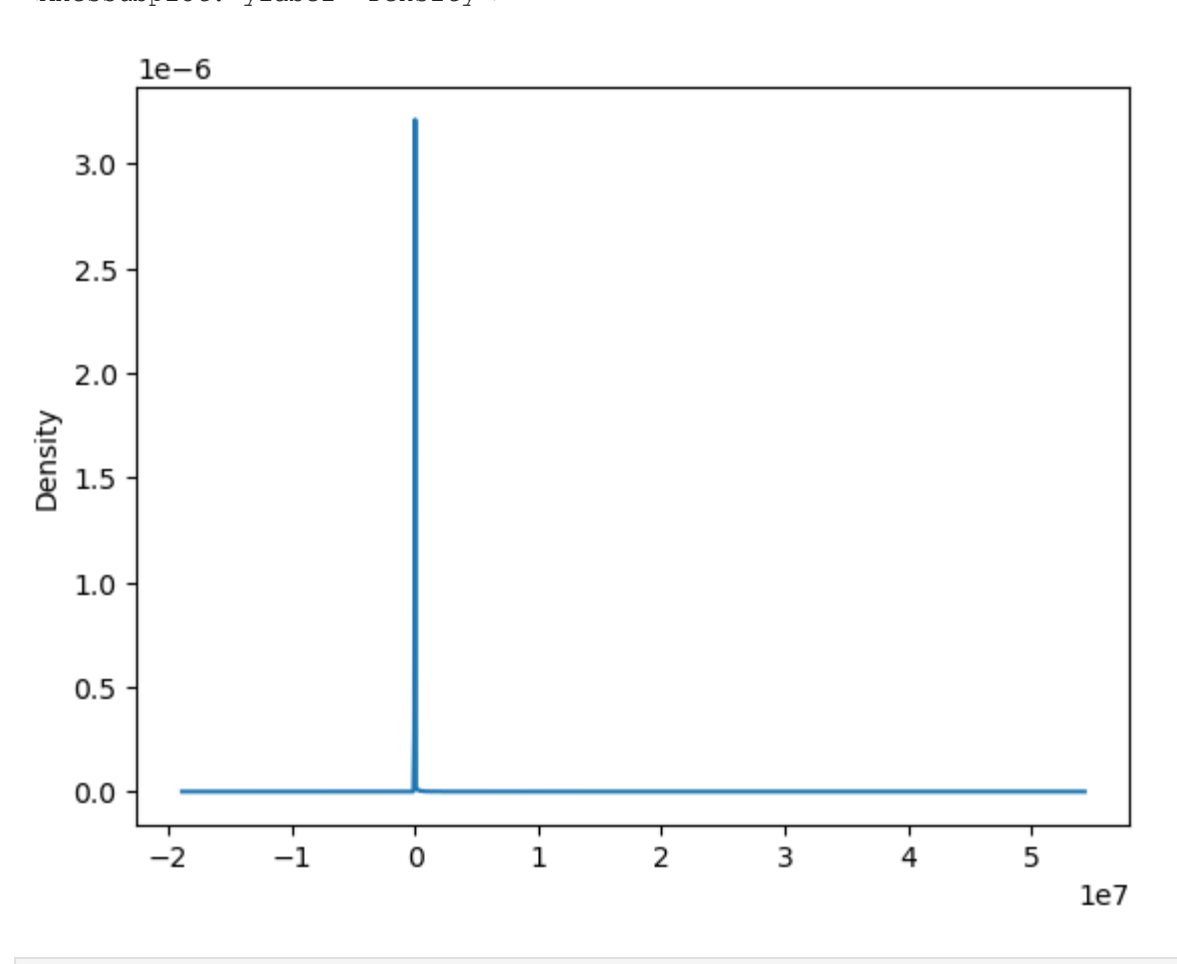


In [64]:
```
#replacing with median values

medpda=ldi['PRI.DISBURSED.AMOUNT'].median()
print(medpda)
ldi['PRI.DISBURSED.AMOUNT'].replace(np.nan, medpda, inplace=True)
```

942024.5

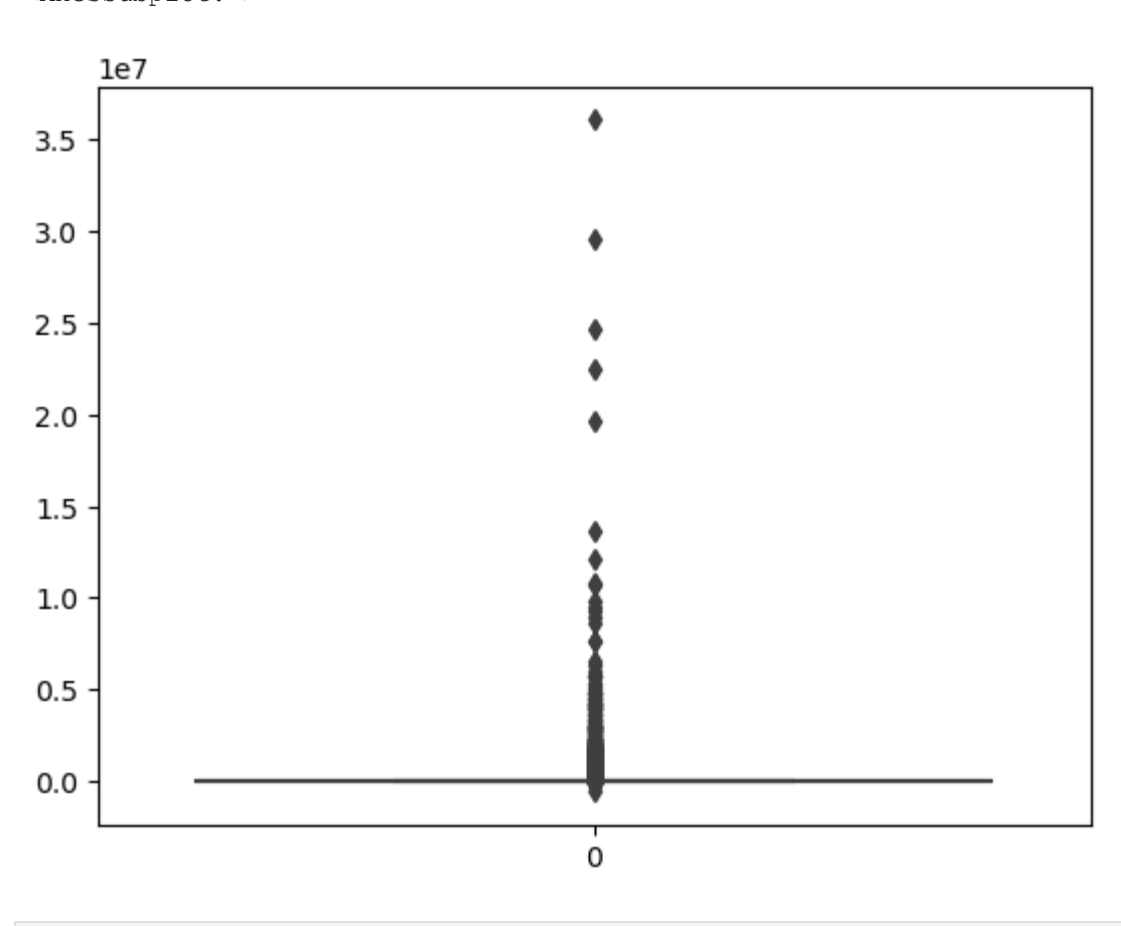
In [65]:
```
#checking distribution for SEC.CURRENT.BALANCE
ldi['SEC.CURRENT.BALANCE'].plot.density()
```

Out[65]: <AxesSubplot: ylabel='Density'>



In [66]:
```
#checking outliers using boxplot SEC.CURRENT.BALANCE
sb.boxplot(ldi['SEC.CURRENT.BALANCE'])
```

Out[66]: <AxesSubplot: >



In [67]:
```
#outlier treatment
ldi['SEC.CURRENT.BALANCE'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[67]:
```
0.10        0.00
0.25        0.00
0.50        0.00
0.70        0.00
0.90        0.00
0.95        0.00
0.99    19733.78
Name: SEC.CURRENT.BALANCE, dtype: float64
```

In [68]:
```
#seeing limits
scb_HE=ldi[ldi['SEC.CURRENT.BALANCE'] > 20000].copy()
scb_LE=ldi[ldi['SEC.CURRENT.BALANCE']< 100].copy()
```

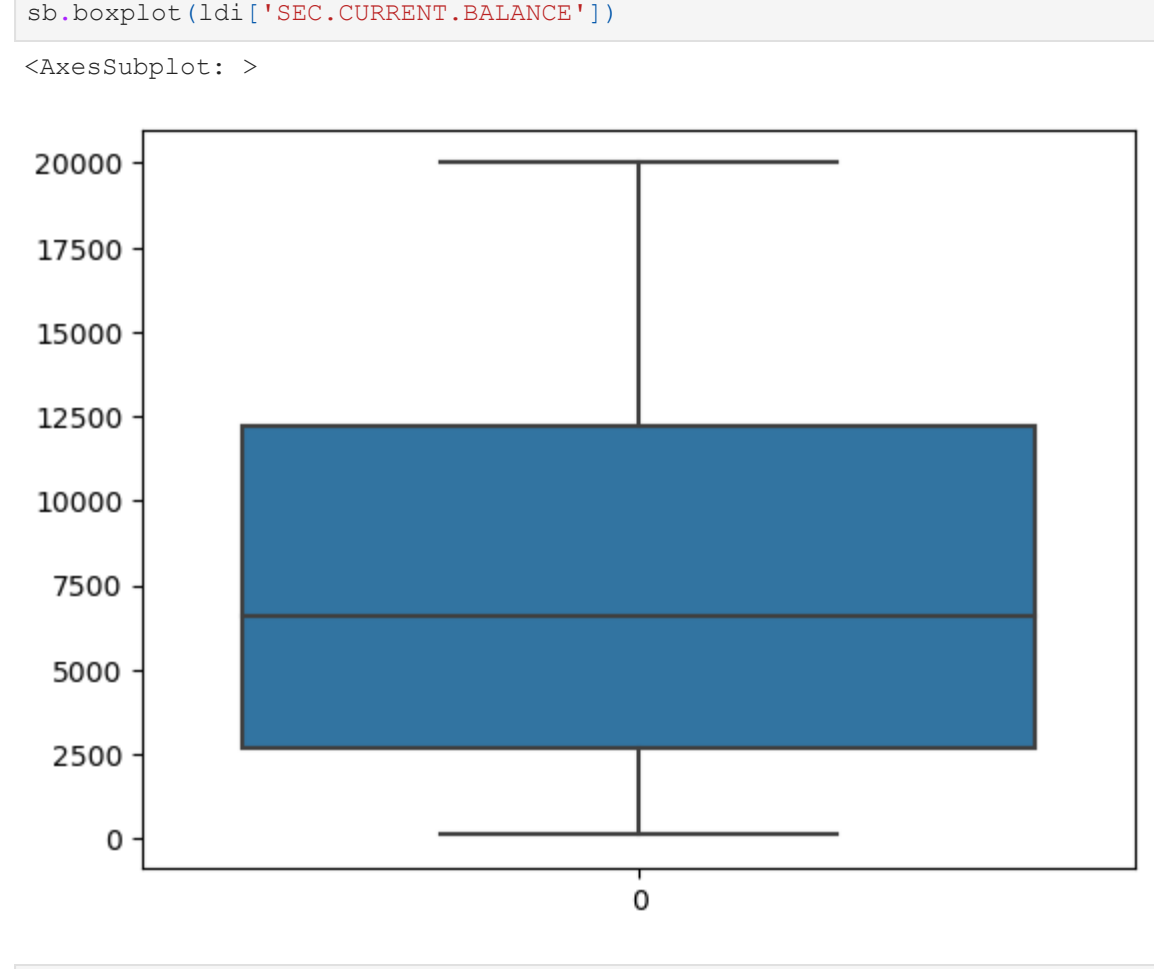
In [69]:
```
#putting them at one place
print(len(scb_HE))
print(len(scb_LE))
```

```
2316
229874
```

In [70]:
```
#removing outliers
for x in ldi['SEC.CURRENT.BALANCE']:
    if x > 20000:
        ldi['SEC.CURRENT.BALANCE'].replace(x,np.nan,inplace=True)
for x in ldi['SEC.CURRENT.BALANCE']:
    if x < 100:
        ldi['SEC.CURRENT.BALANCE'].replace(x,np.nan,inplace=True)
```

In [71]:
```
#checking outliers gone or not
sb.boxplot(ldi['SEC.CURRENT.BALANCE'])
```

Out[71]: <AxesSubplot: >

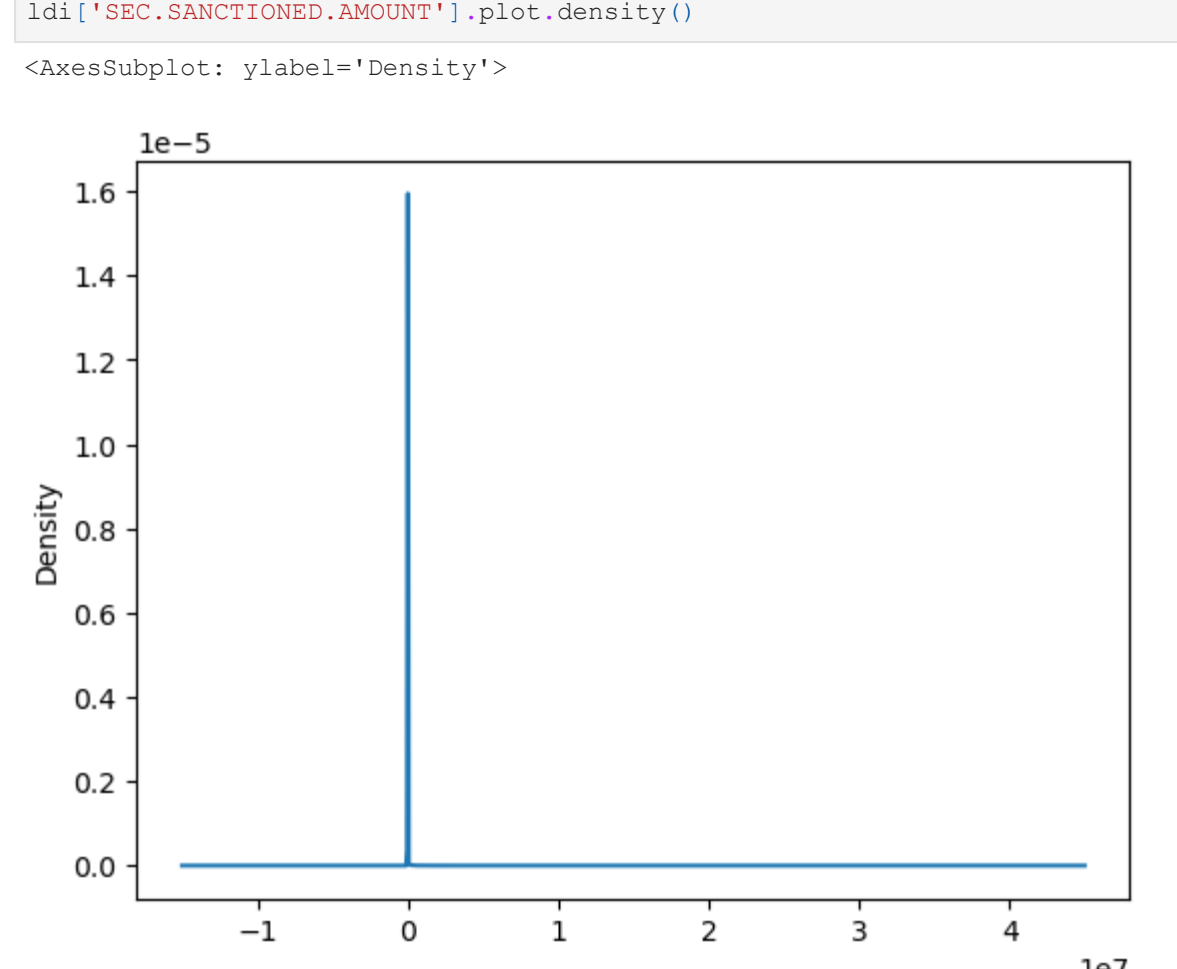


In [72]:
```
#replacing with median values

medscb=ldi['SEC.CURRENT.BALANCE'].median()
print(medscb)
ldi['SEC.CURRENT.BALANCE'].replace(np.nan, medscb, inplace=True)
```

6599.0

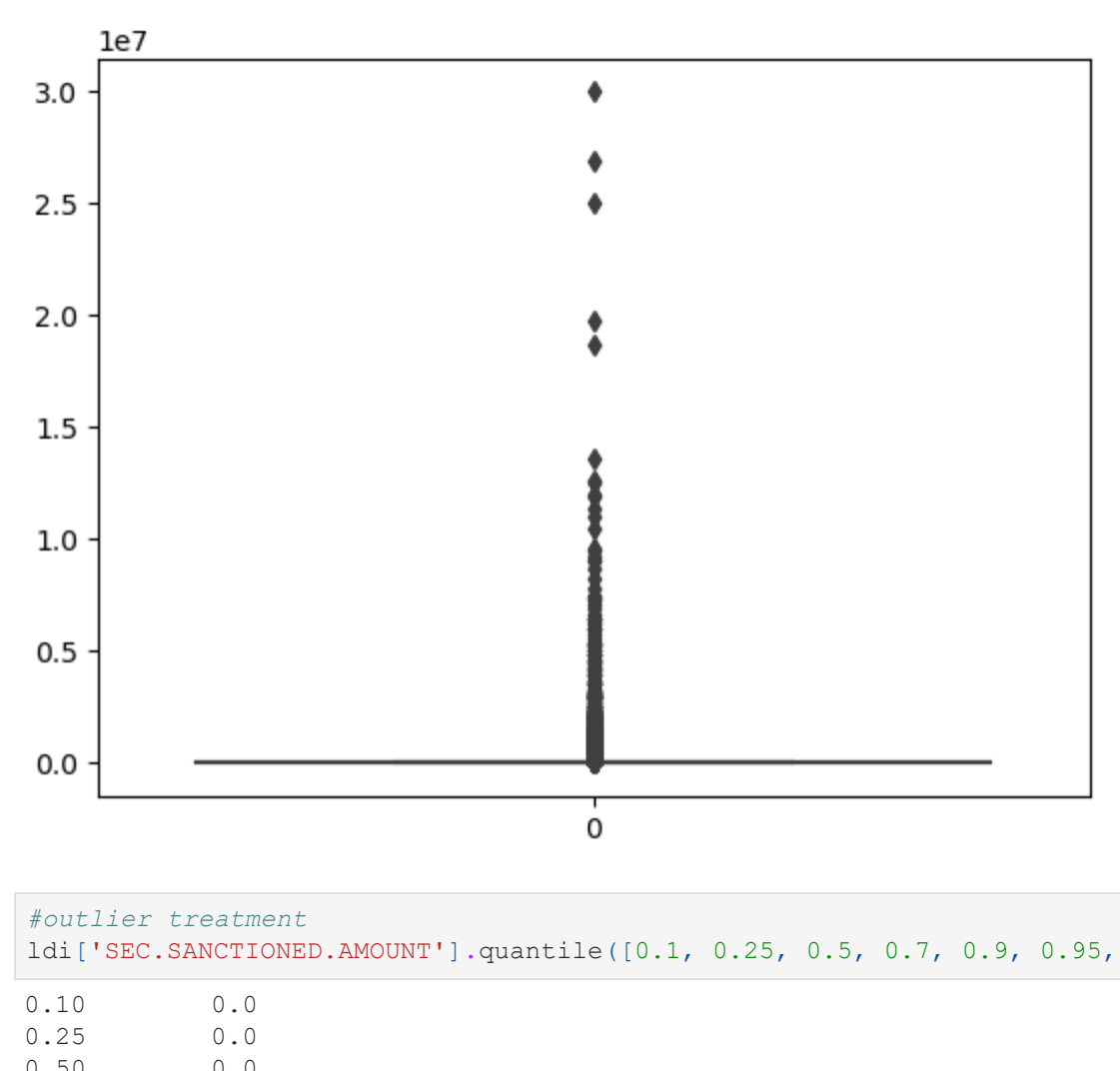
In [73]:
```
#checking distribution for SEC.SANCTIONED.AMOUNT
ldi['SEC.SANCTIONED.AMOUNT'].plot.density()
```

Out[73]: <AxesSubplot: ylabel='Density'>



In [74]:
```
#checking outliers using boxplot SEC.SANCTIONED.AMOUNT
sb.boxplot(ldi['SEC.SANCTIONED.AMOUNT'])
```

Out[74]: <AxesSubplot: >



In [75]:
```
#outlier treatment
ldi['SEC.SANCTIONED.AMOUNT'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[75]:
```
0.10        0.0
0.25        0.0
0.50        0.0
0.70        0.0
0.90        0.0
0.95        0.0
0.99    50000.0
Name: SEC.SANCTIONED.AMOUNT, dtype: float64
```

In [76]:
```
#seeing limits
ssa_HE=ldi[ldi['SEC.SANCTIONED.AMOUNT'] > 100000].copy()
ssa_LE=ldi[ldi['SEC.SANCTIONED.AMOUNT']< 1000].copy()
```

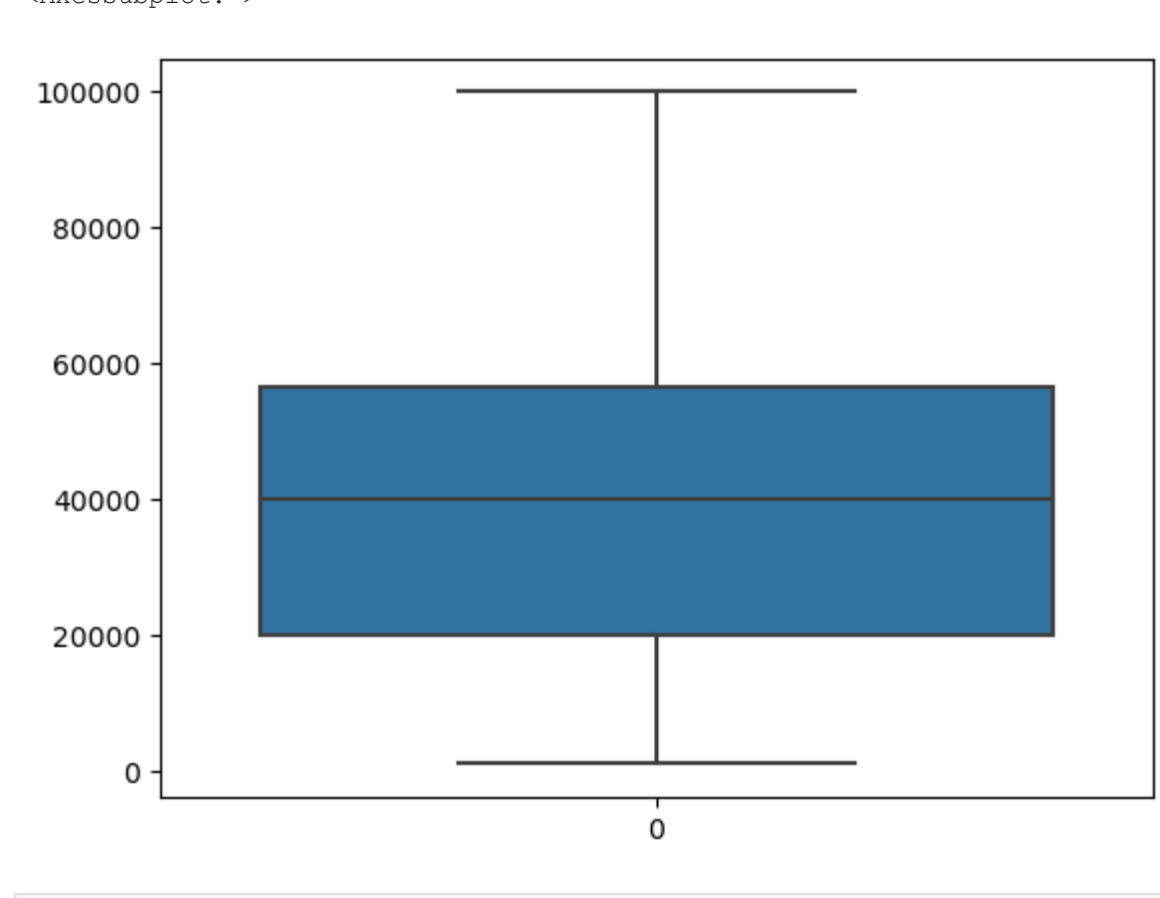
In [77]:
```
#putting them at one place
print(len(ssa_HE))
print(len(ssa_LE))
```

```
1614
229463
```

In [78]:
```
#removing outliers
for x in ldi['SEC.SANCTIONED.AMOUNT']:
    if x > 100000:
        ldi['SEC.SANCTIONED.AMOUNT'].replace(x,np.nan,inplace=True)
for x in  ldi['SEC.SANCTIONED.AMOUNT']:
    if x < 1000:
        ldi['SEC.SANCTIONED.AMOUNT'].replace(x,np.nan,inplace=True)
```

In [79]:
```
#checking outliers gone or not
sb.boxplot(ldi['SEC.SANCTIONED.AMOUNT'])
```

Out[79]: <AxesSubplot: >

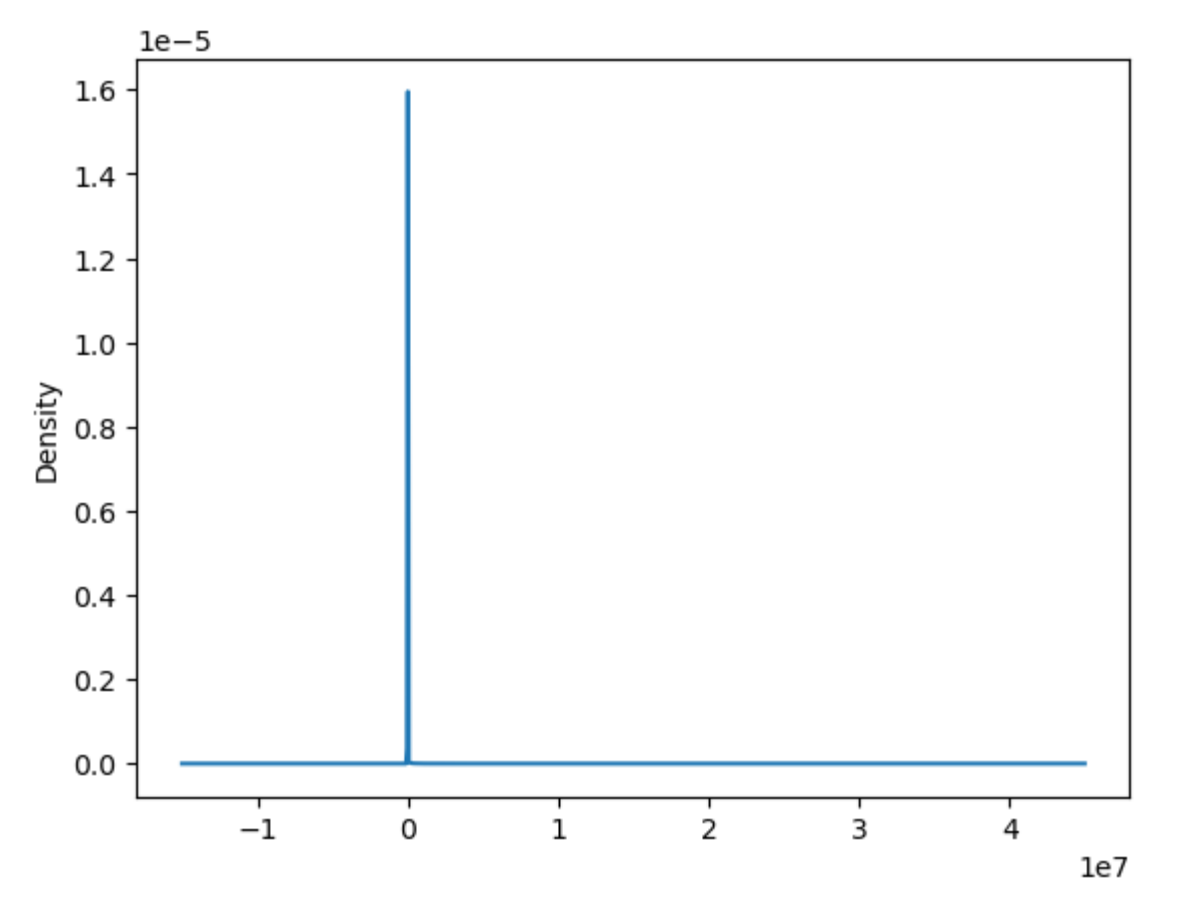


In [80]:
```
#replacing with median values

medssa=ldi['SEC.SANCTIONED.AMOUNT'].median()
print(medssa)
ldi['SEC.SANCTIONED.AMOUNT'].replace(np.nan, medssa, inplace=True)
```

40000.0

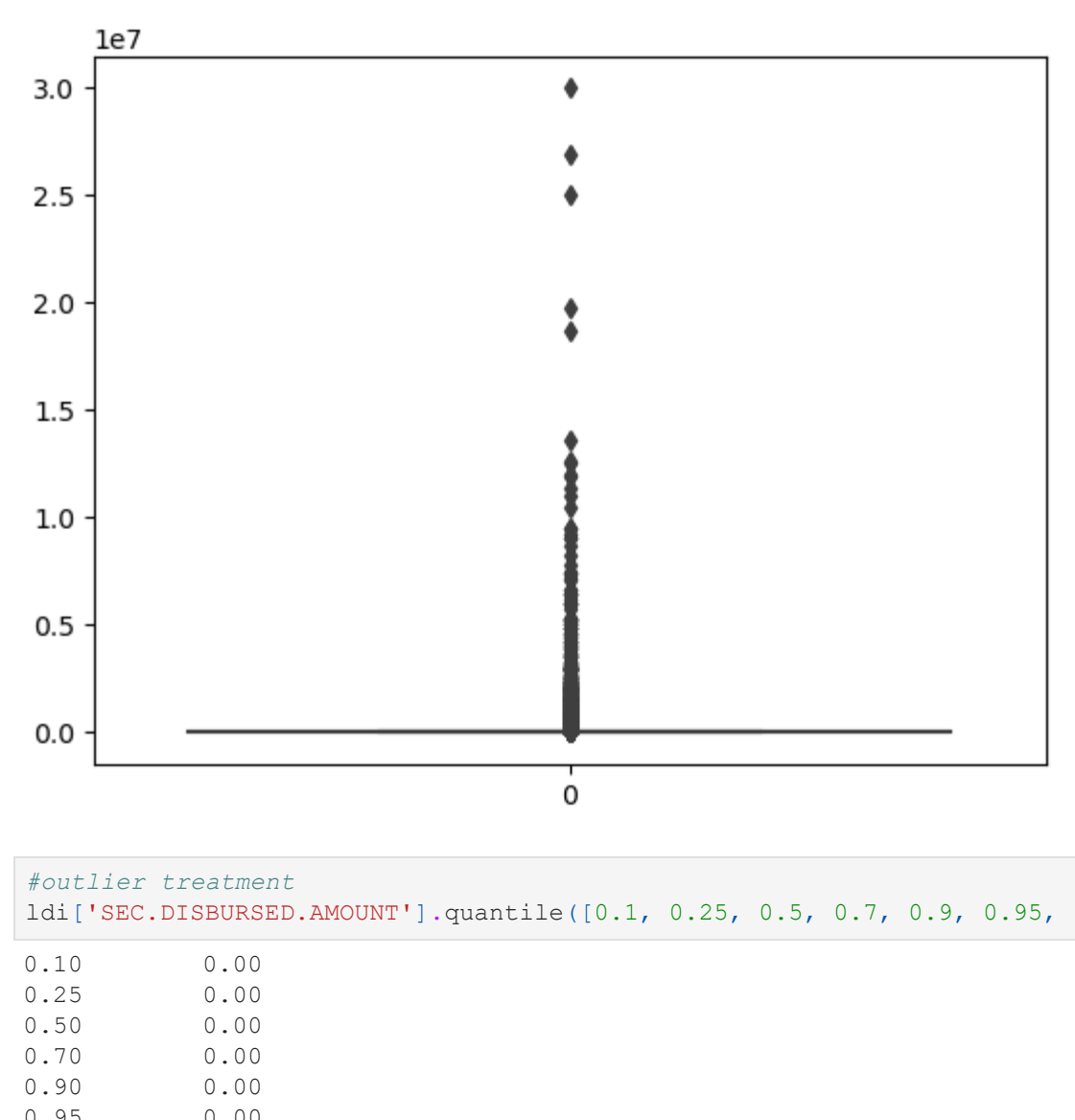
In [81]:
```
#checking distribution for SEC.DISBURSED.AMOUNT
ldi['SEC.DISBURSED.AMOUNT'].plot.density()
```

Out[81]: <AxesSubplot: ylabel='Density'>



In [82]:
```
#checking outliers using boxplot SEC.DISBURSED.AMOUNT
sb.boxplot(ldi['SEC.DISBURSED.AMOUNT'])
```

Out[82]: <AxesSubplot: >



In [83]:
```
#outlier treatment
ldi['SEC.DISBURSED.AMOUNT'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[83]:
```
0.10        0.00
0.25        0.00
0.50        0.00
0.70        0.00
0.90        0.00
0.95        0.00
0.99    47544.83
Name: SEC.DISBURSED.AMOUNT, dtype: float64
```

In [84]:
```
#seeing limits
sda_HE=ldi[ldi['SEC.DISBURSED.AMOUNT'] > 100000].copy()
sda_LE=ldi[ldi['SEC.DISBURSED.AMOUNT']< 1000].copy()
```

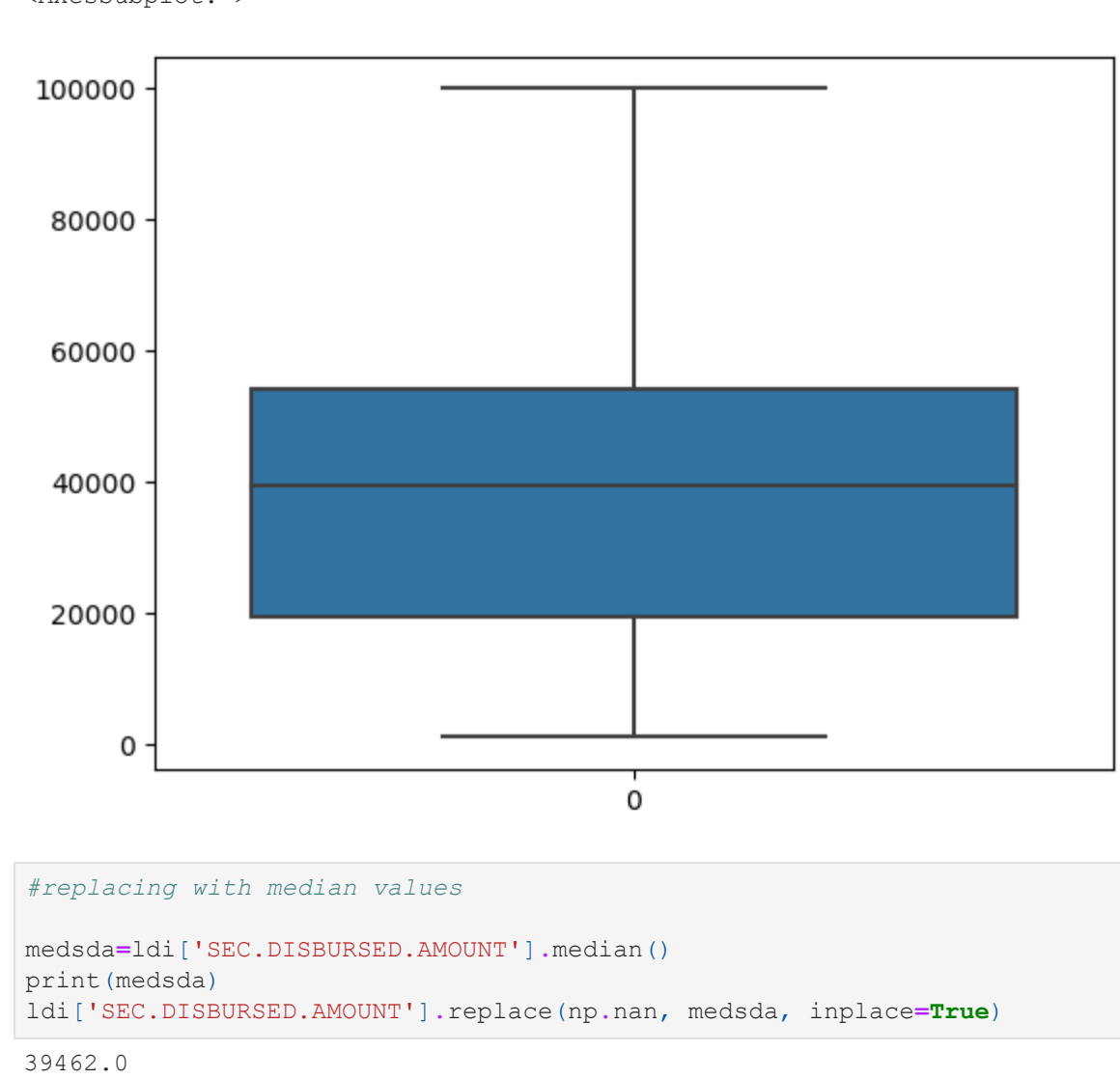
In [85]:
```
#putting them at one place
print(len(sda_HE))
print(len(sda_LE))
```

```
1559
229516
```

In [86]:
```
#removing outliers
for x in ldi['SEC.DISBURSED.AMOUNT']:
    if x > 100000:
        ldi['SEC.DISBURSED.AMOUNT'].replace(x,np.nan,inplace=True)
for x in  ldi['SEC.DISBURSED.AMOUNT']:
    if x < 1000:
        ldi['SEC.DISBURSED.AMOUNT'].replace(x,np.nan,inplace=True)
```

In [87]:
```
#checking outliers gone or not
sb.boxplot(ldi['SEC.DISBURSED.AMOUNT'])
```

Out[87]: <AxesSubplot: >

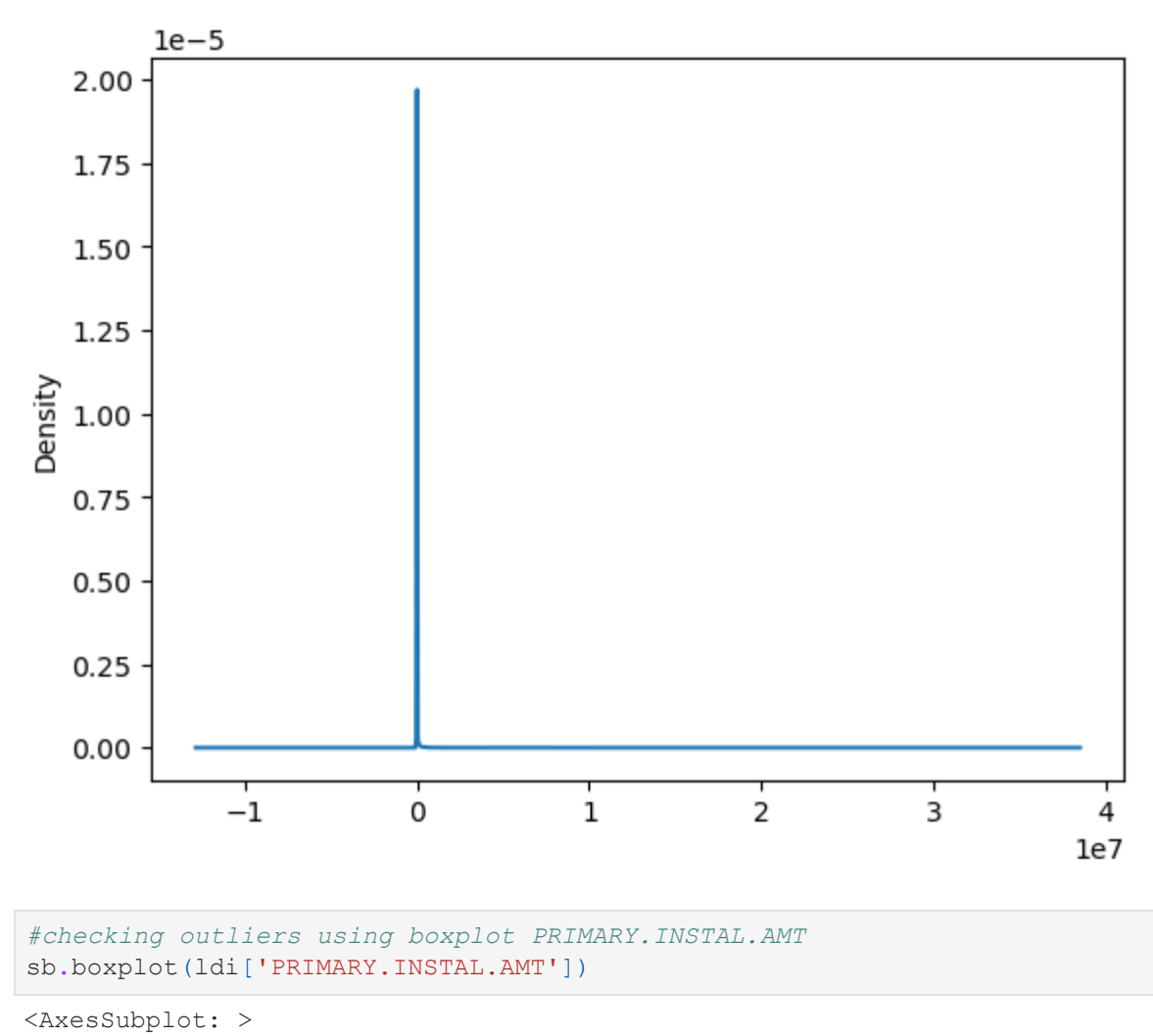


In [88]:
```
#replacing with median values

medsda=ldi['SEC.DISBURSED.AMOUNT'].median()
print(medsda)
ldi['SEC.DISBURSED.AMOUNT'].replace(np.nan, medsda, inplace=True)
```

39462.0

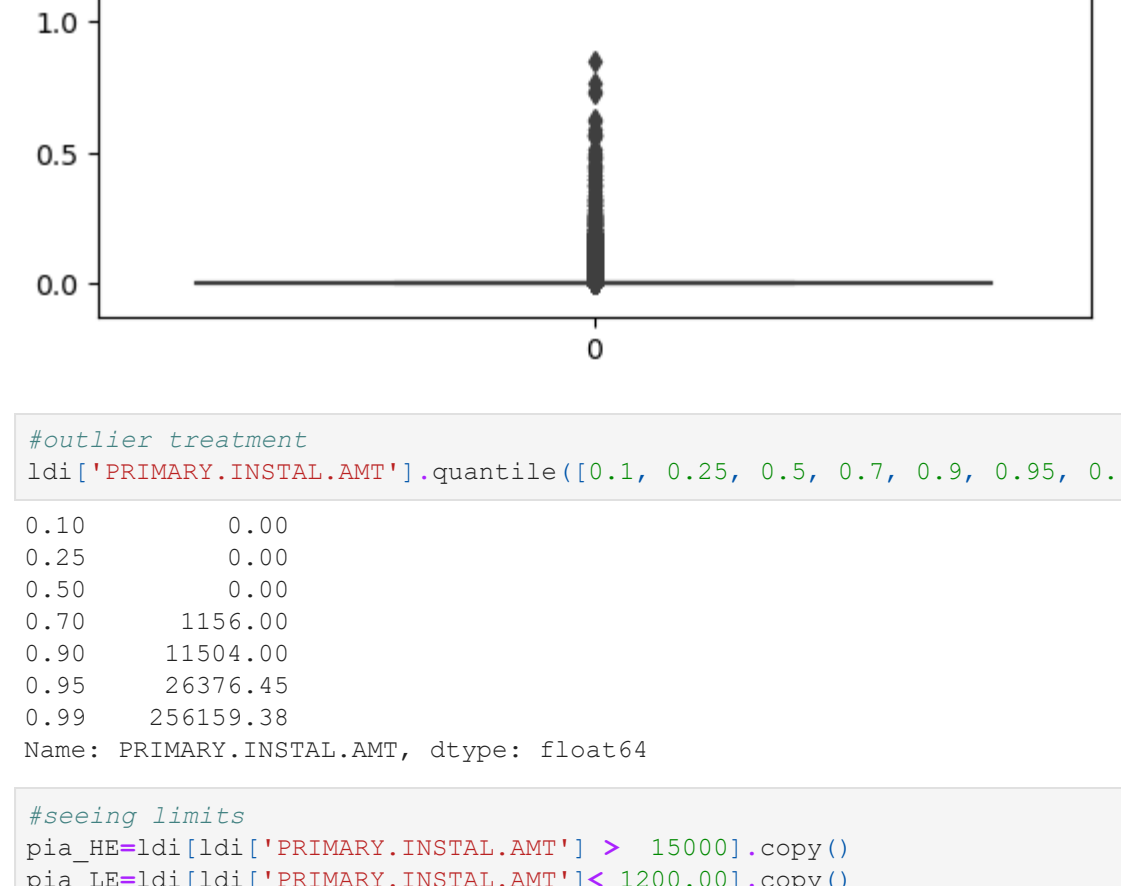
In [89]:
```
#checking distribution for PRIMARY.INSTAL.AMT
ldi['PRIMARY.INSTAL.AMT'].plot.density()
```

Out[89]: <AxesSubplot: ylabel='Density'>



In [90]:
```
#checking outliers using boxplot PRIMARY.INSTAL.AMT
sb.boxplot(ldi['PRIMARY.INSTAL.AMT'])
```

Out[90]: <AxesSubplot: >



In [91]:
```
#outlier treatment
ldi['PRIMARY.INSTAL.AMT'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

Out[91]:
```
0.10         0.00
0.25         0.00
0.50         0.00
0.70      1156.00
0.90     11504.00
0.95     26376.45
0.99    256159.38
Name: PRIMARY.INSTAL.AMT, dtype: float64
```

In [92]:
```
#seeing limits
pia_HE=ldi[ldi['PRIMARY.INSTAL.AMT'] >  15000].copy()
pia_LE=ldi[ldi['PRIMARY.INSTAL.AMT']< 1200.00].copy()
```

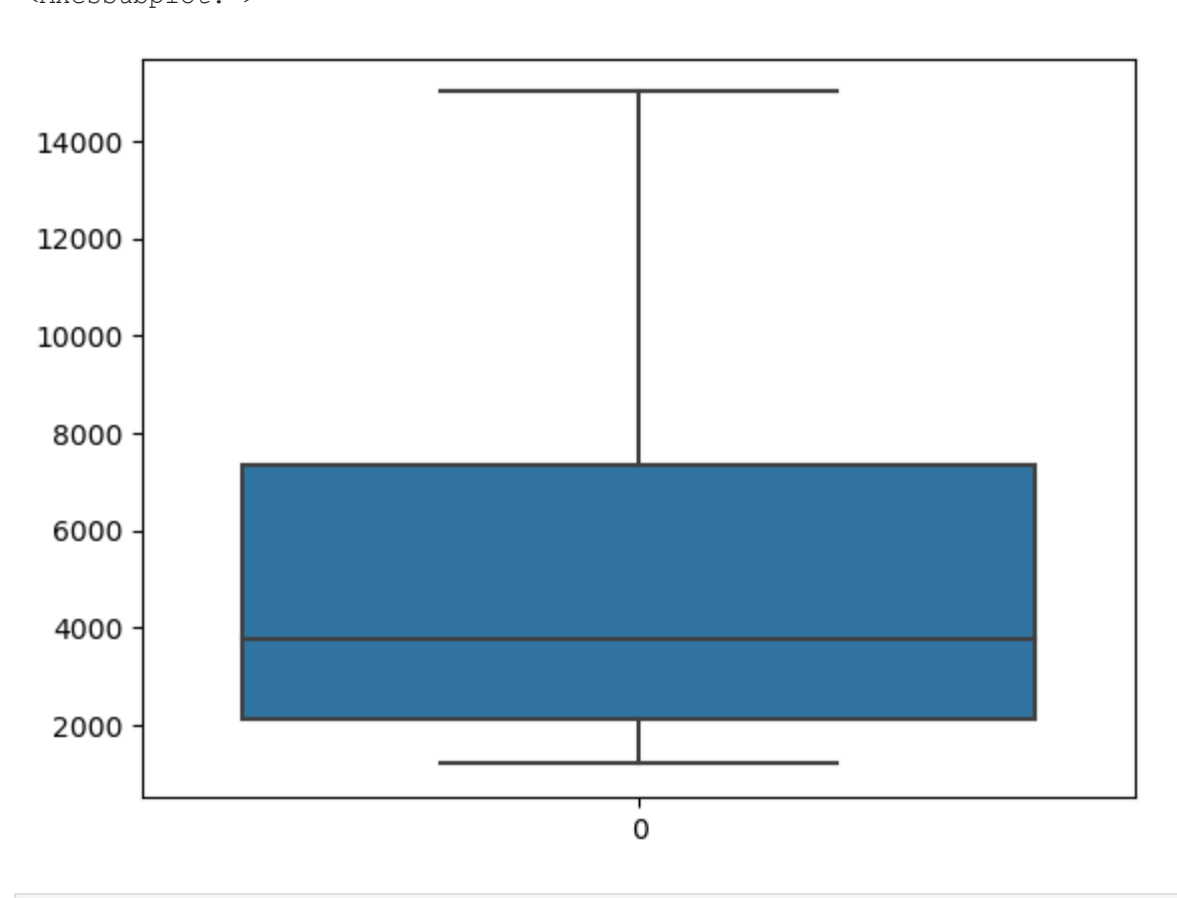
In [93]:
```
#putting them at one place
print(len(pia_HE))
print(len(pia_LE))
```

```
18822
163608
```

In [94]:
```
#removing outliers
for x in ldi['PRIMARY.INSTAL.AMT']:
    if x >  15000:
        ldi['PRIMARY.INSTAL.AMT'].replace(x,np.nan,inplace=True)
for x in ldi['PRIMARY.INSTAL.AMT']:
    if x < 1200.00:
        ldi['PRIMARY.INSTAL.AMT'].replace(x,np.nan,inplace=True)
```

In [95]:
```
#checking outliers gone or not
sb.boxplot(ldi['PRIMARY.INSTAL.AMT'])
```

Out[95]: <AxesSubplot: >

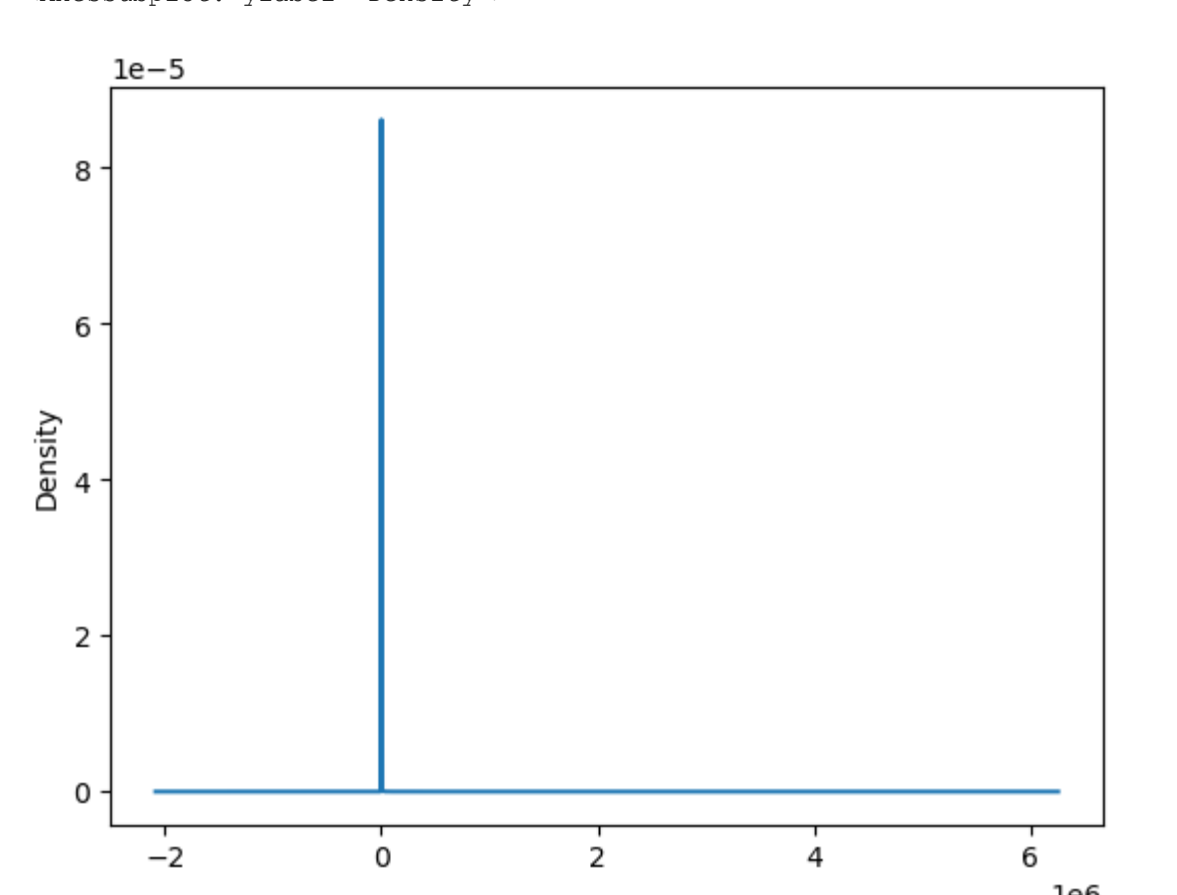


```
In [96]: #replacing with median values

medpia=ldi['PRIMARY.INSTAL.AMT'].median()
print(medpia)
ldi['PRIMARY.INSTAL.AMT'].replace(np.nan, medpia, inplace=True)
```

3786.0

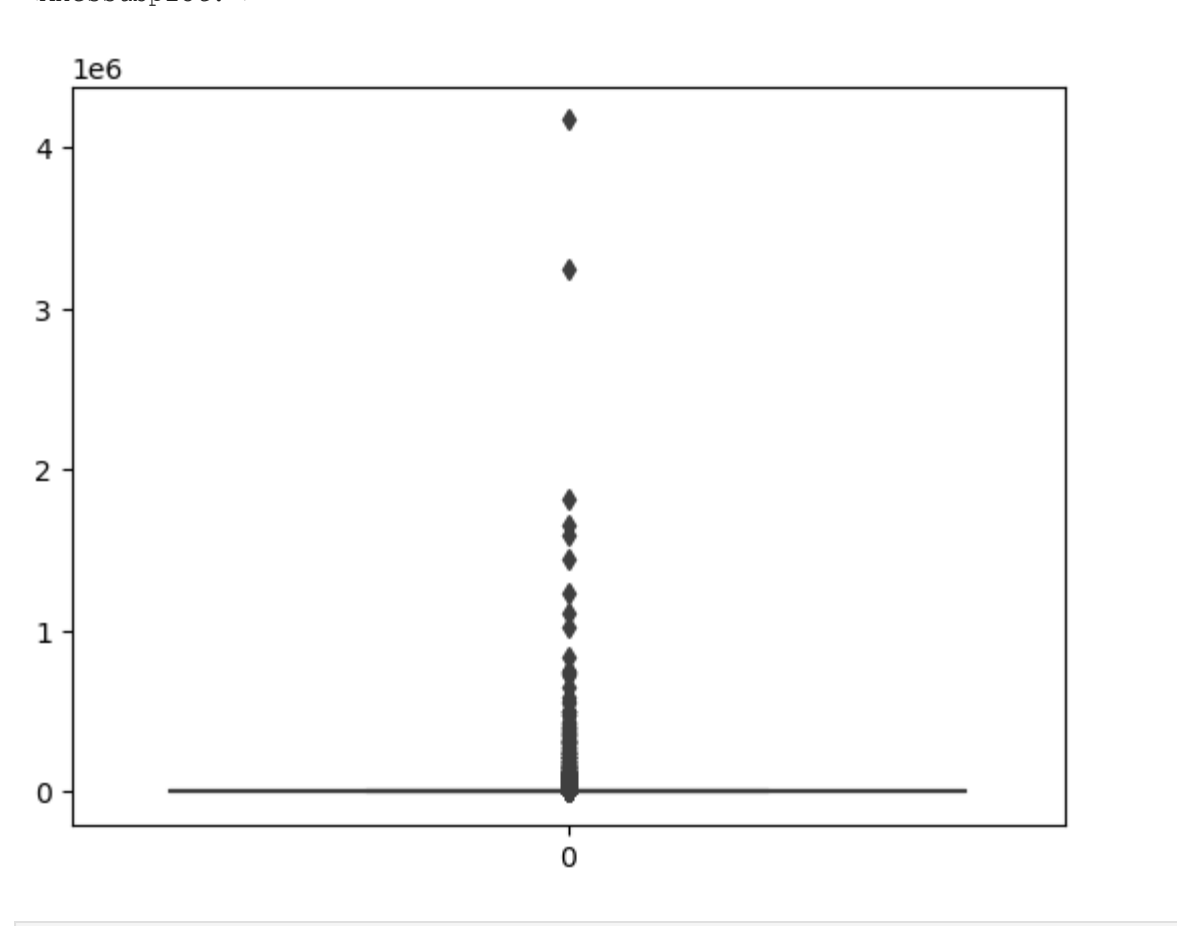
```
In [97]: #checking distribution for SEC.INSTAL.AMT
ldi['SEC.INSTAL.AMT'].plot.density()
```

Out[97]: <AxesSubplot: ylabel='Density'>



```
In [98]: #checking outliers using boxplot SEC.INSTAL.AMT
sb.boxplot(ldi['SEC.INSTAL.AMT'])
```

Out[98]: <AxesSubplot: >



```
In [99]: #outlier treatment
ldi['SEC.INSTAL.AMT'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

```
Out[99]:
0.10    0.0
0.25    0.0
0.50    0.0
0.70    0.0
0.90    0.0
0.95    0.0
0.99    0.0
Name: SEC.INSTAL.AMT, dtype: float64
```

```
In [100…: #seeing limits
sia_HE=ldi[ldi['SEC.INSTAL.AMT'] > 8000.00].copy()
sia_LE=ldi[ldi['SEC.INSTAL.AMT'] < 100.00].copy()
```

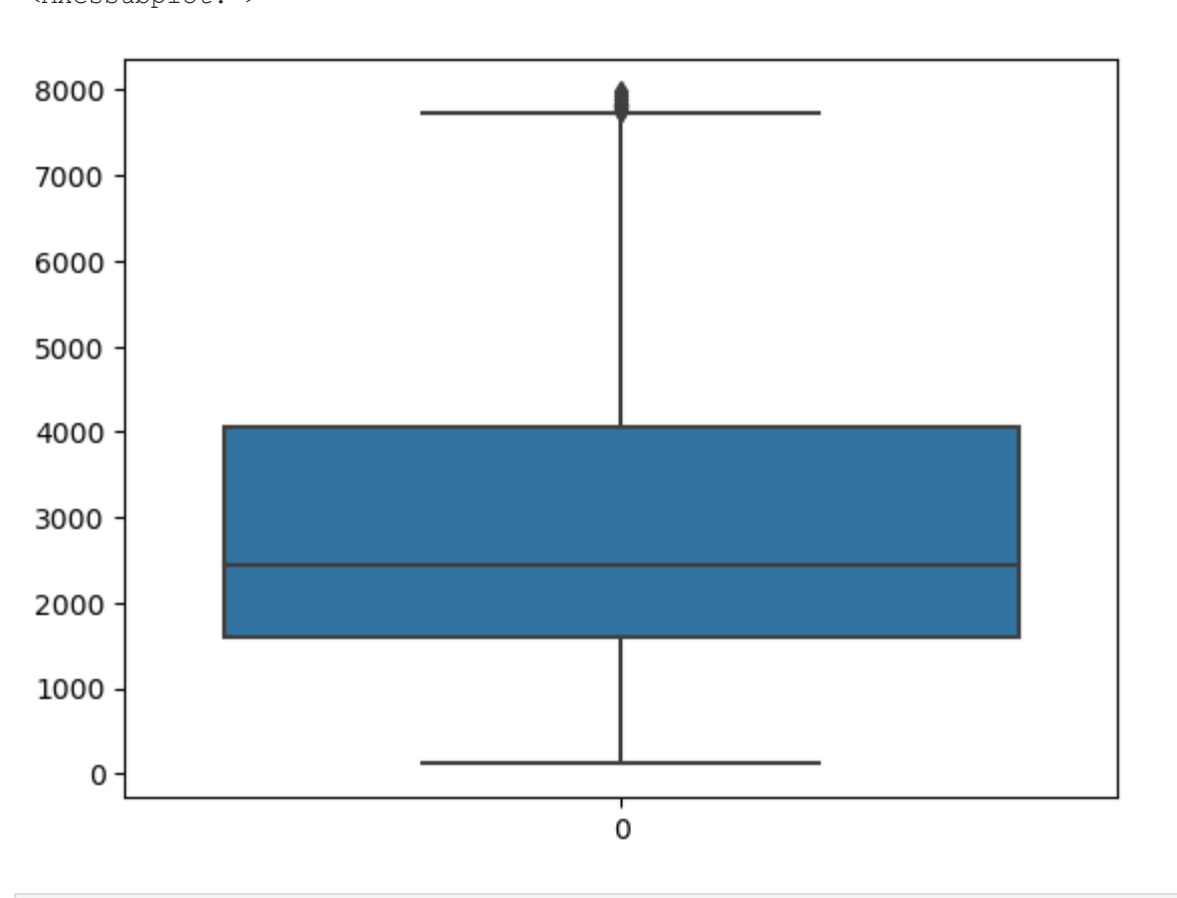
```
In [101…: #putting them at one place
print(len(sia_HE))
print(len(sia_LE))
```

```
813
230974
```

```
In [102…: #removing outliers
for x in ldi['SEC.INSTAL.AMT'] :
    if x > 8000.00:
        ldi['SEC.INSTAL.AMT'] .replace(x,np.nan,inplace=True)
for x in ldi['SEC.INSTAL.AMT'] :
    if x < 100.00:
        ldi['SEC.INSTAL.AMT'] .replace(x,np.nan,inplace=True)
```

```
In [103…: #checking outliers gone or not
sb.boxplot(ldi['SEC.INSTAL.AMT'])
```

Out[103]: <AxesSubplot: >

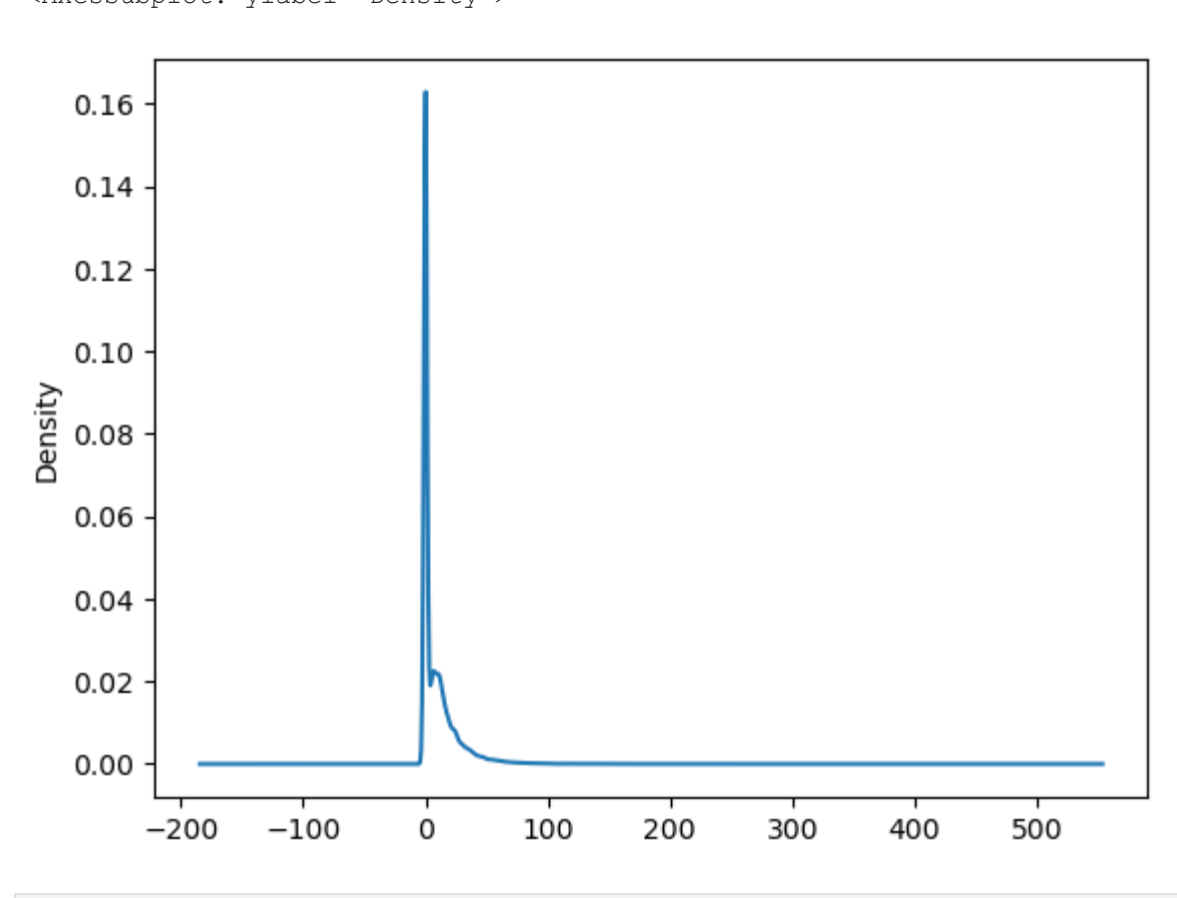


```
In [104…: #replacing with median values

medsia=ldi['SEC.INSTAL.AMT'].median()
print(medsia)
ldi['SEC.INSTAL.AMT'].replace(np.nan, medsia, inplace=True)
```

2440.0

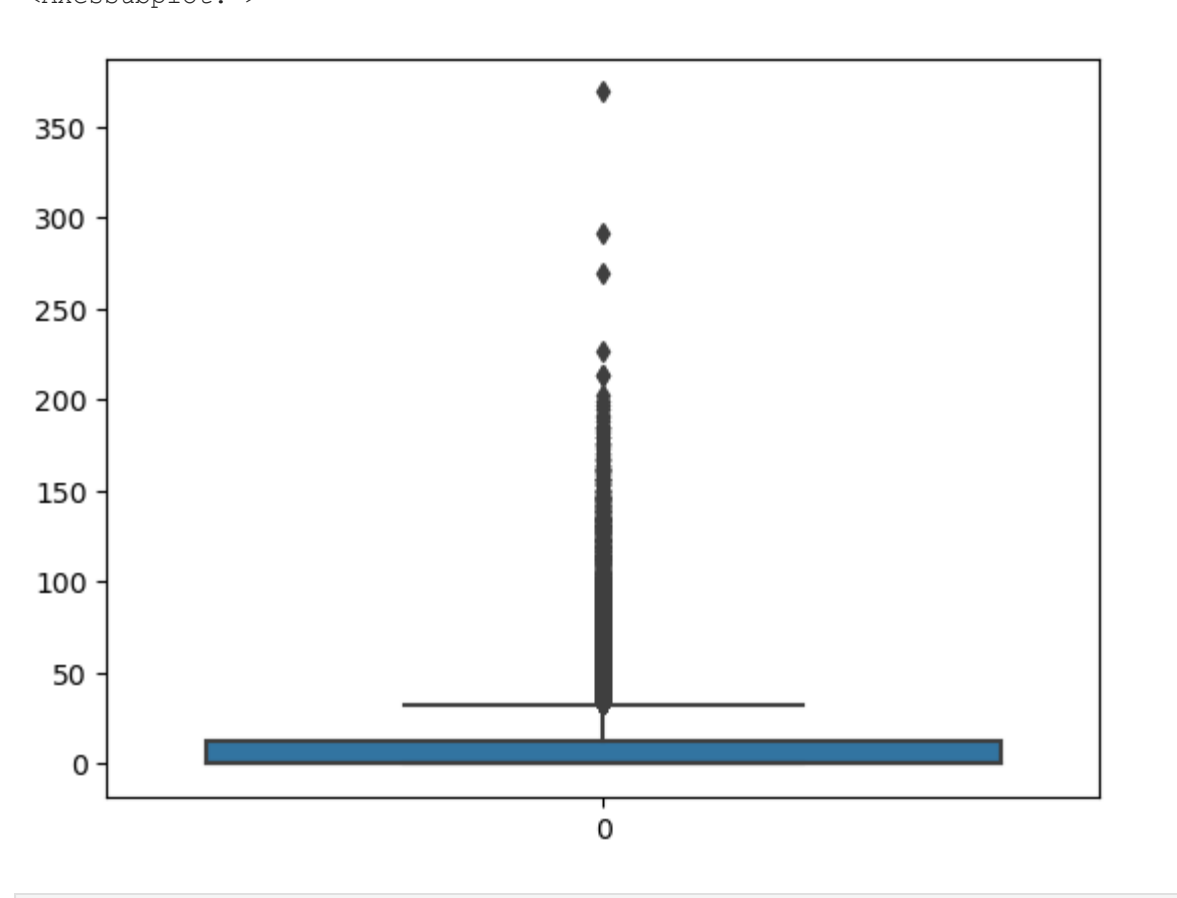
```
In [105…: #checking distribution for Acc_age_months
ldi['Acc_age_months'].plot.density()
```

Out[105]: <AxesSubplot: ylabel='Density'>



```
In [106…: #checking outliers using boxplot disbursed_amount
sb.boxplot(ldi['Acc_age_months'])
```

Out[106]: <AxesSubplot: >



```
In [107…: #outlier treatment
ldi['Acc_age_months'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

```
Out[107]:
0.10     0.0
0.25     0.0
0.50     0.0
0.70    11.0
0.90    26.0
0.95    38.0
0.99    68.0
Name: Acc_age_months, dtype: float64
```

```
In [108…: #seeing limits
aam_HE=ldi[ldi['Acc_age_months'] > 35].copy()
aam_LE=ldi[ldi['Acc_age_months']< 11].copy()
```

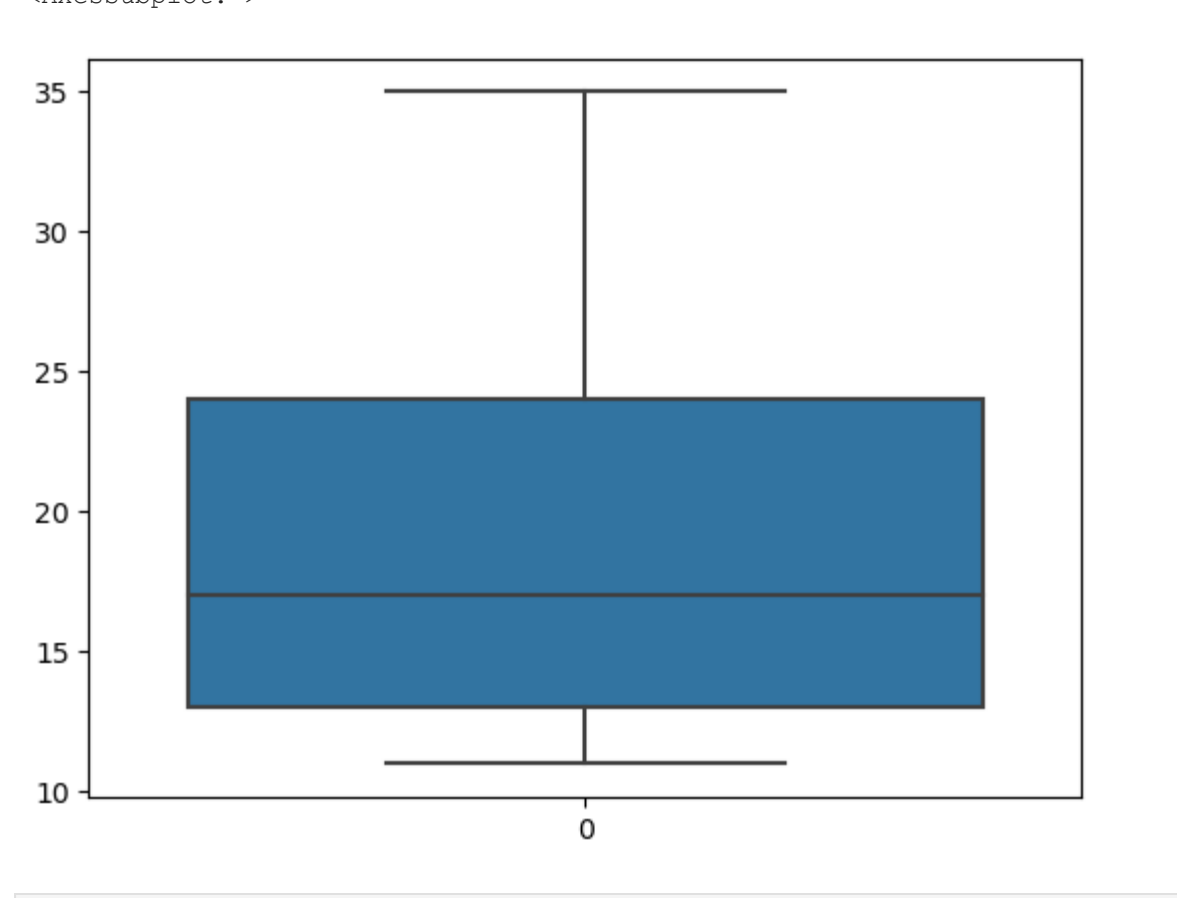
```
In [109…: #putting them at one place
print(len(aam_HE))
print(len(aam_LE))
```

```
13275
163153
```

```
In [110…: #removing outliers
for x in ldi['Acc_age_months']:
    if x > 35:
        ldi['Acc_age_months'].replace(x,np.nan,inplace=True)
for x in ldi['Acc_age_months']:
    if x < 11:
        ldi['Acc_age_months'].replace(x,np.nan,inplace=True)
```

```
In [111…: #checking outliers gone or not
sb.boxplot(ldi['Acc_age_months'])
```

Out[111]: <AxesSubplot: >

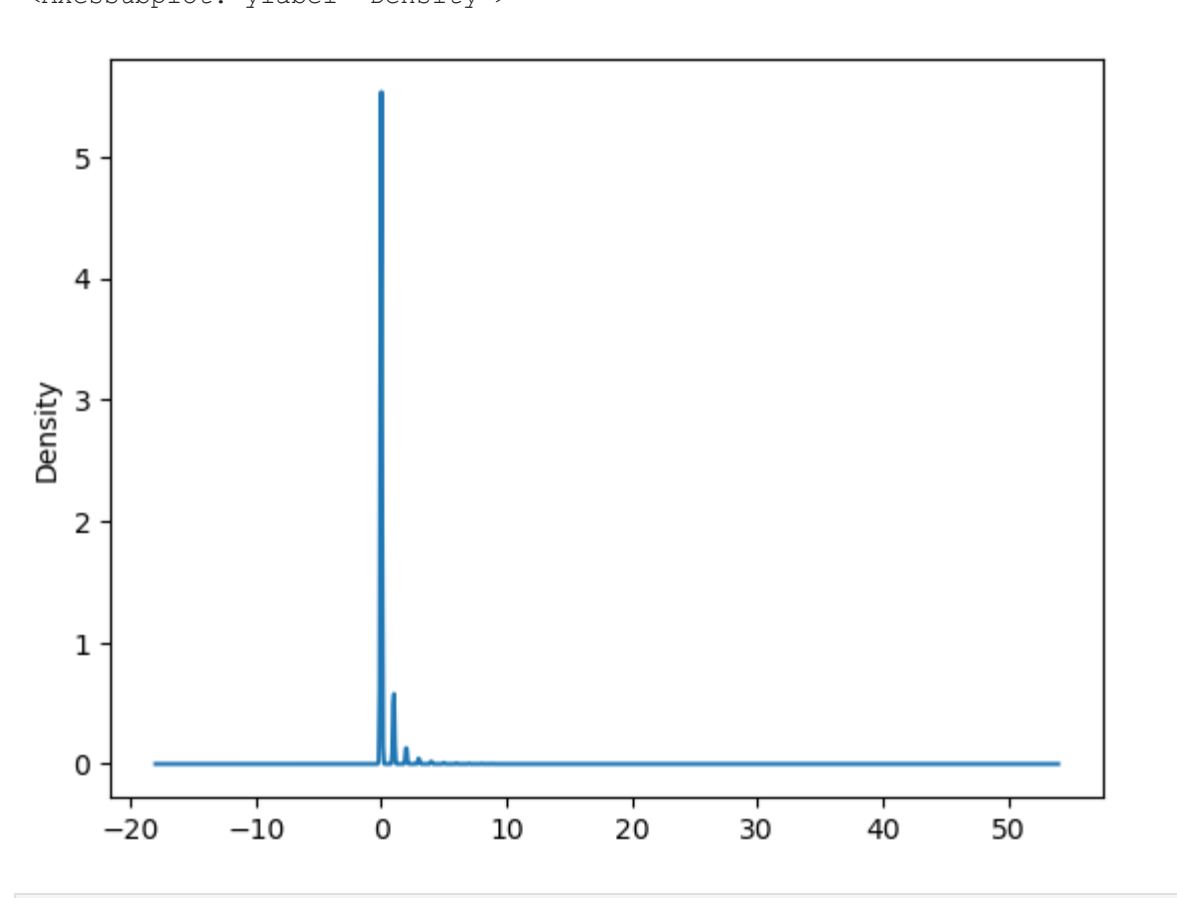


```
In [112…: #replacing with median values

medaam=ldi['Acc_age_months'].median()
print(medaam)
ldi['Acc_age_months'].replace(np.nan, medaam, inplace=True)
```

17.0

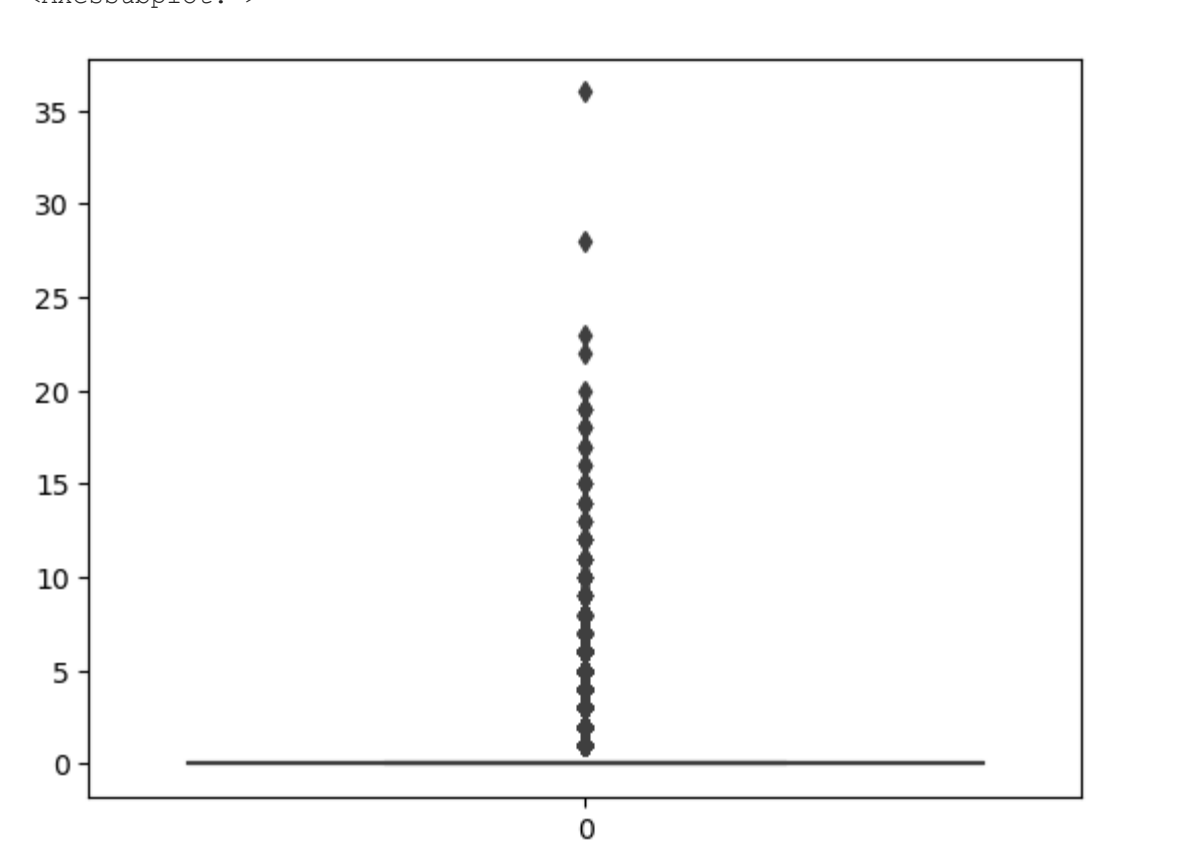
```
In [113…: #checking distribution for NO.OF_INQUIRIES
ldi['NO.OF_INQUIRIES'].plot.density()
```

Out[113]: <AxesSubplot: ylabel='Density'>



```
In [114…: #checking outliers using boxplot NO.OF_INQUIRIES
sb.boxplot(ldi['NO.OF_INQUIRIES'])
```

Out[114]: <AxesSubplot: >



```
In [115…: #outlier treatment
ldi['NO.OF_INQUIRIES'].quantile([0.1, 0.25, 0.5, 0.7, 0.9, 0.95, 0.99])
```

```
Out[115]:
0.10    0.0
0.25    0.0
0.50    0.0
0.70    0.0
0.90    1.0
0.95    1.0
0.99    3.0
Name: NO.OF_INQUIRIES, dtype: float64
```

```
In [116…: #seeing limits
noi_HE=ldi[ldi['NO.OF_INQUIRIES'] > 5].copy()
noi_LE=ldi[ldi['NO.OF_INQUIRIES']< 1].copy()
```

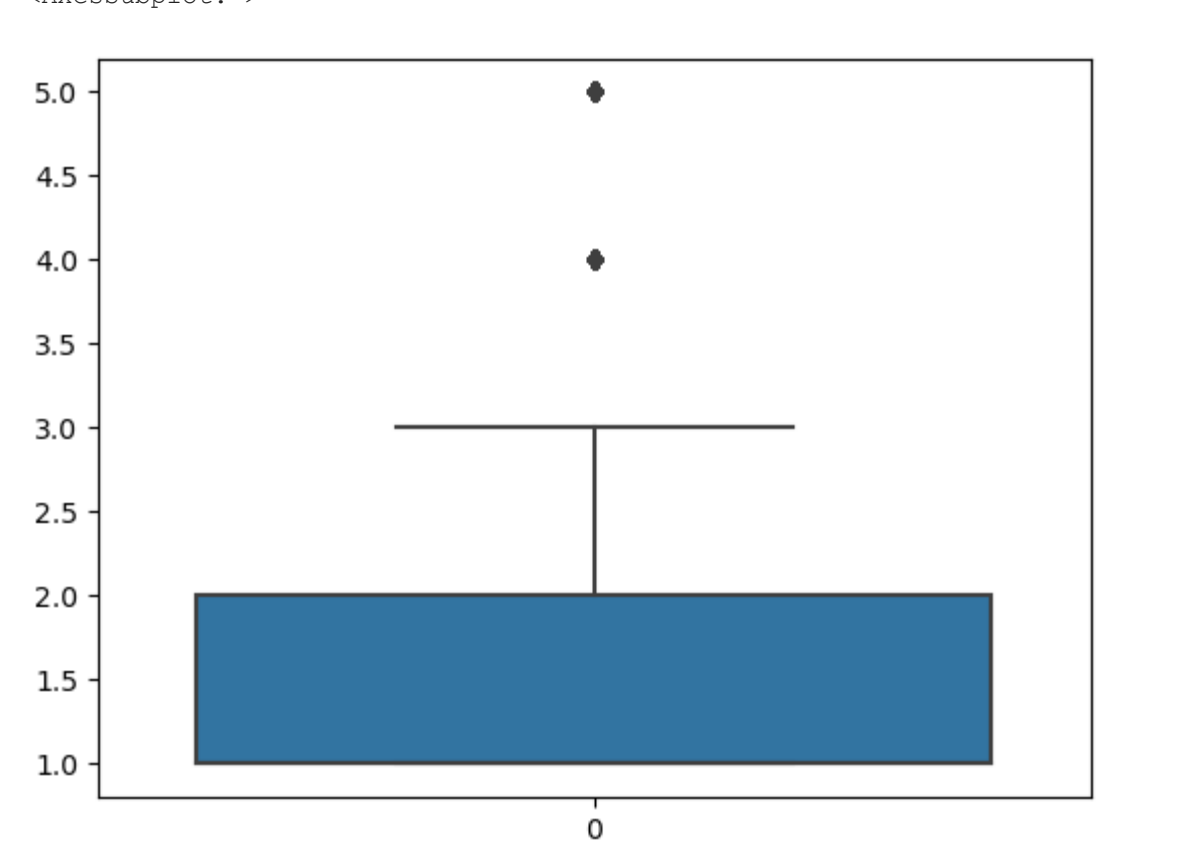
```
In [117…: #putting them at one place
print(len(noi_HE))
print(len(noi_LE))
```

```
629
201961
```

```
In [118…: #removing outliers
for x in ldi['NO.OF_INQUIRIES']:
    if x > 5:
        ldi['NO.OF_INQUIRIES'].replace(x,np.nan,inplace=True)
for x in ldi['NO.OF_INQUIRIES']:
    if x < 1:
        ldi['NO.OF_INQUIRIES'].replace(x,np.nan,inplace=True)
```

```
In [119…: #checking outliers gone or not
sb.boxplot(ldi['NO.OF_INQUIRIES'])
```

Out[119]: <AxesSubplot: >



```
In [120…: #replacing with median values

mednoi=ldi['NO.OF_INQUIRIES'].median()
print(mednoi)
ldi['NO.OF_INQUIRIES'].replace(np.nan, mednoi, inplace=True)
```

1.0

```
In [121…: #dropping useless columns
ldi= ldi.drop(['UniqueID','Current_pincode_ID','Date.of.Birth','Employee_code_ID'],axis=1)
ldi.head()
```

Out[121]:

	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Age	Employment.Type	DisbursalDate	State_ID	...	SEC.
0	50578.0	68763.0	89.55	67	22807	45	39.271233	Salaried	2018-08-03	6	...	
1	53278.0	61360.0	89.63	67	22807	45	37.624658	Self employed	2018-08-01	6	...	
2	52378.0	68763.0	88.39	67	22807	45	45.336986	Self employed	2018-09-26	6	...	
3	46349.0	61500.0	76.42	67	22807	45	34.852055	Salaried	2018-09-23	6	...	
4	43594.0	68763.0	79.77	67	22744	86	28.731507	Self employed	2018-10-08	6	...	

5 rows × 38 columns

```
In [122…: #transforming data types to their appropriate types
obj_vars=['loan_default','branch_id','supplier_id','manufacturer_id','State_ID','MobileNo_Avl_Flag','Aadhar_fla
for x in obj_vars:
    ldi[x]=ldi[x].astype(str)
print(ldi.dtypes)
```

```
disbursed_amount                              float64
asset_cost                                    float64
ltv                                           float64
branch_id                                      object
supplier_id                                    object
manufacturer_id                                object
Age                                           float64
Employment.Type                                object
DisbursalDate                          datetime64[ns]
State_ID                                       object
MobileNo_Avl_Flag                              object
Aadhar_flag                                    object
PAN_flag                                       object
VoterID_flag                                   object
Driving_flag                                   object
Passport_flag                                  object
Profile_strength                               object
PERFORM_CNS.SCORE                               int64
PERFORM_CNS.SCORE.DESCRIPTION                  object
PRI.NO.OF.ACCTS                                 int64
PRI.ACTIVE.ACCTS                                int64
PRI.OVERDUE.ACCTS                               int64
PRI.CURRENT.BALANCE                           float64
PRI.SANCTIONED.AMOUNT                         float64
PRI.DISBURSED.AMOUNT                          float64
SEC.NO.OF.ACCTS                                 int64
SEC.ACTIVE.ACCTS                                int64
SEC.OVERDUE.ACCTS                               int64
SEC.CURRENT.BALANCE                           float64
SEC.SANCTIONED.AMOUNT                         float64
SEC.DISBURSED.AMOUNT                          float64
PRIMARY.INSTAL.AMT                            float64
SEC.INSTAL.AMT                                float64
NEW.ACCTS.IN.LAST.SIX.MONTHS                    int64
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS             int64
Acc_age_months                                float64
NO.OF_INQUIRIES                               float64
loan_default                                   object
dtype: object
```

```
In [123…: #getting non categorical variables
non_cat=[]
for x in ldi.columns:
    if x in obj_vars:
        pass
    else:
        non_cat.append(x)
print(non_cat)
```

```
['disbursed_amount', 'asset_cost', 'ltv', 'Age', 'Employment.Type', 'DisbursalDate', 'Profile_strength', 'PERFO
RM_CNS.SCORE', 'PERFORM_CNS.SCORE.DESCRIPTION', 'PRI.NO.OF.ACCTS', 'PRI.ACTIVE.ACCTS', 'PRI.OVERDUE.ACCTS', 'PR
I.CURRENT.BALANCE', 'PRI.SANCTIONED.AMOUNT', 'PRI.DISBURSED.AMOUNT', 'SEC.NO.OF.ACCTS', 'SEC.ACTIVE.ACCTS', 'SE
C.OVERDUE.ACCTS', 'SEC.CURRENT.BALANCE', 'SEC.SANCTIONED.AMOUNT', 'SEC.DISBURSED.AMOUNT', 'PRIMARY.INSTAL.AMT',
'SEC.INSTAL.AMT', 'NEW.ACCTS.IN.LAST.SIX.MONTHS', 'DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS', 'Acc_age_months', 'NO.
OF_INQUIRIES']
```

Step 3:Exploratory data analysis

In this step, we are going to explore the dataset. Perform hypothesis tests,bivariate analysis and check for correlation between variables.

```
In [124…: #summary stats
ldi.describe().round(2)
```

Out[124]:

	disbursed_amount	asset_cost	ltv	Age	PERFORM_CNS.SCORE	PRI.NO.OF.ACCTS	PRI.ACTIVE.ACCTS	PRI.OVERDUE.ACCTS	P
count	233154.00	233154.00	233154.00	233154.00	233154.00	233154.00	233154.00	233154.00	
mean	53826.46	68860.58	79.88	39.01	289.46	2.44	1.04	0.16	
std	6198.68	3150.76	5.07	9.83	338.37	5.22	1.94	0.55	
min	39794.00	61330.00	68.88	22.46	0.00	0.00	0.00	0.00	
25%	50078.00	67743.25	76.80	30.88	0.00	0.00	0.00	0.00	
50%	53803.00	68763.00	79.77	37.27	0.00	0.00	0.00	0.00	
75%	57363.00	69800.00	83.38	45.94	678.00	3.00	1.00	0.00	
max	68882.00	76807.00	89.95	73.59	890.00	453.00	144.00	25.00	

8 rows × 23 columns

```
In [125…: #seeing correlation of numerical variables

cordata =ldi[non_cat].corr()
cordata.round(2)
```

Out[125]:

	disbursed_amount	asset_cost	ltv	Age	PERFORM_CNS.SCORE	PRI.NO.OF.ACCTS	PRI.ACTIVE.ACCTS
disbursed_amount	1.00	0.27	0.30	-0.01	0.01	0.04	0.03
asset_cost	0.27	1.00	-0.04	-0.01	-0.03	-0.00	-0.01
ltv	0.30	-0.04	1.00	0.08	0.07	0.06	0.06
Age	-0.01	-0.01	0.08	1.00	0.17	0.17	0.14
PERFORM_CNS.SCORE	0.01	-0.03	0.07	0.17	1.00	0.42	0.47
PRI.NO.OF.ACCTS	0.04	-0.00	0.06	0.17	0.42	1.00	0.75
PRI.ACTIVE.ACCTS	0.03	-0.01	0.06	0.14	0.47	0.75	1.00
PRI.OVERDUE.ACCTS	0.02	-0.00	0.03	0.14	0.11	0.35	0.38
PRI.CURRENT.BALANCE	0.00	-0.00	0.00	0.02	0.03	0.04	0.06
PRI.SANCTIONED.AMOUNT	0.00	-0.01	0.01	0.08	0.14	0.19	0.25
PRI.DISBURSED.AMOUNT	0.00	-0.00	0.02	0.06	0.09	0.17	0.22
SEC.NO.OF.ACCTS	-0.01	-0.02	0.00	0.02	0.06	0.06	0.06
SEC.ACTIVE.ACCTS	-0.01	-0.02	0.00	0.02	0.05	0.05	0.06
SEC.OVERDUE.ACCTS	-0.01	-0.01	0.00	0.02	0.04	0.03	0.04
SEC.CURRENT.BALANCE	-0.00	0.00	-0.00	0.00	0.00	0.01	0.01
SEC.SANCTIONED.AMOUNT	0.00	0.00	-0.00	-0.00	-0.00	0.01	0.01
SEC.DISBURSED.AMOUNT	0.00	0.00	0.00	0.00	-0.00	0.00	0.00
PRIMARY.INSTAL.AMT	0.01	-0.00	0.02	0.08	0.15	0.20	0.23
SEC.INSTAL.AMT	-0.00	-0.00	0.00	0.00	0.02	0.01	0.02
NEW.ACCTS.IN.LAST.SIX.MONTHS	0.03	-0.01	0.06	0.03	0.35	0.54	0.70
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS	0.01	-0.00	0.02	0.08	0.16	0.31	0.38
Acc_age_months	-0.01	-0.00	-0.01	0.10	0.10	0.01	0.06
NO.OF_INQUIRIES	0.02	-0.01	0.04	0.00	0.10	0.15	0.16

23 rows × 23 columns

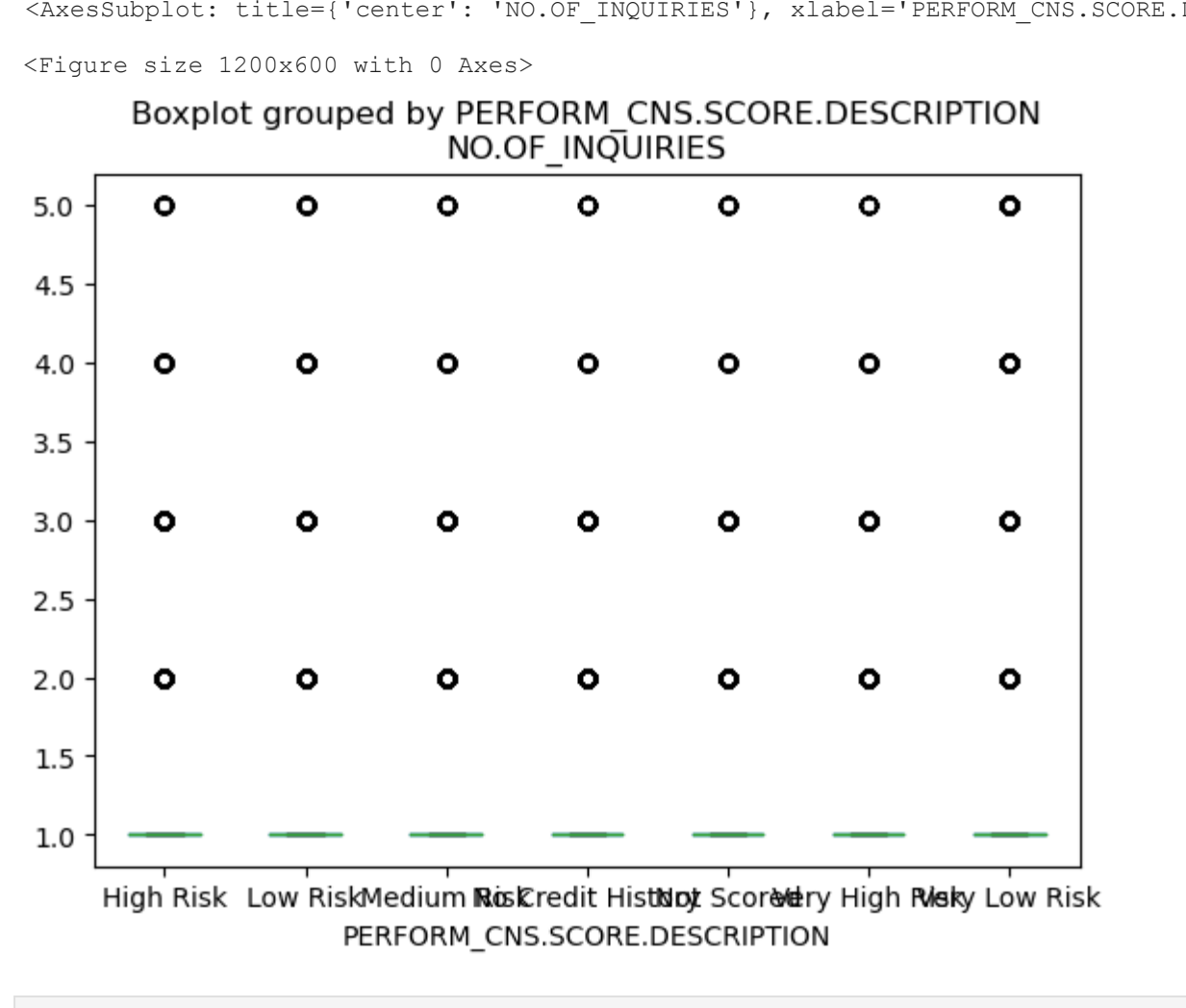
Bivariate Analysis

We are going to be analysing two variables with respect to one another here.

```
In [126…: #are high enquiries a red flag



plt.figure(figsize=[12,6])
ldi.boxplot(by="PERFORM_CNS.SCORE.DESCRIPTION",column="NO.OF_INQUIRIES", grid = False)
```

Out[126]: <AxesSubplot: title={'center': 'NO.OF_INQUIRIES'}, xlabel='PERFORM_CNS.SCORE.DESCRIPTION'>

<Figure size 1200x600 with 0 Axes>

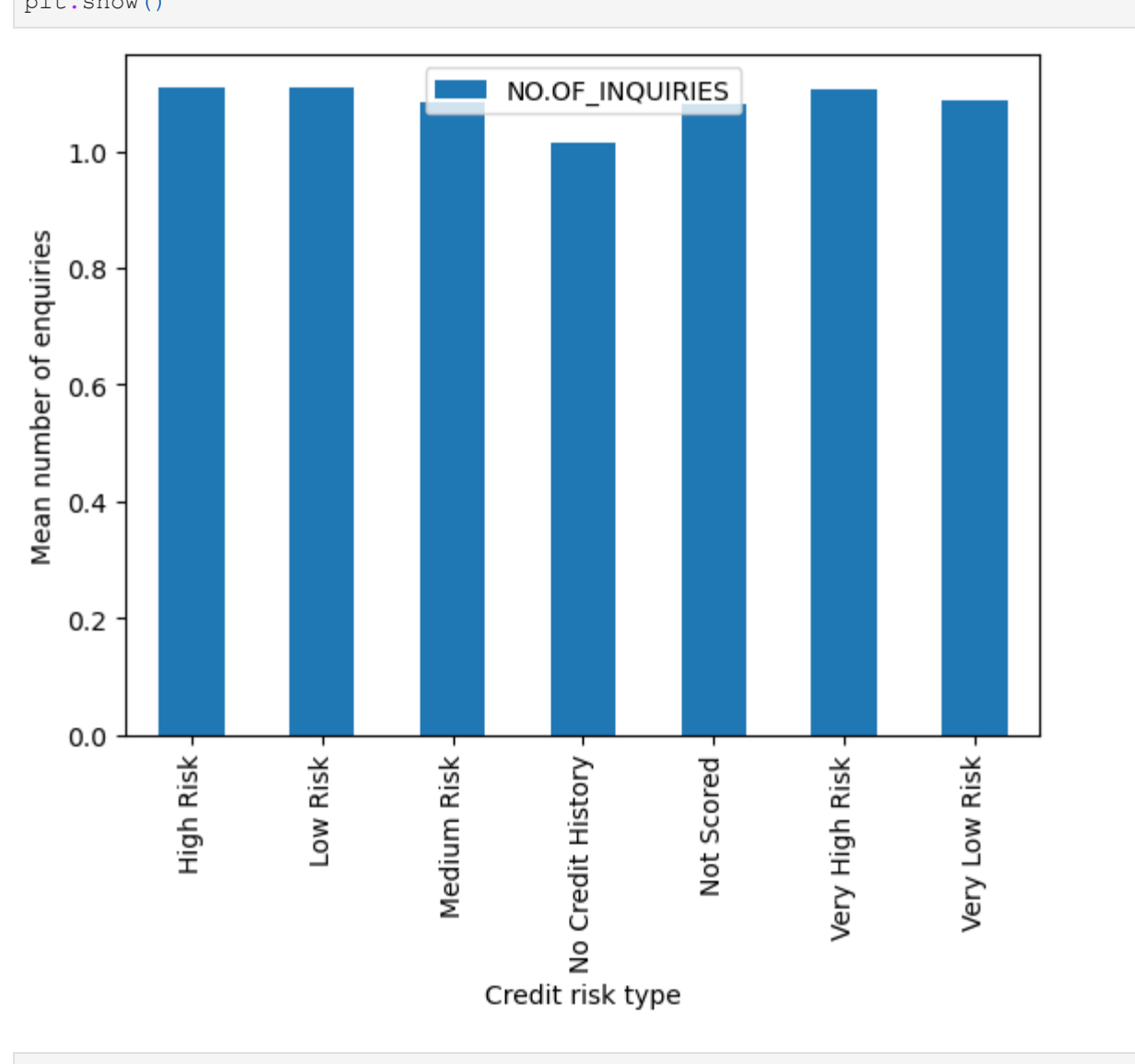



```
In [127…: #checking if enquiries differs according to the risk category'
enqcnt=ldi['NO.OF_INQUIRIES']
riskcat=ldi['PERFORM_CNS.SCORE.DESCRIPTION']
mod = ols("enqcnt ~ riskcat", data=ldi).fit()
anov_table = sm.stats.anova_lm(mod)
anov_table
```

Out[127]:

	df	sum_sq	mean_sq	F	PR(>F)
riskcat	6.0	380.521844	63.420307	622.73396	0.0
Residual	233147.0	23744.095126	0.101842	NaN	NaN

```
In [128…: #seeing the number of enquiries by credit risk
grouped = ldi.groupby('PERFORM_CNS.SCORE.DESCRIPTION').agg({'NO.OF_INQUIRIES': 'mean'})

grouped.plot(kind='bar')
plt.xlabel('Credit risk type')
plt.ylabel('Mean number of enquiries')
plt.show()
```



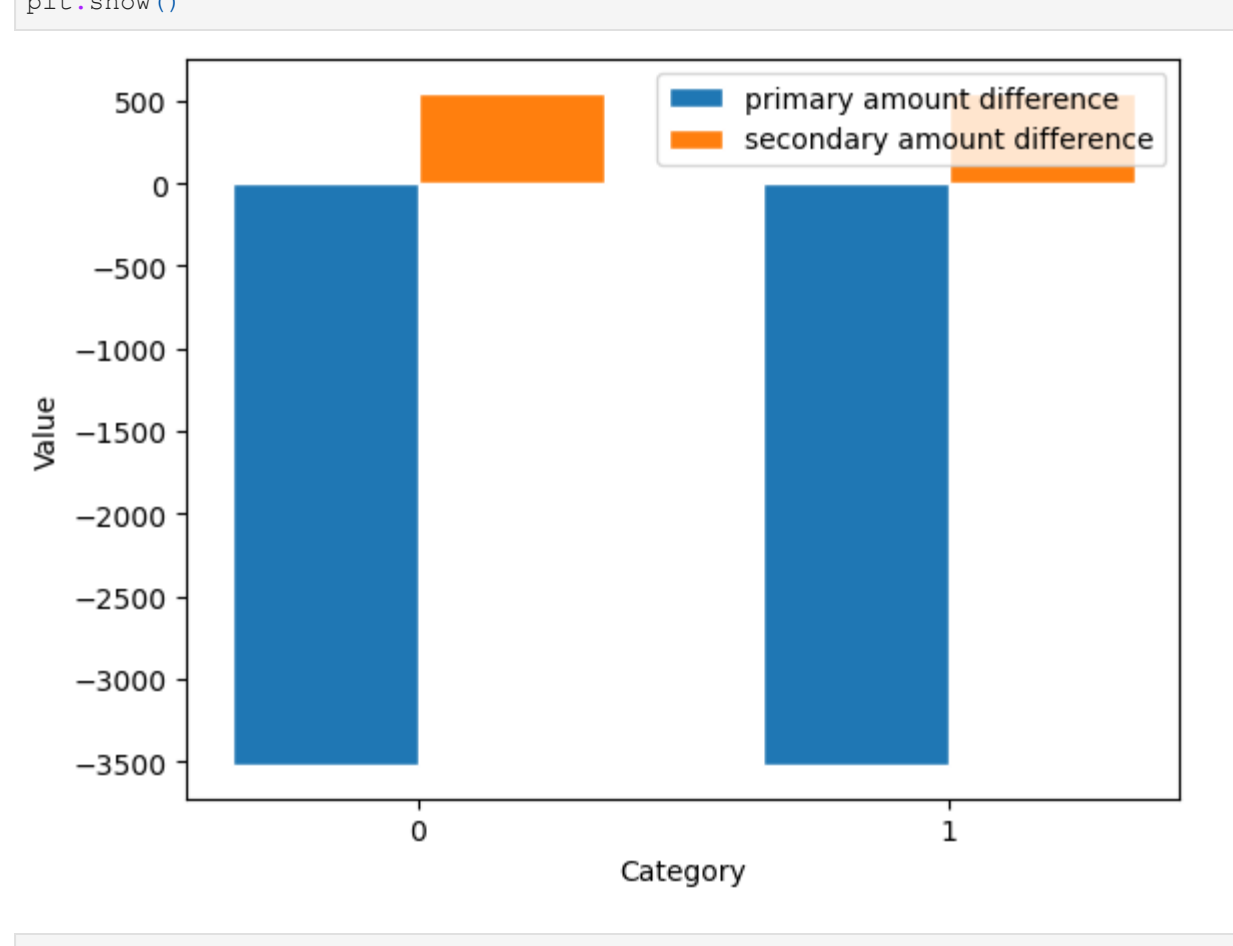

```
In [129…: #discrepancy calculation
ldi['pri_diff']=ldi['PRI.SANCTIONED.AMOUNT']-ldi['PRI.DISBURSED.AMOUNT']
ldi['sec_diff']=ldi['SEC.SANCTIONED.AMOUNT']-ldi['SEC.DISBURSED.AMOUNT']
ldi.head()
```

Out[129]:

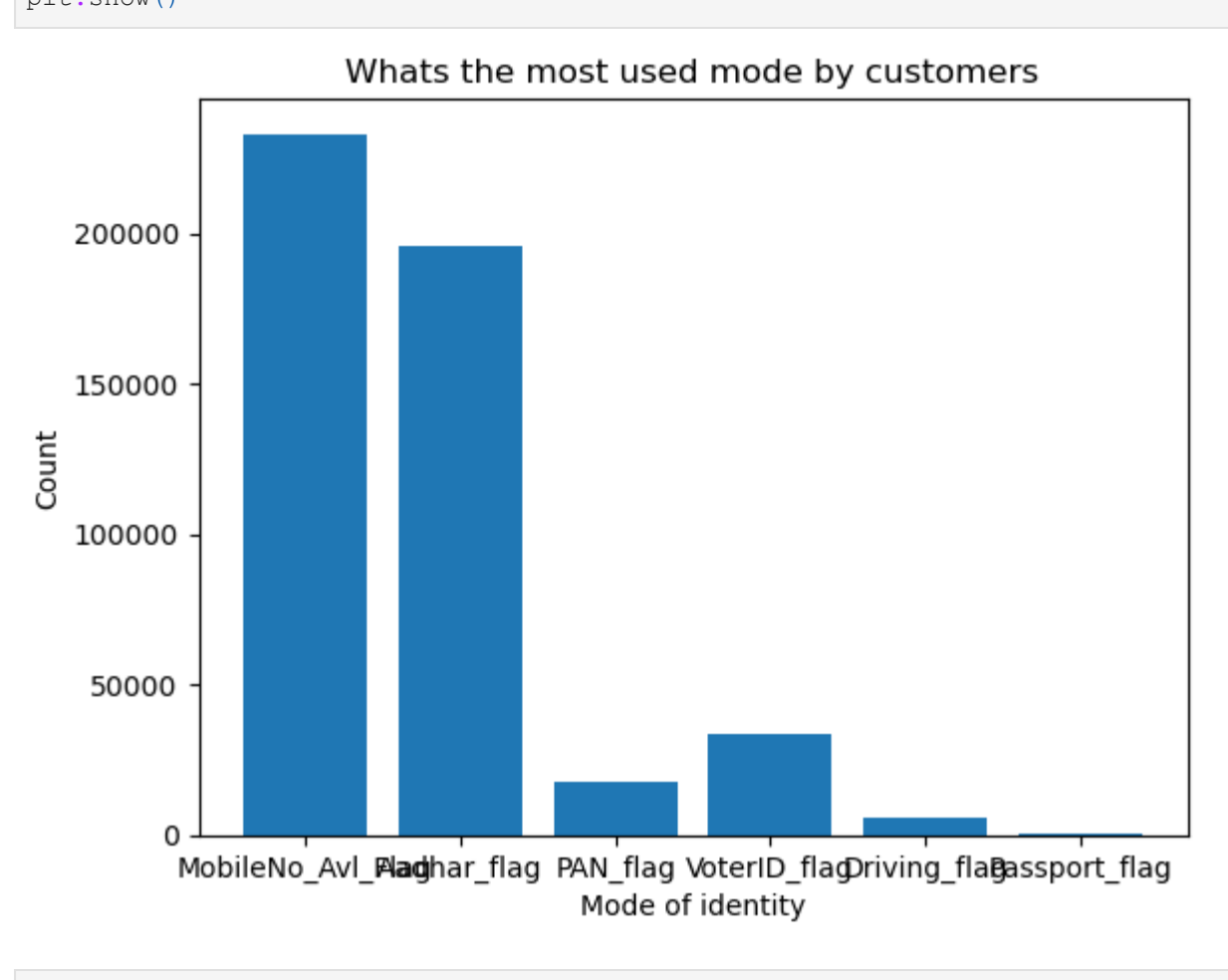
	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Age	Employment.Type	DisbursalDate	State_ID	...	SEC.
0	50578.0	68763.0	89.55	67	22807	45	39.271233	Salaried	2018-08-03	6	...	
1	53278.0	61360.0	89.63	67	22807	45	37.624658	Self employed	2018-08-01	6	...	
2	52378.0	68763.0	88.39	67	22807	45	45.336986	Self employed	2018-09-26	6	...	
3	46349.0	61500.0	76.42	67	22807	45	34.852055	Salaried	2018-09-23	6	...	
4	43594.0	68763.0	79.77	67	22744	86	28.731507	Self employed	2018-10-08	6	...	

5 rows × 40 columns

```
In [130…: #difference btw sanctioned amount disbursed amount btw primary and secondary across loan default types
ldi['pri_diff']=ldi['PRI.SANCTIONED.AMOUNT']-ldi['PRI.DISBURSED.AMOUNT']
ldi['sec_diff']=ldi['SEC.SANCTIONED.AMOUNT']-ldi['SEC.DISBURSED.AMOUNT']

ld=ldi['loan_default'].unique()

value1=ldi['pri_diff'].mean()
value2=ldi['sec_diff'].mean()
# Set the width of the bars
bar_width = 0.35

# Set the positions of the bars on the x-axis
r1 = np.arange(len(ld))
r2 = [x + bar_width for x in r1]

# Create the bar chart
plt.bar(r1, value1, width=bar_width, edgecolor='white', label='primary amount difference')
plt.bar(r2, value2, width=bar_width, edgecolor='white', label='secondary amount difference')

# Add xticks on the middle of the group bars
plt.xlabel('Category')
plt.ylabel('Value')
plt.xticks([r + bar_width/2 for r in range(len(ld))], ld)

# Add legend
plt.legend()

# Show the chart
plt.show()
```



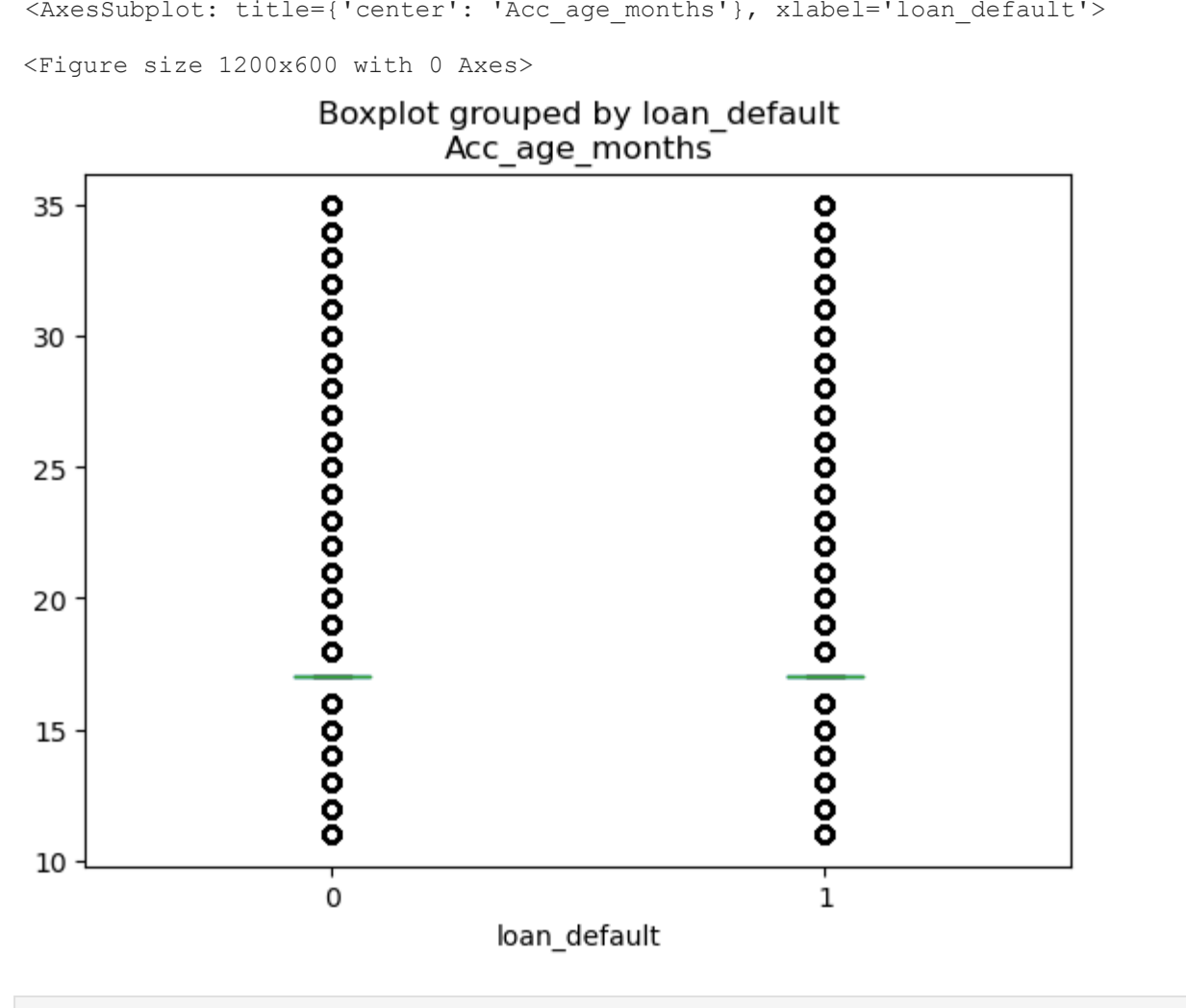

```
In [131…: #which is the most used mode for registration
modes= ldi[['MobileNo_Avl_Flag','Aadhar_flag','PAN_flag','VoterID_flag','Driving_flag','Passport_flag']]
modes.head()

counts = modes.apply(lambda x: x.value_counts()['1'])

# Create the bar chart
plt.bar(counts.index, counts.values)
plt.xlabel('Mode of identity')
plt.ylabel('Count')
plt.title("Whats the most used mode by customers")

# Show the chart
plt.show()
```




```
In [132…: #is account age responsible for default
plt.figure(figsize=[12,6])
ldi.boxplot(by="loan_default",column="Acc_age_months", grid = False)
```

Out[132]: <AxesSubplot: title={'center': 'Acc_age_months'}, xlabel='loan_default'>

<Figure size 1200x600 with 0 Axes>

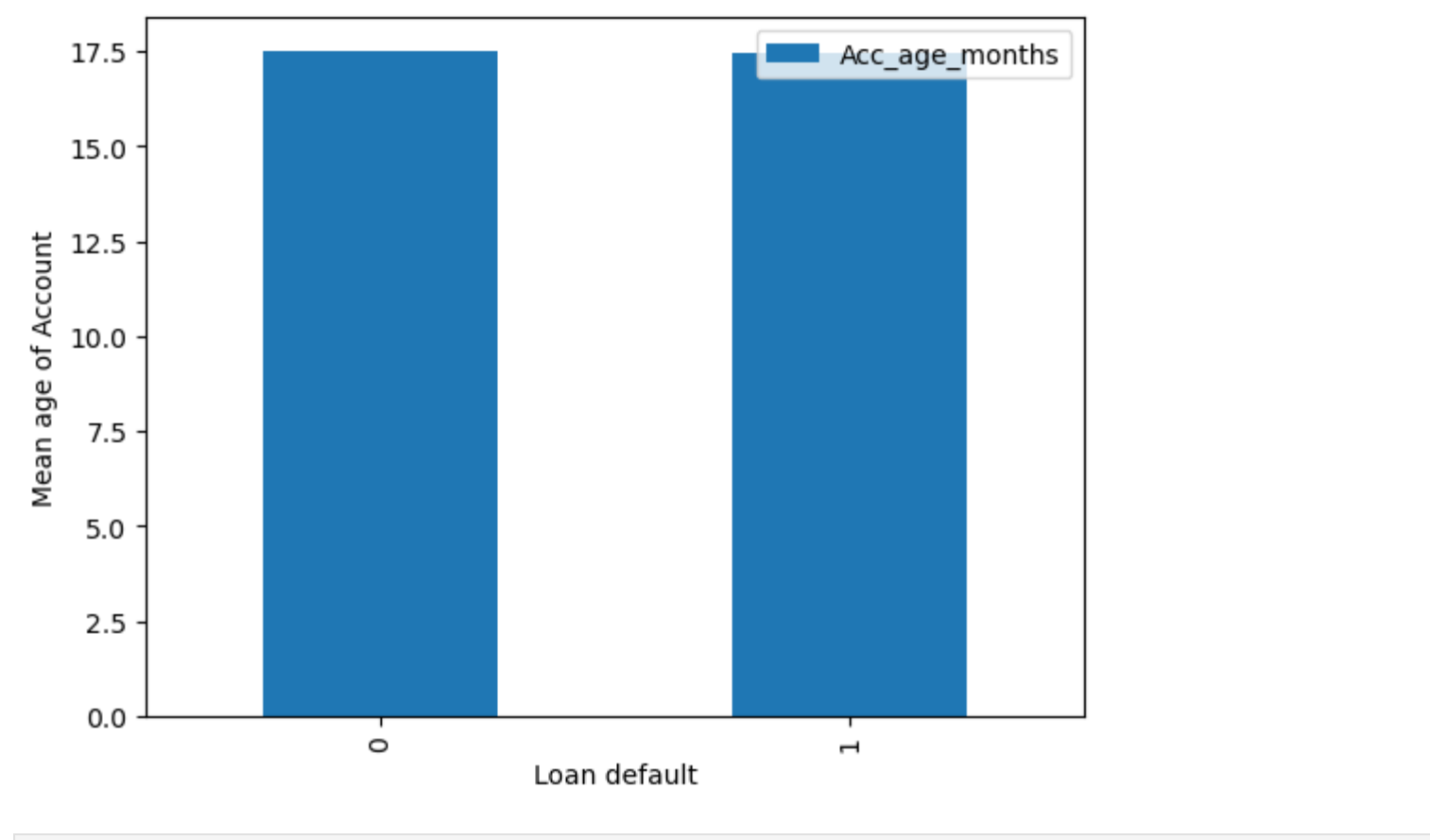



```
In [133…: #checking if account age differs according to the loan default
age=ldi['Acc_age_months']
ld=ldi['loan_default']
mod = ols("age ~ ld", data=ldi).fit()
anov_table = sm.stats.anova_lm(mod)
anov_table
```

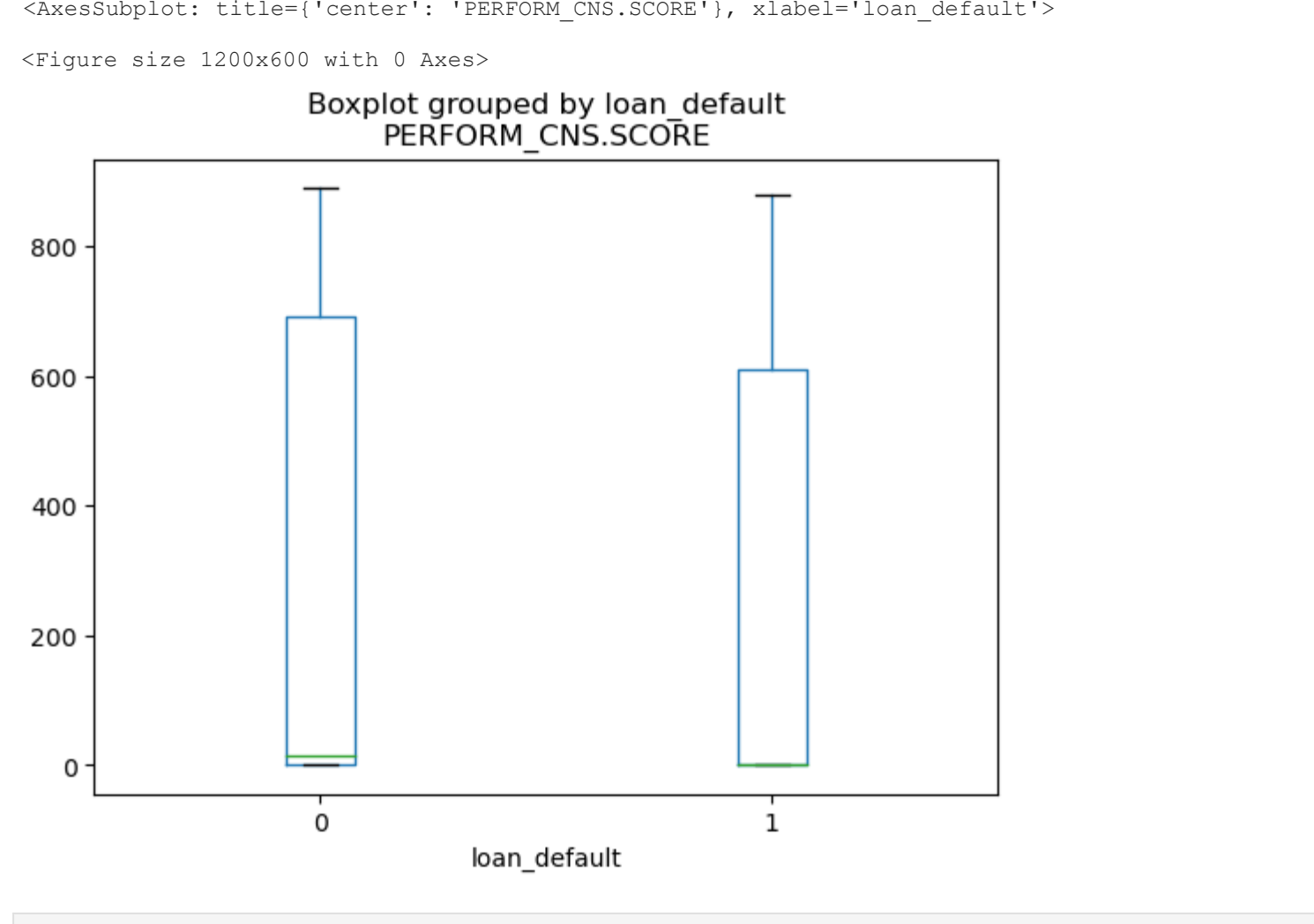
Out[133]:

	df	sum_sq	mean_sq	F	PR(>F)
ld	1.0	9.137824e+01	91.378235	7.822513	0.00516
Residual	233152.0	2.723552e+06	11.681442	NaN	NaN

```
In [134…: #seeing the naverage age of bank accounts according to loan default
grouped = ldi.groupby('loan_default').agg({'Acc_age_months': 'mean'})

grouped.plot(kind='bar')
plt.xlabel('Loan default')
plt.ylabel('Mean age of Account')
plt.show()
```



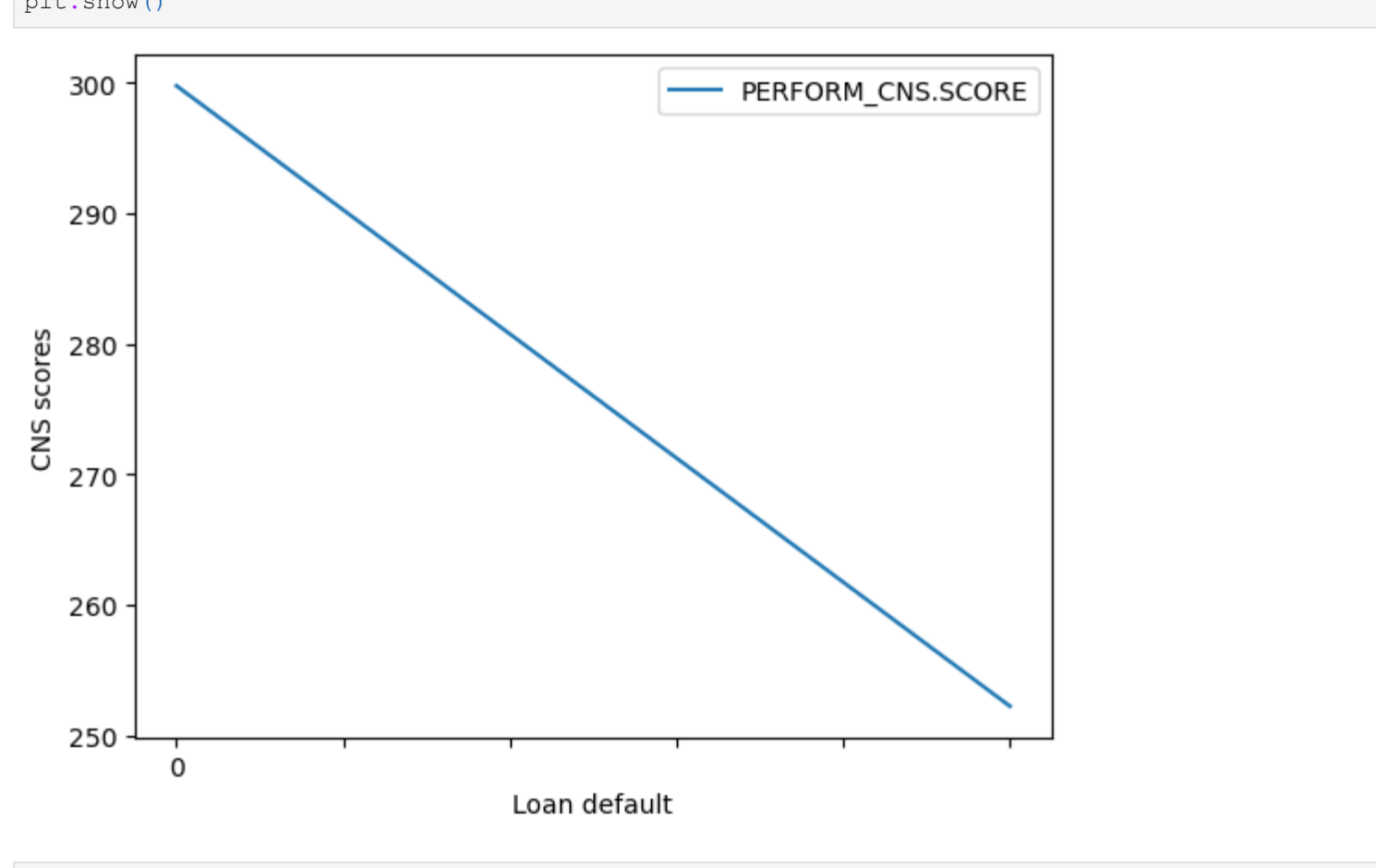
In [135]:
```
#do defaulters have low sns score on average
plt.figure(figsize=[12,6])
ldi.boxplot(by="loan_default",column="PERFORM_CNS.SCORE", grid = False)
```
Out[135]:
```
<AxesSubplot: title={'center': 'PERFORM_CNS.SCORE'}, xlabel='loan_default'>
<Figure size 1200x600 with 0 Axes>
```



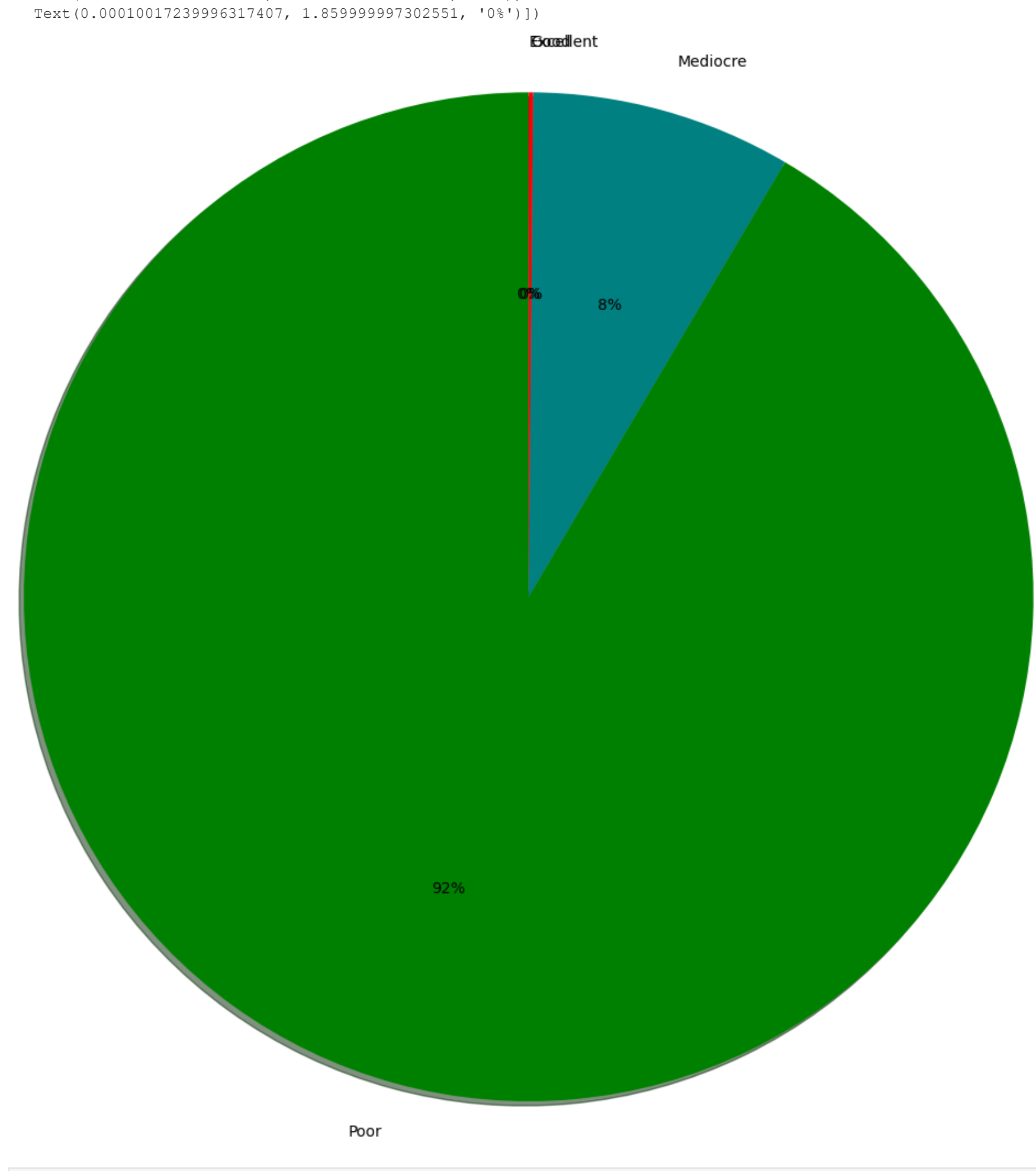
In [136]:
```
#checking if score differs according to the loan default
score=ldi['PERFORM_CNS.SCORE']
ld=ldi['loan_default']
mod = ols("score ~ ld", data=ldi).fit()
anov_table = sm.stats.anova_lm(mod)
anov_table
```
Out[136]:

	df	sum_sq	mean_sq	F	PR(>F)
ld	1.0	8.958388e+07	8.958388e+07	785.040202	1.869421e-172
Residual	233152.0	2.660585e+10	1.141138e+05	NaN	NaN

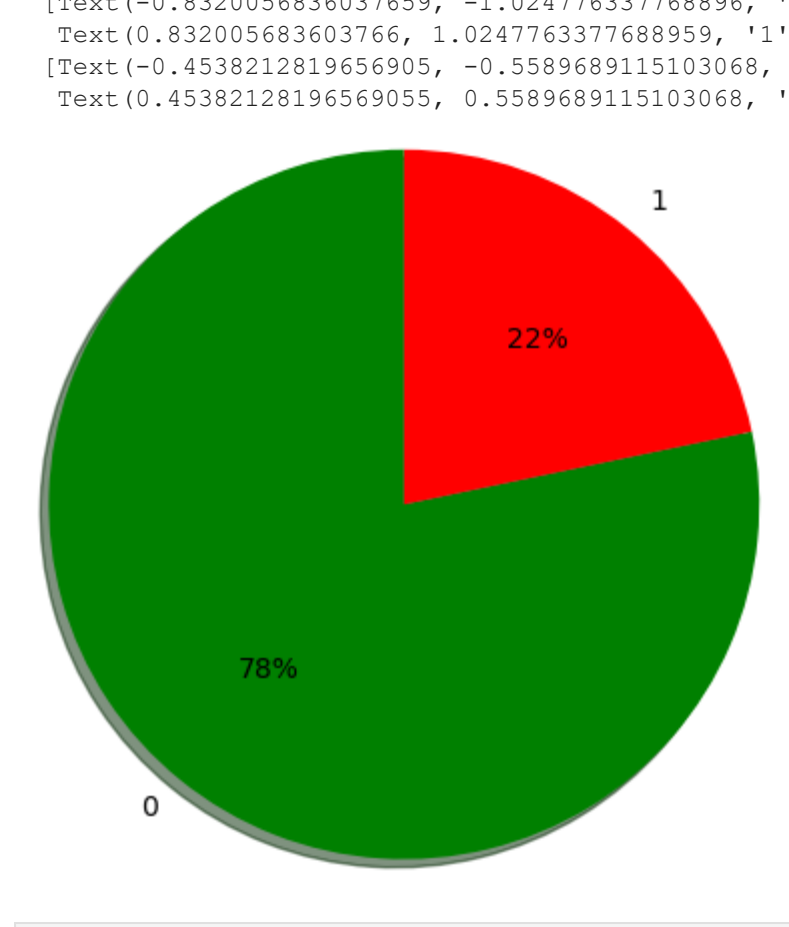
In [137]:
```
#seeing the naverage age of bank accounts according to loan default
grouped = ldi.groupby('loan_default').agg({'PERFORM_CNS.SCORE': 'mean'})

grouped.plot(kind='line')
plt.xlabel('Loan default')
plt.ylabel('CNS scores')
plt.show()
```



In [138]:
```
#What is the profile strength of majority of people


ps=ldi['Profile_strength'].unique()
number=ldi['Profile_strength'].value_counts()


# color for each label
colors = ['green','teal', 'red','blue']

# plotting the pie chart
plt.pie(number, labels =
ps, colors=colors,
        startangle=90, shadow = True,
        radius = 3.1, autopct = '%1.0f%%')
```
Out[138]:
```
([<matplotlib.patches.Wedge at 0x26d9c0f7730>,
  <matplotlib.patches.Wedge at 0x26d9c0ff880>,
  <matplotlib.patches.Wedge at 0x26d9c115af0>,
  <matplotlib.patches.Wedge at 0x26db23f7bb0>],
 [Text(-0.8974538984937792, -3.2897836555126725, 'Poor'),
  Text(0.9121172959636276, 3.2857483224379807, 'Mediocre'),
  Text(0.0153922018262098l, 3.409965260838143, 'Good'),
  Text(0.00018364939993248584, 3.4099999950546773, 'Excellent')],
 [Text(-0.489520308269334, -1.7944274484614573, '92%'),
  Text(0.4975185250710695, 1.7922263576934436, '8%'),
  Text(0.008395746450659894, 1.8599810513662596, '0%'),
  Text(0.00010017239996317407, 1.859999997302551, '0%')])
```



In [139]:
```
#seeing the %age of loan defaults


ld=ldi['loan_default'].unique()
number=ldi['loan_default'].value_counts()


# color for each label
colors = ['green', 'red']

# plotting the pie chart
plt.pie(number, labels =
ld, colors=colors,
        startangle=90, shadow = True,
        radius = 1.2, autopct = '%1.0f%%')
```
Out[139]:
```
([<matplotlib.patches.Wedge at 0x26d9c2809d0>,
  <matplotlib.patches.Wedge at 0x26d9c28c2e0>],
 [Text(-0.8320056836037659, -1.024776337768896, '0'),
  Text(0.832005683603766, 1.0247763377688959, '1')],
 [Text(-0.4538212819656905, -0.5589689115103068, '78%'),
  Text(0.45382128196569055, 0.5589689115103068, '22%')])
```



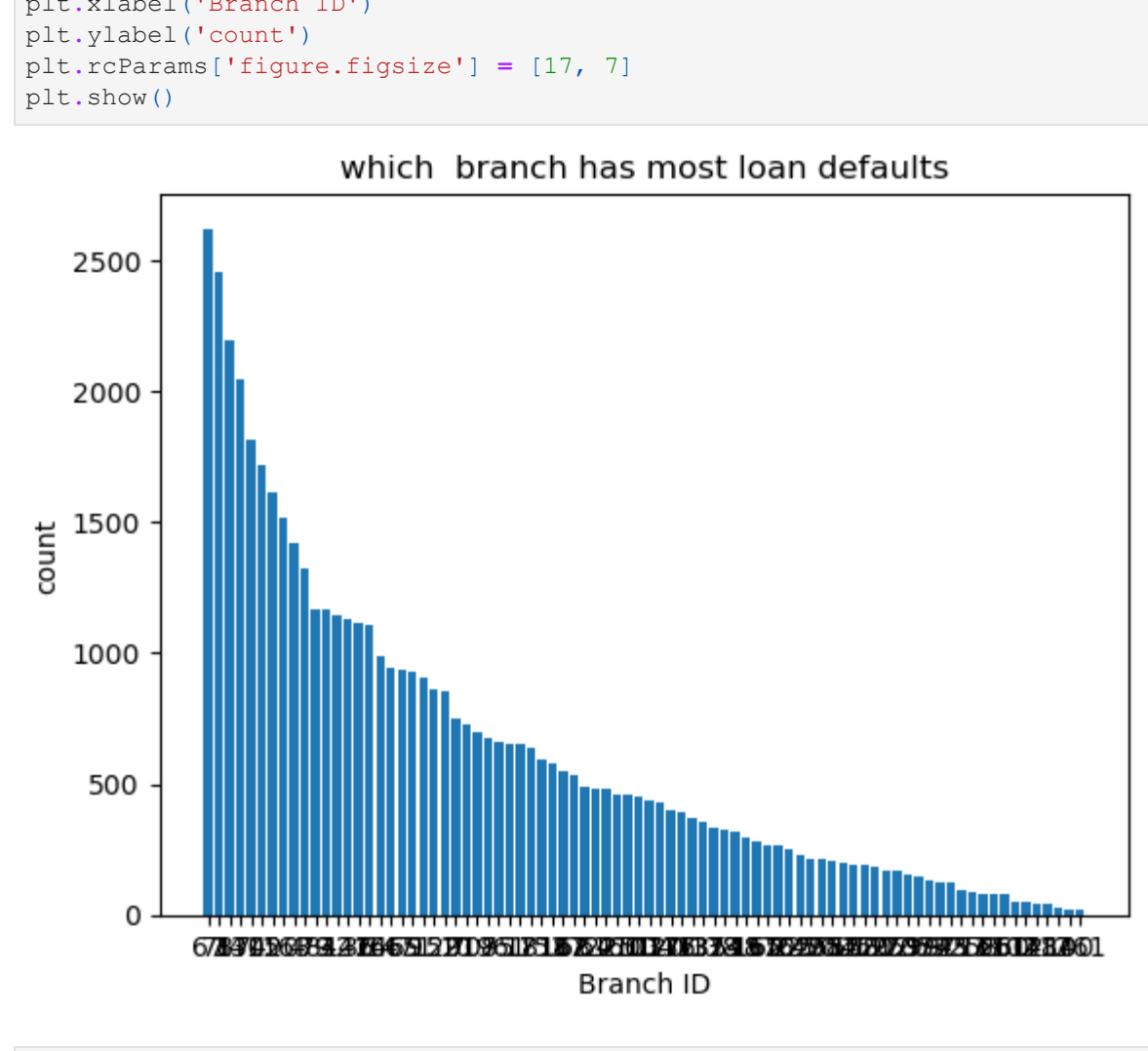
In [140]:
```
#dividing into age group
age_groups=[]
for x in ldi['Age']:
    if x<100 and x>=60:
        age_groups.append('Senior citizen')
    elif x<60 and x>=40:
        age_groups.append('Middle Aged')
    elif x<40 and x>=25:
        age_groups.append('Adult')
    else:
        age_groups.append("Youth")


ldi['age_group']= age_groups
```

In [141]:
```
#taking subset of default
ldef=ldi[ldi.loan_default=='1']
ldef
```
Out[141]:

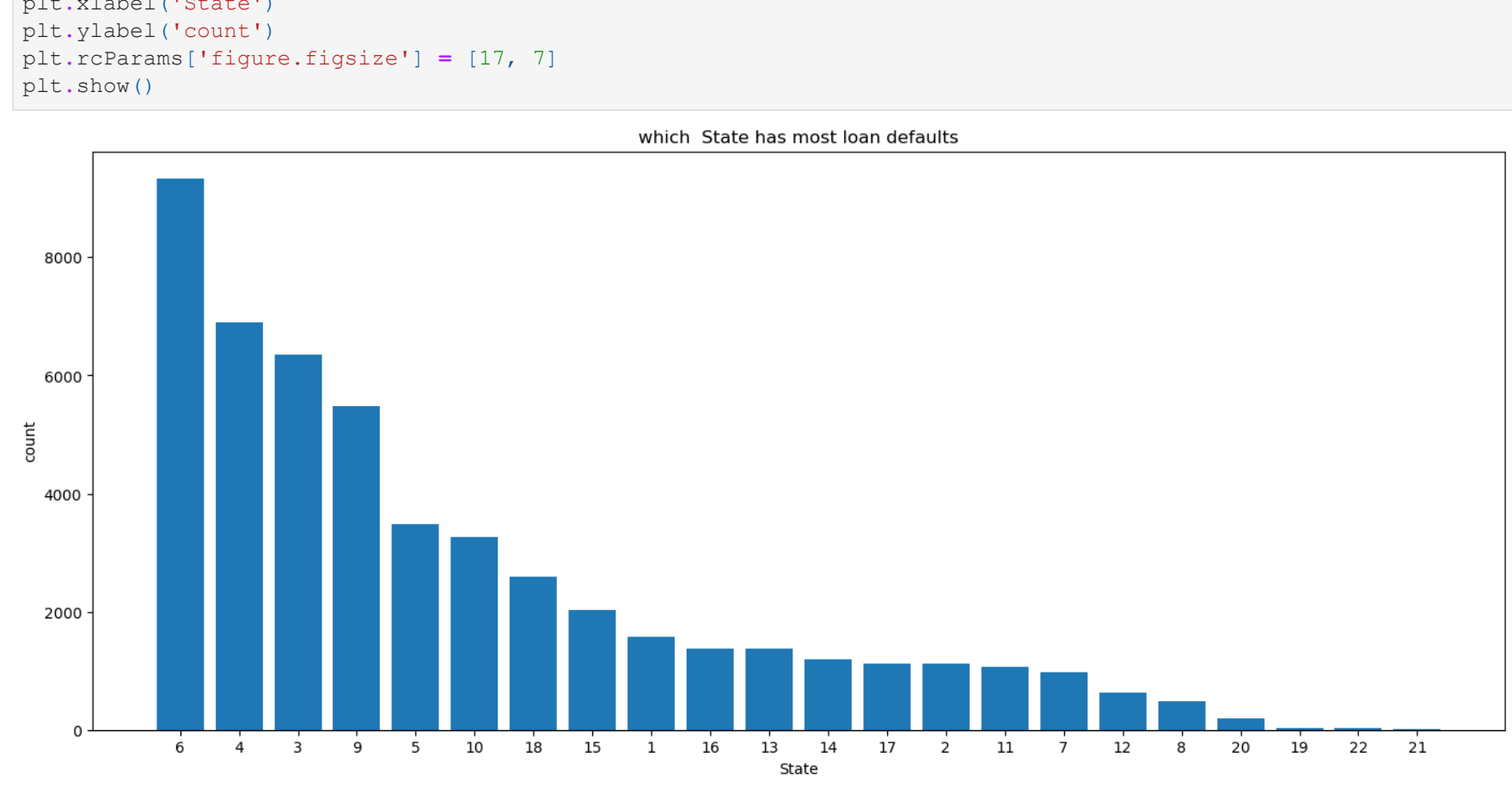
	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Age	Employment.Type	DisbursalDate	State_ID	...
2	52378.0	68763.0	88.39	67	22807	45	45.336986	Self employed	2018-09-26	6	...
7	53713.0	61780.0	89.83	67	22807	45	54.695890	Self employed	2018-08-16	6	...
8	57713.0	68040.0	86.27	67	22807	45	47.276712	Self employed	2018-10-10	6	...
12	50578.0	68763.0	89.55	67	22807	45	26.638356	Salaried	2018-08-06	6	...
18	55513.0	67950.0	83.15	67	22807	45	28.841096	Salaried	2018-10-04	6	...
...	...	...	...	...	...	...	...	...	...	...	...
233125	51303.0	68763.0	79.77	159	24319	48	38.268493	Self employed	2018-10-20	7	...
233127	53803.0	68763.0	79.80	258	24412	86	48.276712	Self employed	2018-10-24	16	...
233134	49763.0	68763.0	79.77	120	22936	86	43.273973	Salaried	2018-10-28	12	...
233138	53303.0	71088.0	77.37	20	14518	45	59.284932	Self employed	2018-10-08	5	...
233147	43272.0	68763.0	74.86	78	17014	45	34.265753	Salaried	2018-08-14	4	...

50611 rows × 41 columns

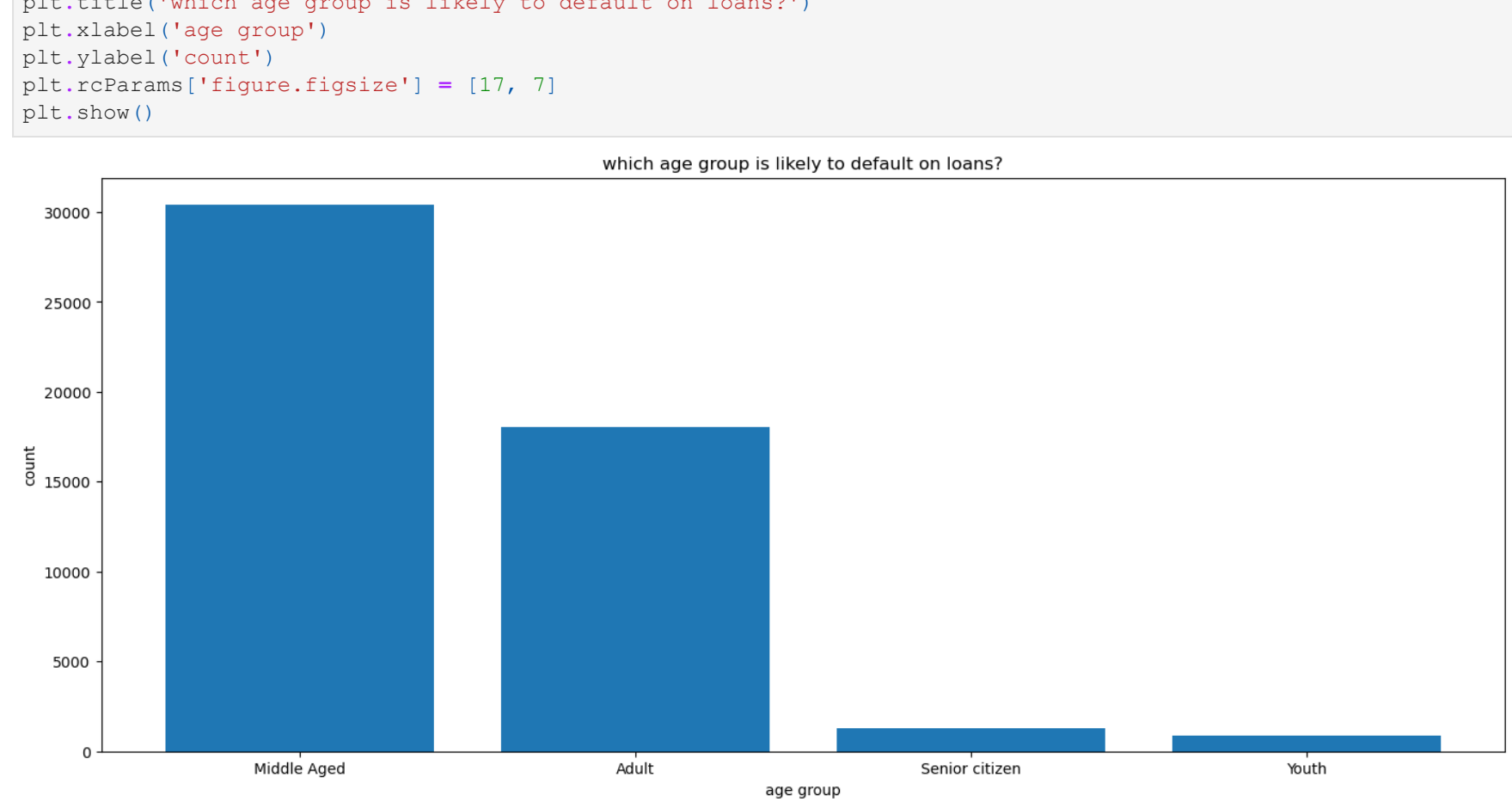
In [142]:
```
#most default branch
branch=ldef['branch_id'].unique()
count=ldef['branch_id'].value_counts()

#plotting survivors per hospital number
plt.bar(branch,count)


plt.title('which  branch has most loan defaults')
plt.xlabel('Branch ID')
plt.ylabel('count')
plt.rcParams['figure.figsize'] = [17, 7]
plt.show()
```



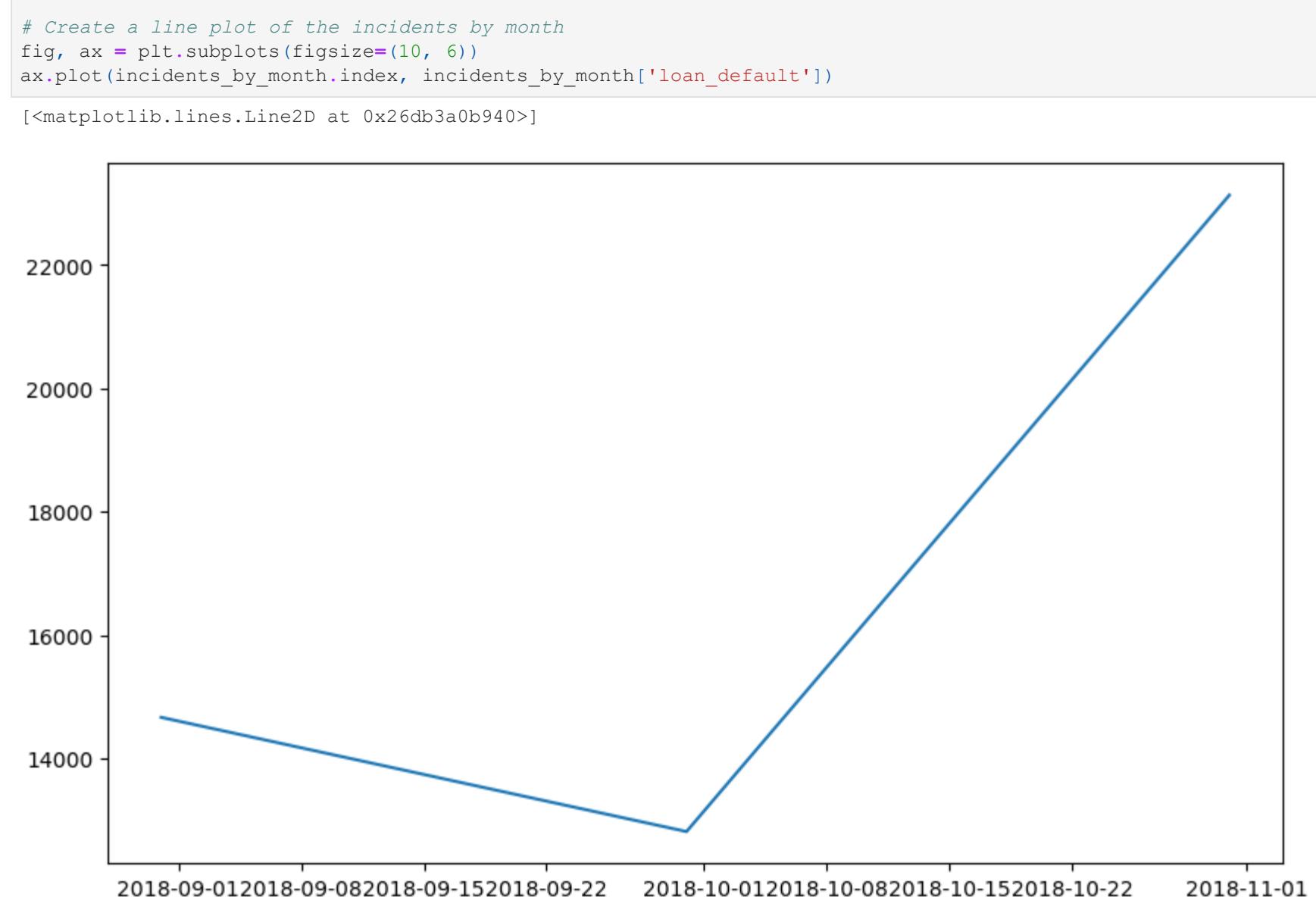
In [143]:
```
#most default state
State=ldef['State_ID'].unique()
count=ldef['State_ID'].value_counts()

#plotting survivors per hospital number
plt.bar(State,count)


plt.title('which  State has most loan defaults')
plt.xlabel('State')
plt.ylabel('count')
plt.rcParams['figure.figsize'] = [17, 7]
plt.show()
```



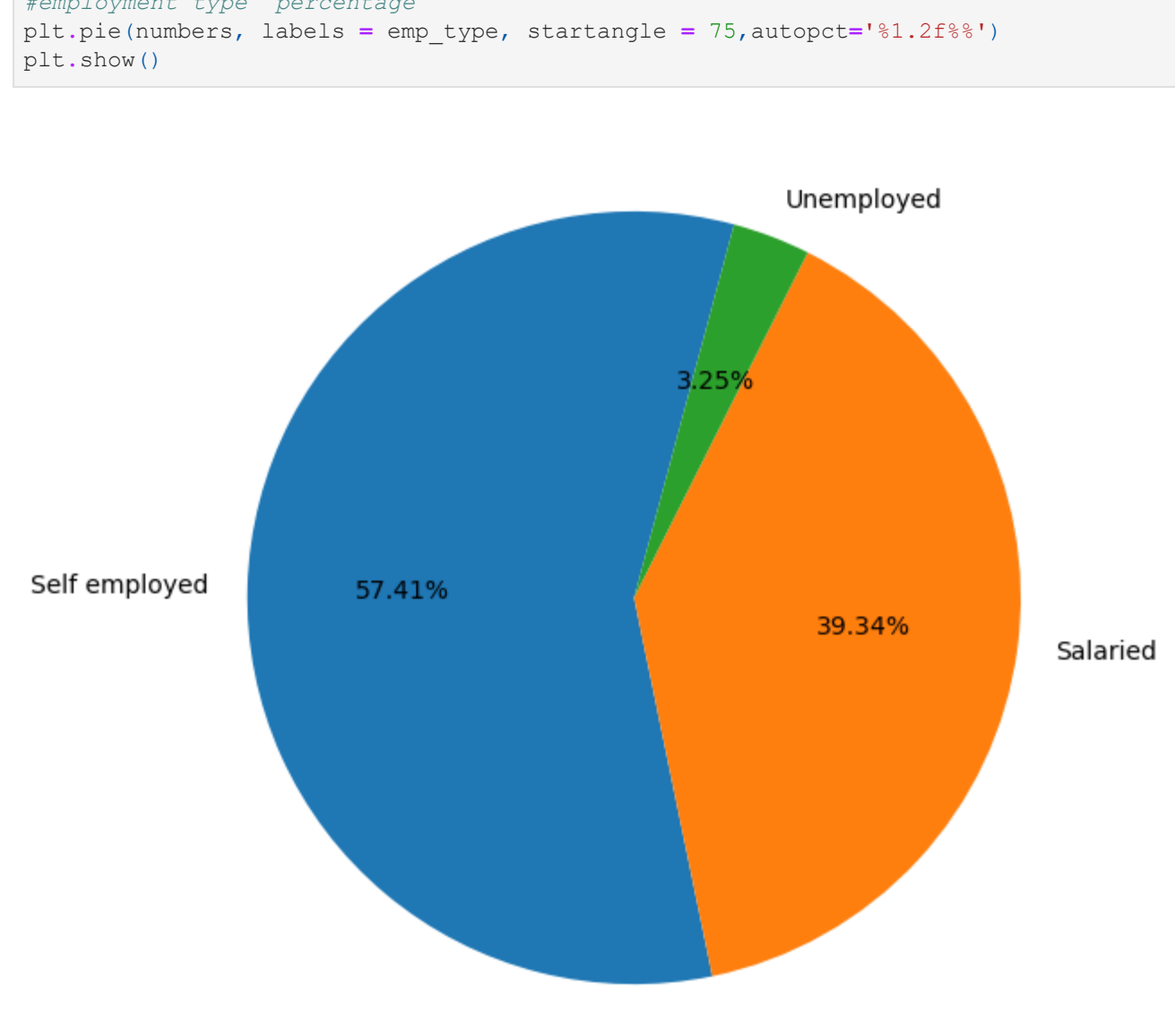
In [144]:
```
#which age group is likely to default on loans?
agegroup=ldef['age_group'].unique()
count=ldef['age_group'].value_counts()

#plotting survivors per hospital number
plt.bar(agegroup,count)


plt.title('which age group is likely to default on loans?')
plt.xlabel('age group')
plt.ylabel('count')
plt.rcParams['figure.figsize'] = [17, 7]
plt.show()
```



In [145]:
```
#which month type has most defaults?



incidents_by_month = ldef.groupby(pd.Grouper(key='DisbursalDate', freq='M')).count()

# Create a line plot of the incidents by month
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(incidents_by_month.index, incidents_by_month['loan_default'])
```
Out[145]:
```
[<matplotlib.lines.Line2D at 0x26db3a0b940>]
```



In [146]:
```
#What employment type are majorly in loan default?
emp_type=ldef['Employment.Type'].unique()
numbers=ldef['Employment.Type'].value_counts()

#employment type  percentage
plt.pie(numbers, labels = emp_type, startangle = 75,autopct='%1.2f%%')
plt.show()
```



In [147]:
```
#Summary of loan default
ldef.describe()
```
Out[147]:

	disbursed_amount	asset_cost	ltv	Age	PERFORM_CNS.SCORE	PRI.NO.OF.ACCTS	PRI.ACTIVE.ACCTS	PRI.OVERDUE.A
count	50611.000000	50611.000000	50611.000000	50611.000000	50611.000000	50611.000000	50611.000000	50611.0
mean	54632.217838	68913.288080	80.369582	38.340156	252.236372	2.089328	0.887060	0.1
std	6111.229891	3117.714297	5.262033	9.688751	318.826242	5.040134	1.669966	0.6
min	39794.000000	61330.000000	68.880000	22.457534	0.000000	0.000000	0.000000	0.0
25%	51303.000000	67997.000000	77.155000	30.263014	0.000000	0.000000	0.000000	0.0
50%	53803.000000	68763.000000	79.770000	36.268493	0.000000	0.000000	0.000000	0.0
75%	58313.000000	69800.000000	84.540000	45.056164	610.000000	2.000000	1.000000	0.0
max	68882.000000	76807.000000	89.950000	68.758904	879.000000	453.000000	35.000000	18.0

8 rows × 25 columns

Observations

1.The loan defaulters have a lower average CNS score.

2.Middle aged people and self employed people are more likely to be lone defaulters.

3.Both groups of people make same amount of enquiries.

4.Loan defaults shoot up in November after a sharp dip in October.

5.Giving loans to states 21 and 22 is a lot safer than to people of state 6 and 4.

6.Every 5th person defaults on their loans.

7.It doesnt matter if the account is old or new they can still default on their loans.

8.Branch no. 67xxx has the most number of loan defaults.

9.There is a lot of difference between the disbursed and sanctioned amount in both primory and secondary.

10.Making more enquiries is a green flag.

11.Around 92% of the people have a poor profile strength and incomplete documentation.

12.Mobile is the most used verification mode.

13.The amount disbursed has a moderate positive correlation with ltv ratio and asset value.

Step 4: Model Building

Now we are going to build the Decision Tree classifier to predict who is more likely to default on a loan.

In [148]:
```
#creating target variable which is the outcome of the loan default
target=ldi['loan_default']
```

In [149]:
```
#removing target variable from main dataframe
ldi= ldi.drop(['loan_default'], axis=1)
```

In [150]:
```
#listing categorical variables
obj_vars=['Employment.Type','age_group','branch_id','supplier_id','manufacturer_id','PERFORM_CNS.SCORE.DESCRIPT
          'MobileNo_Avl_Flag','Aadhar_flag','PAN_flag','VoterID_flag','Driving_flag','Passport_flag','Profile_s
```

In [151]:
```
#integrating them
ldi[obj_vars]=['Employment.Type','age_group','branch_id','supplier_id','manufacturer_id','PERFORM_CNS.SCORE.DES
          'MobileNo_Avl_Flag','Aadhar_flag','PAN_flag','VoterID_flag','Driving_flag','Passport_flag','Profile_s
```

In [152]:
```
#creating dummy variables


for category in ldi[obj_vars]:
    ldi[category] = pd.get_dummies(ldi[category])
```

In [153]:
```
#checking missing values

ldi.isnull().sum()
```
Out[153]:
```
disbursed_amount                       0
asset_cost                             0
ltv                                    0
branch_id                              0
supplier_id                            0
manufacturer_id                        0
Age                                    0
Employment.Type                        0
DisbursalDate                          0
State_ID                               0
MobileNo_Avl_Flag                      0
Aadhar_flag                            0
PAN_flag                               0
VoterID_flag                           0
Driving_flag                           0
Passport_flag                          0
Profile_strength                       0
PERFORM_CNS.SCORE                      0
PERFORM_CNS.SCORE.DESCRIPTION          0
PRI.NO.OF.ACCTS                        0
PRI.ACTIVE.ACCTS                       0
PRI.OVERDUE.ACCTS                      0
PRI.CURRENT.BALANCE                    0
PRI.SANCTIONED.AMOUNT                  0
PRI.DISBURSED.AMOUNT                   0
SEC.NO.OF.ACCTS                        0
SEC.ACTIVE.ACCTS                       0
SEC.OVERDUE.ACCTS                      0
SEC.CURRENT.BALANCE                    0
SEC.SANCTIONED.AMOUNT                  0
SEC.DISBURSED.AMOUNT                   0
PRIMARY.INSTAL.AMT                     0
SEC.INSTAL.AMT                         0
NEW.ACCTS.IN.LAST.SIX.MONTHS           0
DELINQUENT.ACCTS.IN.LAST.SIX.MONTHS    0
Acc_age_months                         0
NO.OF_INQUIRIES                        0
pri_diff                               0
sec_diff                               0
age_group                              0
dtype: int64
```

In [154]:
```
#checking the new dummy variables
ldi.head()
```
Out[154]:

	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Age	Employment.Type	DisbursalDate	State_ID	...	SEC.
0	50578.0	68763.0	89.55	1	1	1	39.271233	1	2018-08-03	1	...	
1	53278.0	61360.0	89.63	1	1	1	37.624658	1	2018-08-01	1	...	
2	52378.0	68763.0	88.39	1	1	1	45.336986	1	2018-09-26	1	...	
3	46349.0	61500.0	76.42	1	1	1	34.852055	1	2018-09-23	1	...	
4	43594.0	68763.0	79.77	1	1	1	28.731507	1	2018-10-08	1	...	

5 rows × 40 columns

In [155]:
```
#checking shape of transformed dataset
ldi.shape
```
Out[155]:
```
(233154, 40)
```

In [156]:
```
#getting classifier

dct=DecisionTreeClassifier()

dct
```
Out[156]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [157]:
```
#initializing label encoder
le=LabelEncoder()
le
```
Out[157]:
```
▾ LabelEncoder
LabelEncoder()
```

In [158]:
```
#standard scaler initiated
sc=StandardScaler()
```

In [159]:
```
#transforming target variable
target=le.fit_transform(target)
target
```
Out[159]:
```
array([0, 0, 1, ..., 0, 0, 0])
```

In [160]:
```
#dropping useless columns again
ldi=ldi.drop(['DisbursalDate'],axis=1)
ldi.head()
```
Out[160]:

	disbursed_amount	asset_cost	ltv	branch_id	supplier_id	manufacturer_id	Age	Employment.Type	State_ID	MobileNo_Avl_Flag	...
0	50578.0	68763.0	89.55	1	1	1	39.271233	1	1	1	...
1	53278.0	61360.0	89.63	1	1	1	37.624658	1	1	1	...
2	52378.0	68763.0	88.39	1	1	1	45.336986	1	1	1	...
3	46349.0	61500.0	76.42	1	1	1	34.852055	1	1	1	...
4	43594.0	68763.0	79.77	1	1	1	28.731507	1	1	1	...

5 rows × 39 columns

In [161]:
```
#converting to Dataframe
ldi= pd.DataFrame(ldi)
target= pd.DataFrame(target)
```

In [162]:
```
#creating X and y
X = ldi.values
y= target.values
```

In [163]:
```
#train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.25, random_state=1)
```

In [164]:
```
#using standard scaler
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [165]:
```
#fitting the model
dct.fit(X_train,y_train)
```
Out[165]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [166]:
```
#predict
ypred= dct.predict(X_test)
ypred
```
Out[166]:
```
array([0, 0, 0, ..., 0, 1, 0])
```

In [167]:
```
#accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred)*100))
```
```
Accuracy of the model: 66.65923244522988%
```

In [168]:
```
#confusion matrix
cm=confusion_matrix(y_test,ypred)
print(cm)
```
```
[[35597  9974]
 [ 9460  3258]]
```

In [169]:
```
#printing metrics
print(classification_report(y_test,ypred))
```
```
              precision    recall  f1-score   support

           0       0.79      0.78      0.79     45571
           1       0.25      0.26      0.25     12718

    accuracy                           0.67     58289
   macro avg       0.52      0.52      0.52     58289
weighted avg       0.67      0.67      0.67     58289
```

In [170]:
```
#using random forest
rf=RandomForestClassifier()
```

In [171]:
```
#fitting the model
rf.fit(X_train,y_train)
```
```
C:\Users\sujoydutta\AppData\Local\Temp\ipykernel_9184\2316523344.py:2: 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
().
  rf.fit(X_train,y_train)
```
Out[171]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [172]:
```
#predict
ypred= rf.predict(X_test)
ypred
```
Out[172]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [173]:
```
#accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred)*100))
```
```
Accuracy of the model: 76.54102832438367%
```

In [174]:
```
#confusion matrix
cm=confusion_matrix(y_test,ypred)
print(cm)
```
```
[[43993  1578]
 [12096   622]]
```

In [175]:
```
#printing metrics
print(classification_report(y_test,ypred))
```
```
              precision    recall  f1-score   support

           0       0.78      0.97      0.87     45571
           1       0.28      0.05      0.08     12718

    accuracy                           0.77     58289
   macro avg       0.53      0.51      0.47     58289
weighted avg       0.67      0.77      0.69     58289
```

In [176]:
```
#using logistic regression
lgr=LogisticRegression()
```

In [177]:
```
#fitting the model
lgr.fit(X_train,y_train)
```
```
C:\Users\sujoydutta\anaconda3\lib\site-packages\sklearn\utils\validation.py:1111: DataConversionWarning: A colu
mn-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)
```
Out[177]:
```
▾ LogisticRegression
LogisticRegression()
```

In [179]:
```
#predict
ypred= lgr.predict(X_test)
ypred
```
Out[179]:
```
array([0, 0, 0, ..., 0, 0, 0])
```

In [180]:
```
#accuracy
print("Accuracy of the model: {0}%".format(accuracy_score(y_test,ypred)*100))
```
```
Accuracy of the model: 78.16054487124501%
```

In [181]:
```
#confusion matrix
cm=confusion_matrix(y_test,ypred)
print(cm)
```
```
[[45522    49]
 [12681    37]]
```

In [182]:
```
#printing metrics
print(classification_report(y_test,ypred))
```
```
              precision    recall  f1-score   support

           0       0.78      1.00      0.88     45571
           1       0.43      0.00      0.01     12718

    accuracy                           0.78     58289
   macro avg       0.61      0.50      0.44     58289
weighted avg       0.71      0.78      0.69     58289
```

Conclusion

We have used three models to classify the people into loan defaulter or not. The third model has been found to be the most accurate(almost 80%) and hence this is the best model to classify people into defaulters and non defaulters.