CASE STUDY – PHASE 2

Name	Andrew ID
Bharat Madan	bmadan
Dhwani Panjwani	dpanjwan
Sahil Ahuja	sahilahu
Sujai Chaudhary	sujaic

Phase 1 - Ingestion and Cleaning

In the Phase 2 of the Case Study, we will carry out the following steps:

- Ingest raw downloaded data
- Output a combined dataset ready for analysis and modeling

In [1]:
```
import pandas as pd
import os
from sys import platform
import matplotlib.pyplot as plt
import datetime
import numpy as np
import pickle
import seaborn
```

In [2]:
```
# A helper function that you'll be using while reading the raw files
def is_integer(x):
    '''
    This function returns True if x is an integer, and False otherwise
    '''
    try:
        return (int(x) == float(x))
    except:
        return False
```

Parameters

In [3]:
```
# Define the directories that contain the files downloaded
dir_cs = '2003_download'
# path to the directory where all the *.csv.zip files are located

# Define the output path for the pickle
pickle_file = "2003_download\\" + "clean_data.pickle"
# path to save cleaned data
```

In [4]:
```
# Identify the columns we'll be keeping from the dataset
cols_to_pick = ['id', 'loan_amnt','funded_amnt', 'term', 'int_rate', 'grade', 'emp_length', 'home_ownership',
                'annual_inc', 'verification_status','issue_d', 'loan_status', 'purpose', 'dti',
                'delinq_2yrs', 'earliest_cr_line', 'open_acc', 'pub_rec', 'fico_range_high',
                'fico_range_low', 'revol_bal', 'revol_util', 'total_pymnt', 'recoveries','last_pymnt_d']
# list of features to use for this study as indicated in the handout

# Identify the type of each of these column based on your CS-Phase 1 response
float_cols = ['loan_amnt', 'funded_amnt','annual_inc',  'dti', 'open_acc','revol_bal' ,  'total_pymnt', 'recoveries' ,'
cat_cols = [ 'term', 'grade', 'emp_length','home_ownership' , 'verification_status', 'loan_status', 'purpose']
# categorical features
perc_cols = ['int_rate', 'revol_util']
date_cols = ['issue_d', 'earliest_cr_line', 'last_pymnt_d']

# Ensure that we have types for every column
assert set(cols_to_pick) - set(float_cols) - set(cat_cols) - set(perc_cols) - set(date_cols) == set(["id"])
```

In [5]:
```
# Some of the columns selected will not be used directly in the model,
# but will be used to generate other features.
#
# Create variables specifying the features that will be used

# All categorical columns other than "loan_status" will be used as
# discrete features

discrete_features = list(set(cat_cols) - set(["loan_status"]))

# All numeric columns will be used as continuous features
continuous_features = list(float_cols + perc_cols)
```

Ingestion

Ingest the data files from both sets, perform consistency checks, and prepare one single file for each set

In [6]:
```
def ingest_files(directory):
    '''
    This function will ingest every file in the specified directory
    into a pandas dataframe. It will return a dictionary containing
    these dataframes, keyed by the file name.

    We assume the directory contains files directly downloaded from
    the link given in the handout, and *only* those files. Thus, we
```

```
    assume the files are zipped (pd.read_csv can read zipped files)
    and we assume the first line in each file needs to be skipped.

    Note that each file will be read *without* formatting
    '''

    # If the directory has no trailing slash, add one

    if directory[-1] != "/":
        directory+= "/"


    all_files =  os.listdir(directory)
    # get list of all files from the directory
    output = {}

    print("Directory " + directory + " has " + str(len(all_files)) + " files:")
    for i in all_files:
        #i = i.replace(".zip", "")
        print("    Reading file " + i)

        output[i] = pd.read_csv(directory+i,dtype='str', skiprows =1, compression='zip' )
        # read each with dtype='str' and skip_rows =1

        # Some of the files have "summary" lines that, for example
        # read "Total number of loans number in Policy 1: ....."
        # To remove those lines, find any lines with non-integer IDs
        # and remove them
        invalid_rows = output[i]['id'].apply(lambda x:is_integer(x))
        # mask rows that have non-integer IDs. Use is_integer method
        if invalid_rows.sum() > 0:
            print("Found " + str(invalid_rows.sum()) + " invalid rows which were removed")
#             print(invalid_rows)
            output[i] = output[i][invalid_rows]
            #remove invalid rows

    return output # return dictionary of dataframe
```

In [7]:

```
# Ingest the set of files we downloaded using the defined method "ingest_files"
files_cs = ingest_files(dir_cs) # dictioary of (filename, dataframe) as (key, value)
```

```
Directory 2003_download/ has 20 files:
    Reading file LoanStats3a_securev1.csv.zip
Found 42535 invalid rows which were removed
    Reading file LoanStats3b_securev1.csv.zip
Found 188181 invalid rows which were removed
    Reading file LoanStats3c_securev1.csv.zip
Found 235629 invalid rows which were removed
    Reading file LoanStats3d_securev1.csv.zip
Found 421095 invalid rows which were removed
    Reading file LoanStats_securev1_2016Q1.csv.zip
Found 133887 invalid rows which were removed
    Reading file LoanStats_securev1_2016Q2.csv.zip
Found 97854 invalid rows which were removed
    Reading file LoanStats_securev1_2016Q3.csv.zip
Found 99120 invalid rows which were removed
    Reading file LoanStats_securev1_2016Q4.csv.zip
Found 103546 invalid rows which were removed
    Reading file LoanStats_securev1_2017Q1.csv.zip
Found 96779 invalid rows which were removed
    Reading file LoanStats_securev1_2017Q2.csv.zip
Found 105451 invalid rows which were removed
    Reading file LoanStats_securev1_2017Q3.csv.zip
Found 122701 invalid rows which were removed
    Reading file LoanStats_securev1_2017Q4.csv.zip
Found 118648 invalid rows which were removed
    Reading file LoanStats_securev1_2018Q1.csv.zip
Found 107759 invalid rows which were removed
    Reading file LoanStats_securev1_2018Q2.csv.zip
Found 130633 invalid rows which were removed
    Reading file LoanStats_securev1_2018Q3.csv.zip
Found 128059 invalid rows which were removed
    Reading file LoanStats_securev1_2018Q4.csv.zip
Found 128284 invalid rows which were removed
    Reading file LoanStats_securev1_2019Q1.csv.zip
Found 115573 invalid rows which were removed
    Reading file LoanStats_securev1_2019Q2.csv.zip
Found 131004 invalid rows which were removed
    Reading file LoanStats_securev1_2019Q3.csv.zip
Found 142901 invalid rows which were removed
    Reading file LoanStats_securev1_2019Q4.csv.zip
Found 128137 invalid rows which were removed
```

Combine the files

In [8]:
```
files_cs.keys()
```

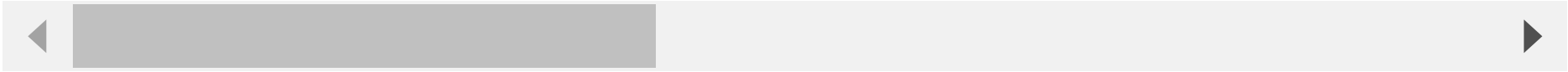
Out[8]:
```
dict_keys(['LoanStats3a_securev1.csv.zip', 'LoanStats3b_securev1.csv.zip', 'LoanStats3c_securev1.csv.zip', 'LoanStats3d
_securev1.csv.zip', 'LoanStats_securev1_2016Q1.csv.zip', 'LoanStats_securev1_2016Q2.csv.zip', 'LoanStats_securev1_2016Q
3.csv.zip', 'LoanStats_securev1_2016Q4.csv.zip', 'LoanStats_securev1_2017Q1.csv.zip', 'LoanStats_securev1_2017Q2.csv.zi
p', 'LoanStats_securev1_2017Q3.csv.zip', 'LoanStats_securev1_2017Q4.csv.zip', 'LoanStats_securev1_2018Q1.csv.zip', 'Loa
nStats_securev1_2018Q2.csv.zip', 'LoanStats_securev1_2018Q3.csv.zip', 'LoanStats_securev1_2018Q4.csv.zip', 'LoanStats_s
ecurev1_2019Q1.csv.zip', 'LoanStats_securev1_2019Q2.csv.zip', 'LoanStats_securev1_2019Q3.csv.zip', 'LoanStats_securev1_
2019Q4.csv.zip'])
```

In [9]:
```
# combine "files_cs" into a pandas dataframe
data_cs = pd.concat(files_cs.values())
# resent index with drop = True
data_cs.reset_index(drop = True,)
```

Out[9]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	...	hardship_p
0	1077501	NaN	5000	5000	4975	36 months	10.65%	162.87	B	B2	...	
1	1077430	NaN	2500	2500	2500	60 months	15.27%	59.83	C	C4	...	
2	1077175	NaN	2400	2400	2400	36 months	15.96%	84.33	C	C5	...	
3	1076863	NaN	10000	10000	10000	36 months	13.49%	339.31	C	C1	...	
4	1075358	NaN	3000	3000	3000	60 months	12.69%	67.79	B	B5	...	
...	...	...	...	...	...	...	...	...	...	...	...	
2777771	158872331	NaN	3000	3000	3000	36 months	17.74%	108.07	C	C5	...	
2777772	158833440	NaN	10000	10000	10000	36 months	6.46%	306.31	A	A1	...	
2777773	158748525	NaN	19000	19000	19000	36 months	6.46%	581.99	A	A1	...	
2777774	158298751	NaN	10000	10000	10000	60 months	28.80%	316.21	D	D5	...	
2777775	158206429	NaN	14875	14875	14875	36 months	16.95%	529.97	C	C4	...	

2777776 rows × 151 columns



Prepare Final Dataset

In [10]:
```
# Keep only the columns of interest from 'data_cs'
final_data = data_cs[cols_to_pick]
```

In [11]:
```
print("Starting with " + str(len(final_data)) + " rows")
```

```
Starting with 2777776 rows
```

Typecast the columns

In [12]:
```
# Remember that we read the data as string (without any formatting).
# Now we would typecast the columns based on feature types which you found out in CS Phase 1

for i in float_cols:
    final_data[i] = final_data[i].astype("float") # typecast float columns
```

```
C:\Users\panjw\AppData\Local\Temp\ipykernel_27708\2410077581.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  final_data[i] = final_data[i].astype("float") # typecast float columns
```

In [13]:
```
def clean_perc(x):
    if pd.isnull(x):
        return np.nan
    else:
        return float(x.strip()[:-1])

for i in perc_cols:
    final_data[i] = final_data[i].apply(lambda x : clean_perc(x))# apply clean_perc to percentage columns
```

```
C:\Users\panjw\AppData\Local\Temp\ipykernel_27708\4107308680.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  final_data[i] = final_data[i].apply(lambda x : clean_perc(x))# apply clean_perc to percentage columns
```

In [14]:
```
def clean_date(x):
    if pd.isnull(x):
        return None
    else:
        return datetime.datetime.strptime( x, "%b-%Y").date()

for i in date_cols:
    final_data[i] = final_data[i].apply(lambda x : clean_date(x))
    # typecast date cloumns to datatime using clean_date
```

```
C:\Users\panjw\AppData\Local\Temp\ipykernel_27708\2587472064.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  final_data[i] = final_data[i].apply(lambda x : clean_date(x))
```

In [15]:
```
for i in cat_cols:
    final_data[i] = final_data[i].apply(lambda x : None if pd.isnull(x) else x)
# for categorical features if the value is null/empty set it to None
```

```
C:\Users\panjw\AppData\Local\Temp\ipykernel_27708\2310750127.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  final_data[i] = final_data[i].apply(lambda x : None if pd.isnull(x) else x)
```

Calculate returns for each loan

In [16]:
```
# Define the names of the four returns we'll be calculating as described in Q.6
# ret_PESS: Pessimistic return
# ret_OPT: Optimistic return
# ret_INTa, ret_INTb: Method3 at two differnt values of "i"
ret_cols = ["ret_PESS", "ret_OPT", "ret_INTa", "ret_INTb"]
```

In [17]:
```
# Remove all rows for loans that were paid back on the days they were issued
final_data['loan_length'] = (final_data.last_pymnt_d - final_data.issue_d) / np.timedelta64(1, 'M')
n_rows = len(final_data)

final_data = final_data[final_data['loan_length']!=0] # select rows where loan_length is not 0.

print("Removed " + str(n_rows - len(final_data)) + " rows")
```

```
C:\Users\panjw\AppData\Local\Temp\ipykernel_27708\1867855074.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  final_data['loan_length'] = (final_data.last_pymnt_d - final_data.issue_d) / np.timedelta64(1, 'M')
```

```
Removed 12361 rows
```

In [18]:
```
final_data.reset_index(drop=True,inplace=True)
```

M1-Pessimistic Method

In [19]:
```
# Calculate the return using a simple annualized profit margin
# Pessimistic definition (Handout 6a.) (M1)

final_data['term_num'] = final_data.term.str.extract('(\d+)',expand=False).astype(int) # length of loan in months

final_data['ret_PESS'] = (final_data["total_pymnt"] -final_data["funded_amnt"])/ final_data["funded_amnt"] +\
                           12/final_data['term_num']
```

M2-Optimistic Method

In [20]:
```
# Assuming that if a loan gives a positive return, we can
# immediately find a similar loan to invest in; if the loan
# takes a loss, we use M1-pessimistic to compute the return

final_data['ret_OPT'] = (final_data["total_pymnt"] -final_data["funded_amnt"])/ final_data["funded_amnt"] +\
                           12/final_data['loan_length']
final_data.loc[final_data.ret_OPT < 0,'ret_OPT'] = final_data['ret_PESS']
```

Method 3

In [21]:
```
def ret_method_3(T, i):
    '''
    Given an investment time horizon (in months) and re-investment
    interest rate, calculate the return of each loan
    '''

    # Assuming that the total amount paid back was paid at equal
    # intervals during the duration of the loan, calculate the
    # size of each of these installment
    actual_installment = (final_data.total_pymnt - final_data.recoveries) / final_data.loan_length

    # Assuming the amount is immediately re-invested at the prime
    # rate, find the total amount of money we'll have by the end
    # of the loan
    cash_by_end_of_loan = actual_installment * ((1 - np.power(1 + i, final_data.loan_length)) / (1 - (1 + i)) ) # compu
```

```
    cash_by_end_of_loan = cash_by_end_of_loan + final_data.recoveries

    # Assuming that cash is then re-invested at the prime rate,
    # with monthly re-investment, until T months from the start
    # of the loan
    remaining_months = T - final_data['loan_length']
    final_return = (cash_by_end_of_loan * (np.power(1 + i, remaining_months)))-  final_data.funded_amnt

    # Find the percentage return
    ret_val = (12/T) * (final_return/final_data.funded_amnt)
    return ret_val
```

In [22]:
```
final_data['ret_INTa'] = ret_method_3(60,0.023) # call ret_method_3 with T=60, i=0.023
final_data['ret_INTb'] = ret_method_3(60,0.04) # call ret_method_3 with T=60, i=0.04
```

Visualize the variables

In [23]:
```
def visualize_float_columns():
    '''
    This function visualizes Box-and-whisker plots for continuous variables
    '''


    # FLoat columns
    for i in float_cols + perc_cols + ret_cols:
        seaborn.boxplot(final_data[i])

        # Print the three highest values
        highest_vals = list(final_data[i].nlargest(3)) # get 3 highest values

        smallest_val = min(final_data[i])

        plt.text(smallest_val, -0.3, highest_vals[0])
        plt.text(smallest_val, -0.2, highest_vals[1])
        plt.text(smallest_val, -0.1, highest_vals[2])

        plt.show()
```

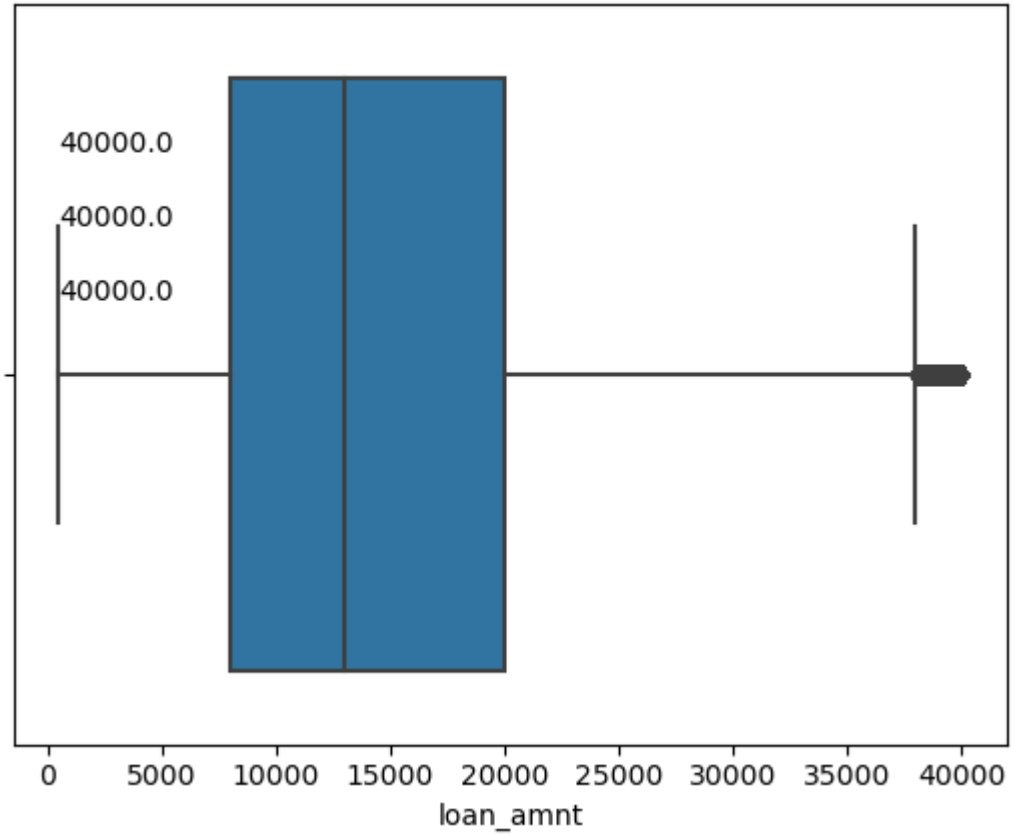
In [24]:
```
def visualize_cat_columns():
    '''
    Lists the distinct values for categorical columns
    '''
    # Categorical columns
```

```
    for i in cat_cols:
        print("Column name:",i) # print field name
        print("Number of distinct values:",final_data[i].nunique()) # print number of distinct values
        print(final_data[i].value_counts()) # for each distinct value print the number of occurances
        print("")
        print("")
```

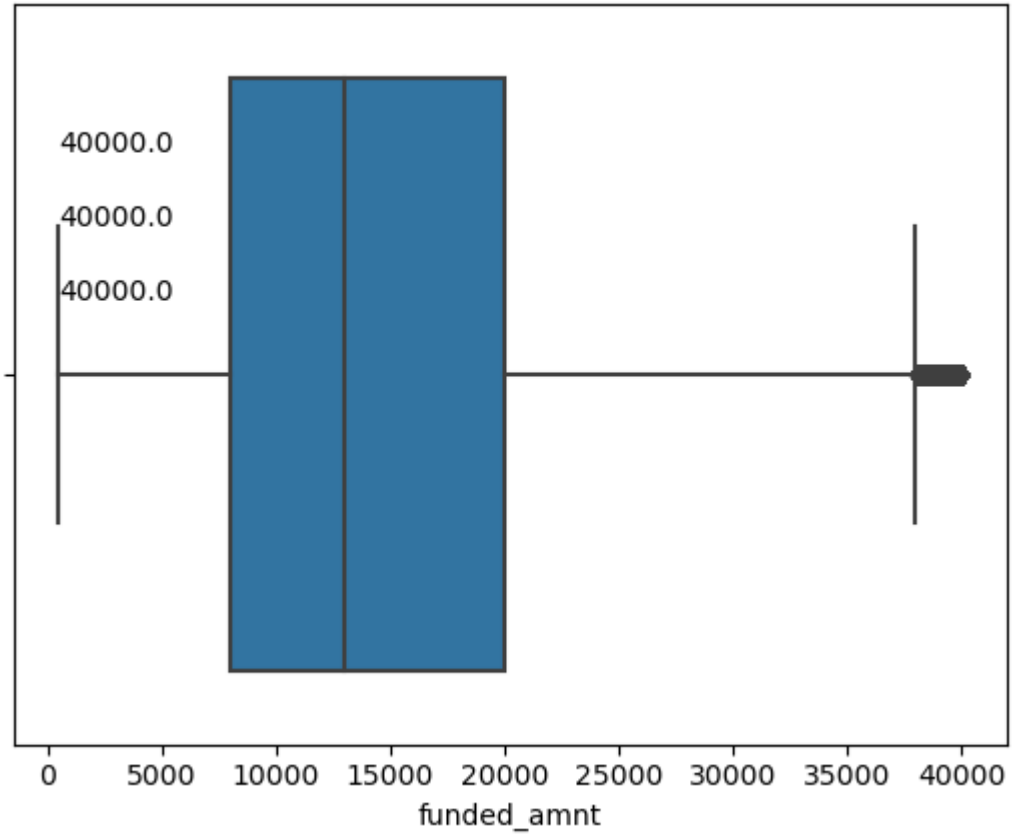
In [25]:
```
def visualize_date_columns():
    '''
    This function visualizes a timeline density for dates
    '''

    # Date columns
    for i in date_cols:
        final_data[final_data[i].isnull() == False][i].apply(lambda x : str(x.year) +
                                                "-" + str(x.month)).value_counts(ascending = True).plot()
        plt.title(i + " (" + str(final_data[i].isnull().sum()) + " null values)")
        plt.show()
```

In [26]:
```
# visualize continuous features
visualize_float_columns()

# visulaize categorical features
visualize_cat_columns()

# visualize date columns
visualize_date_columns()
```

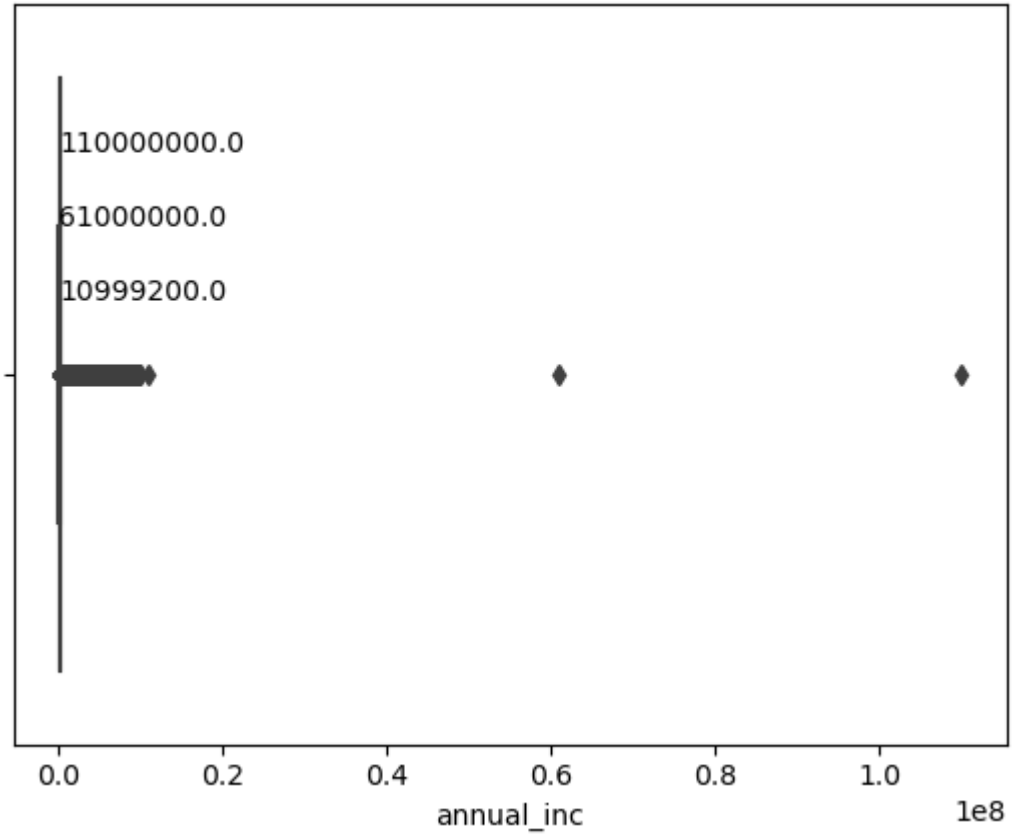
```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



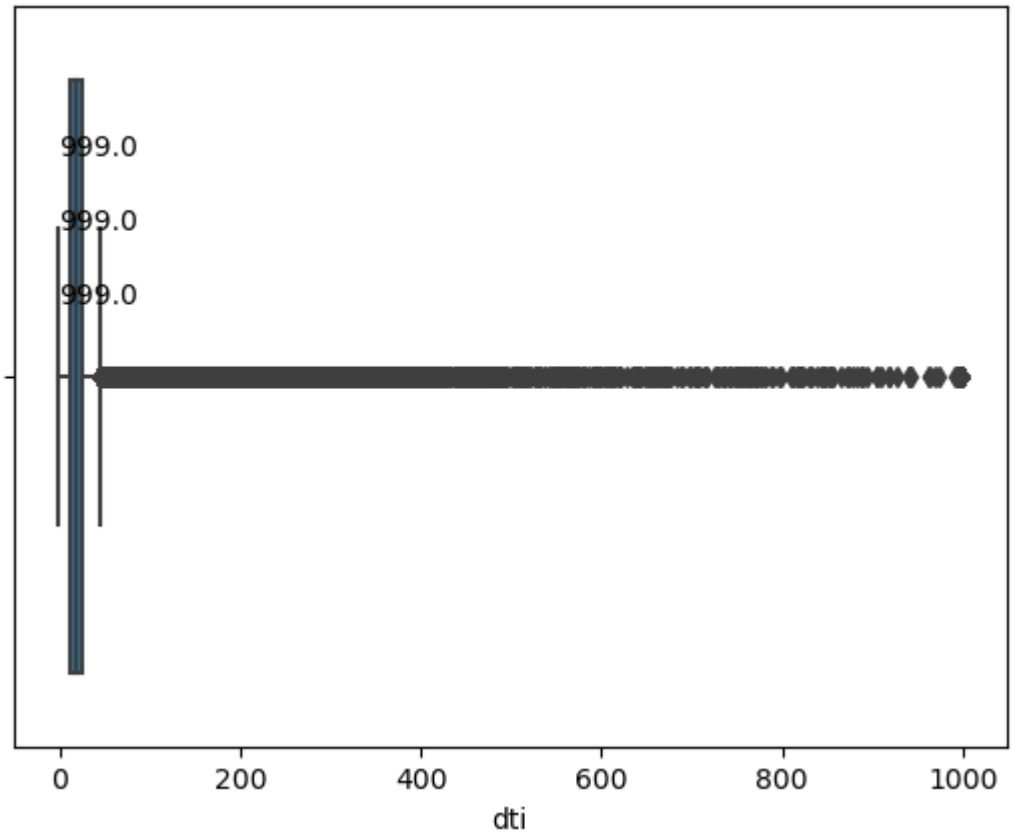

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



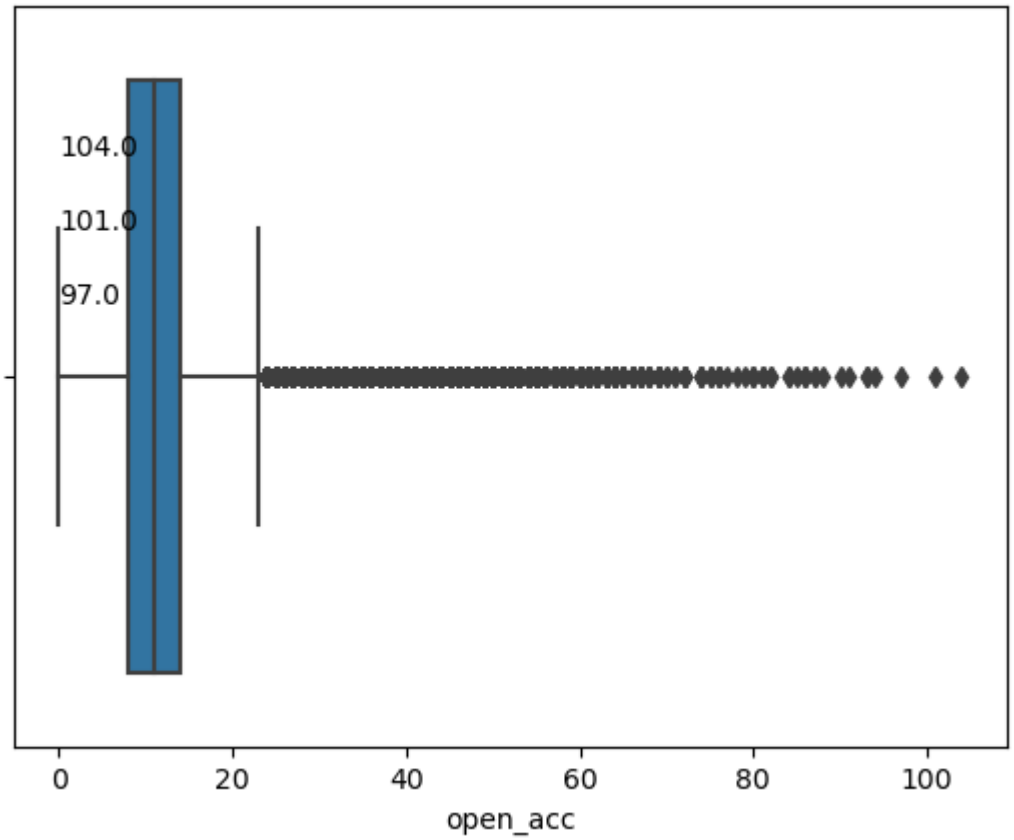

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



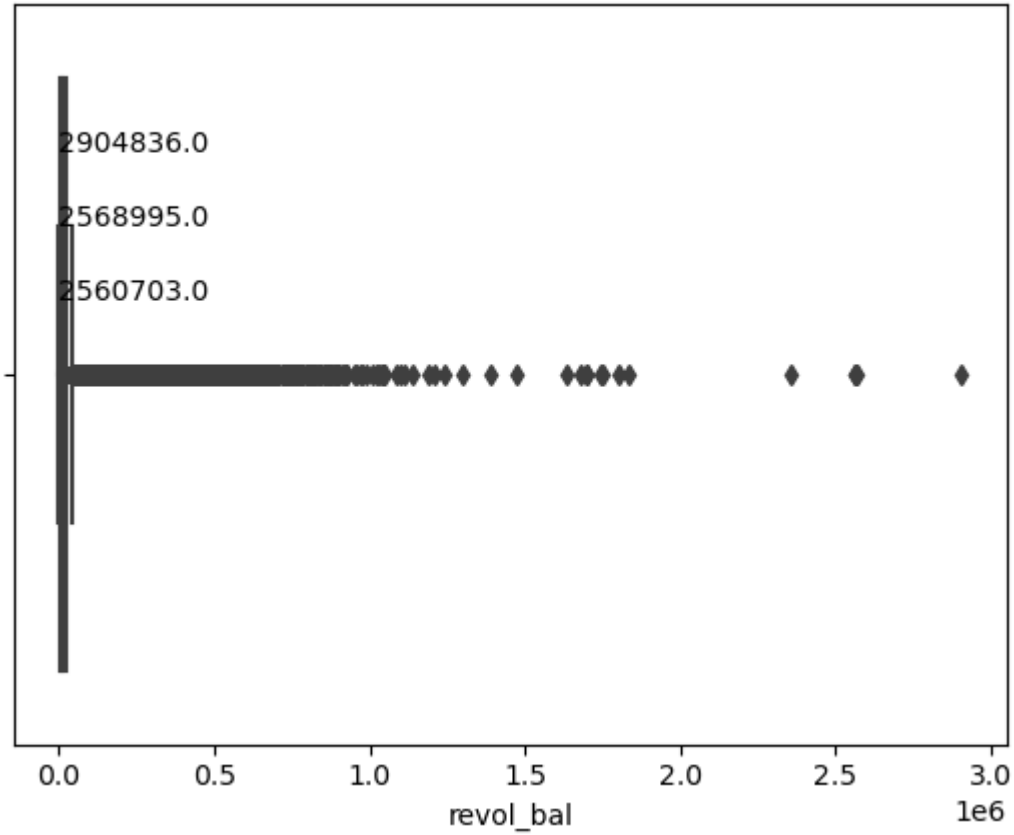

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



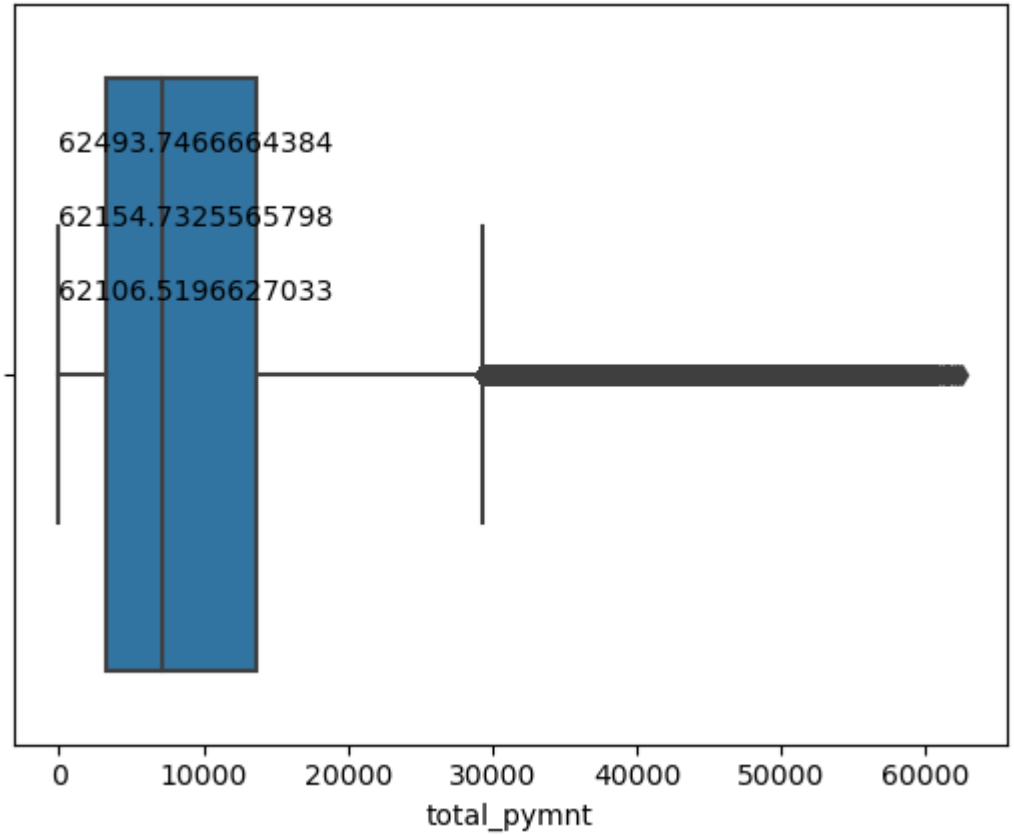

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



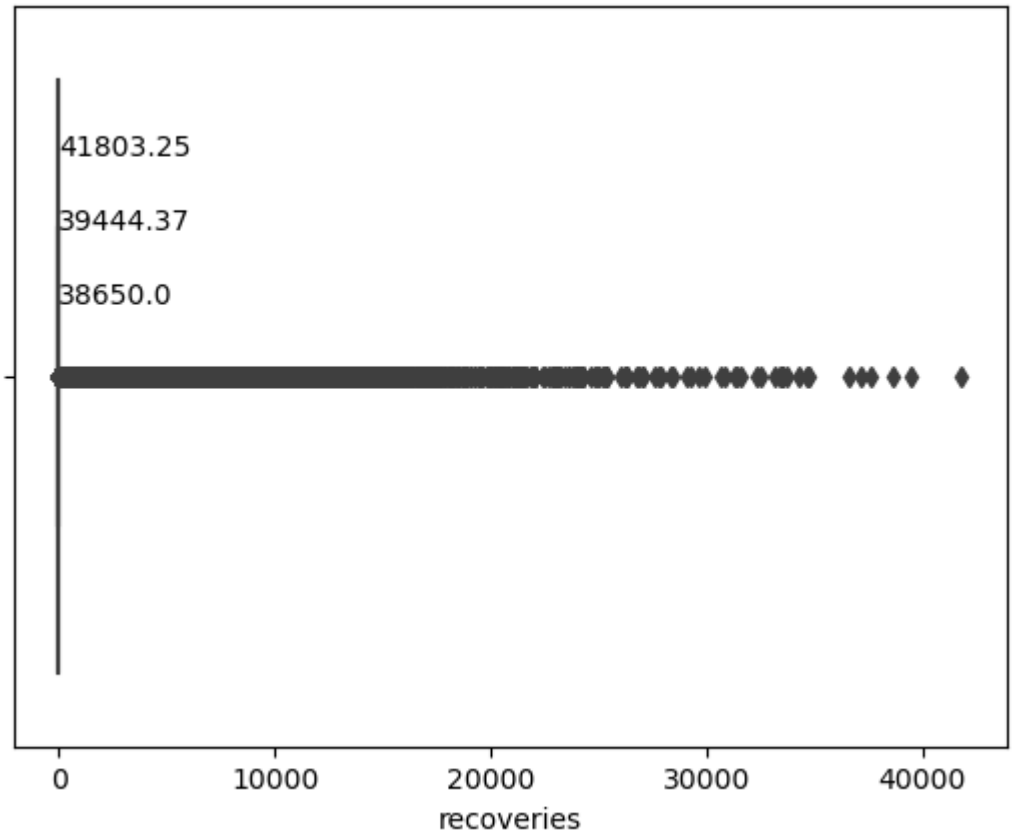

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



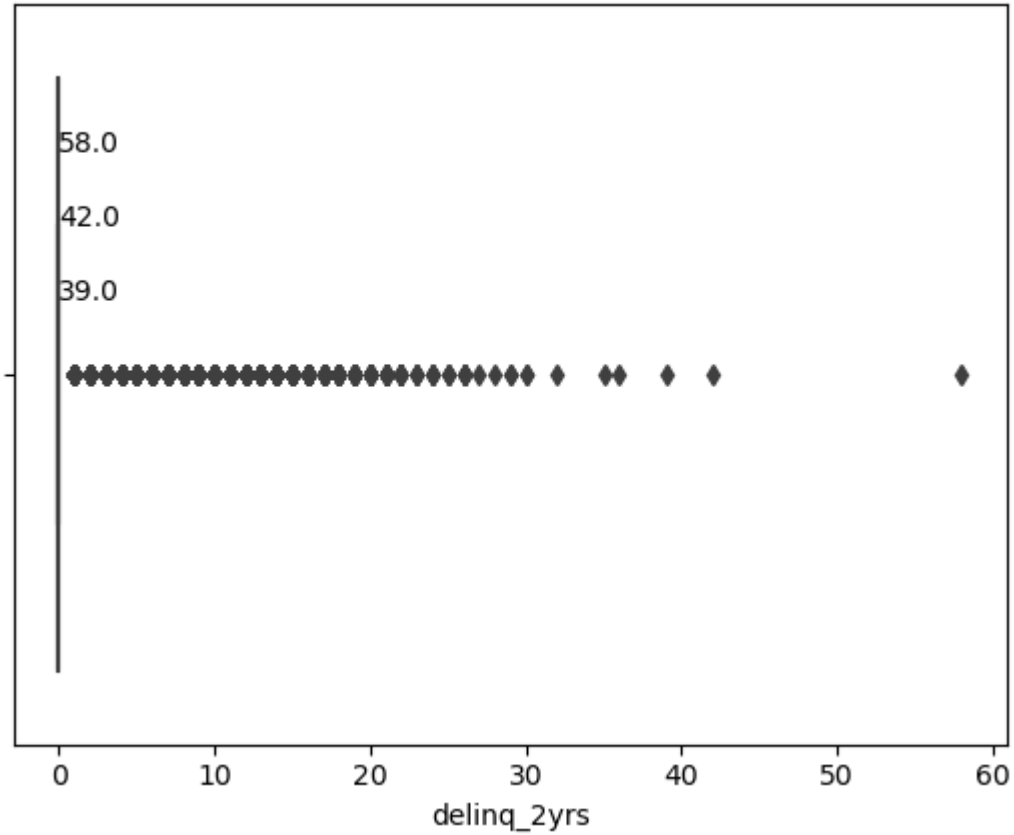

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



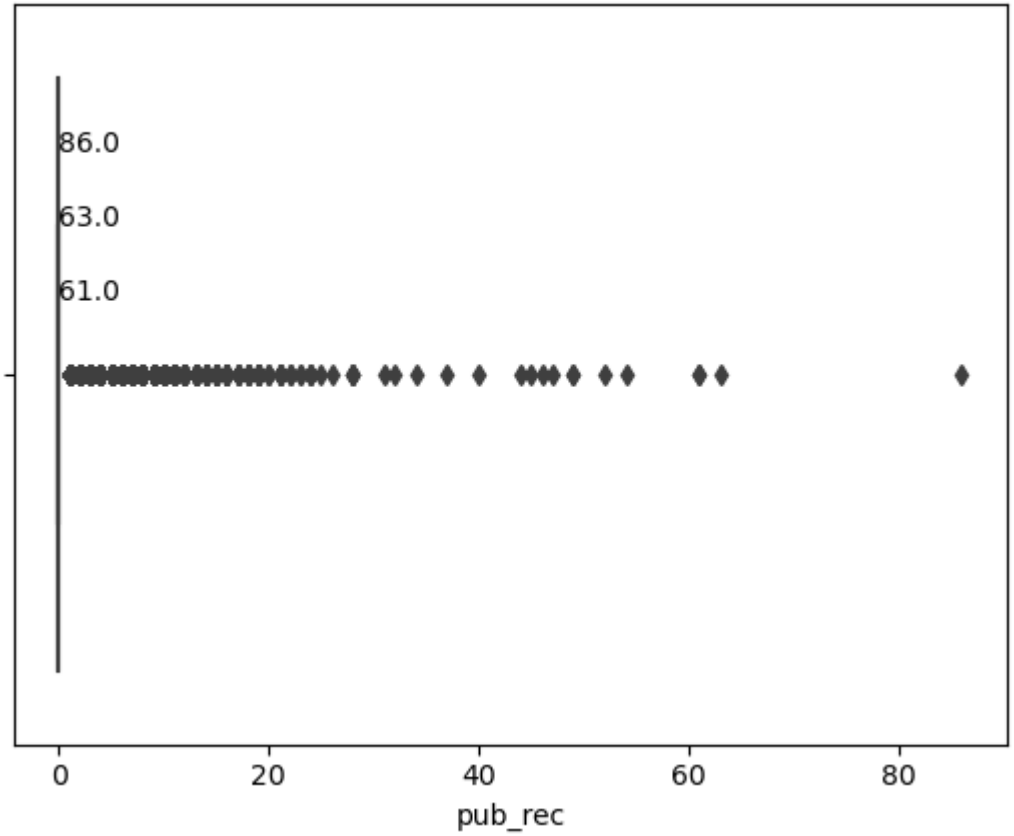

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



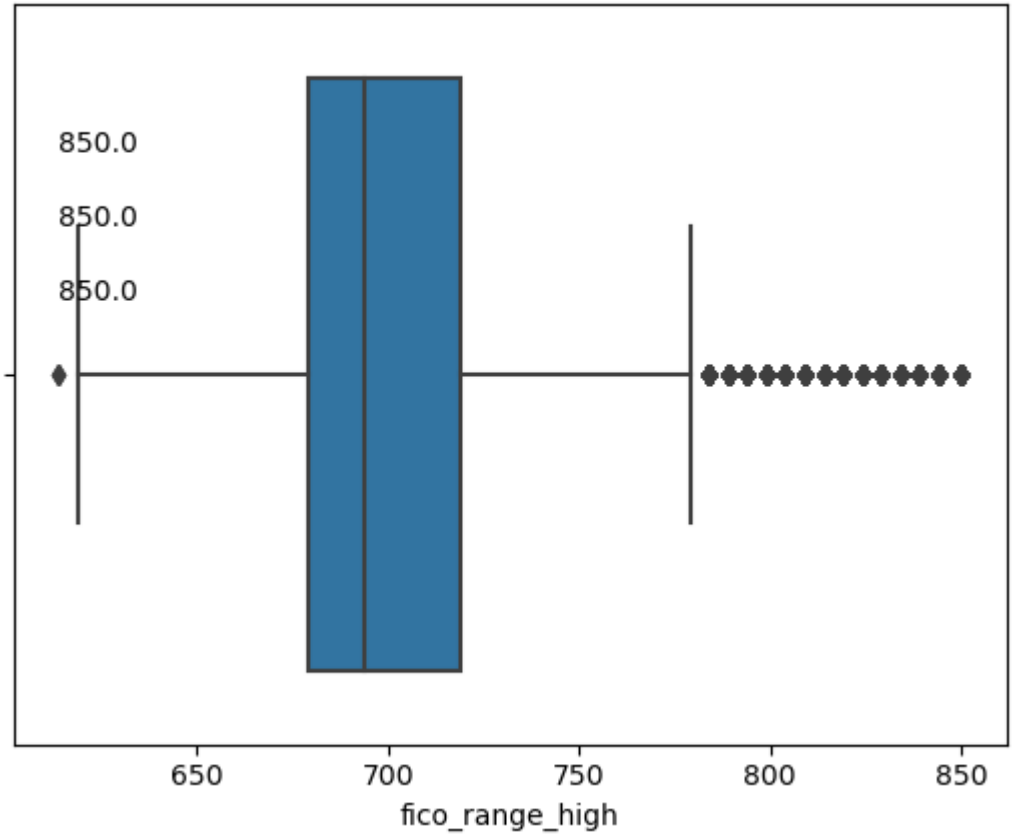

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



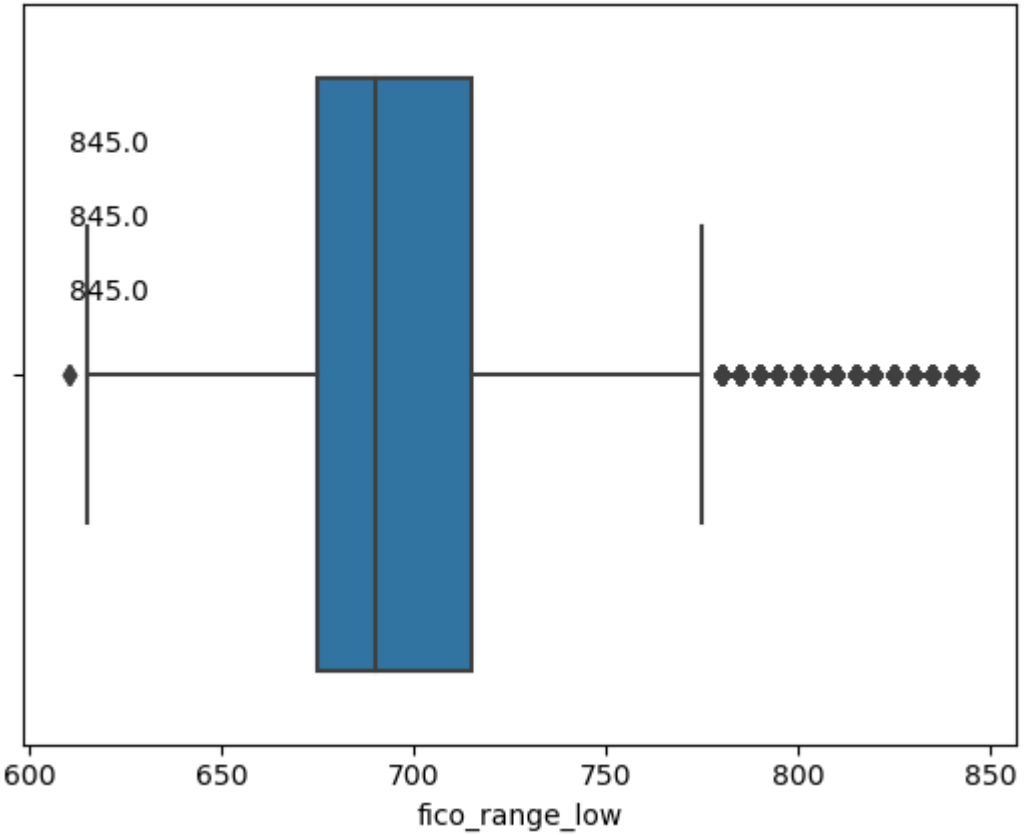

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



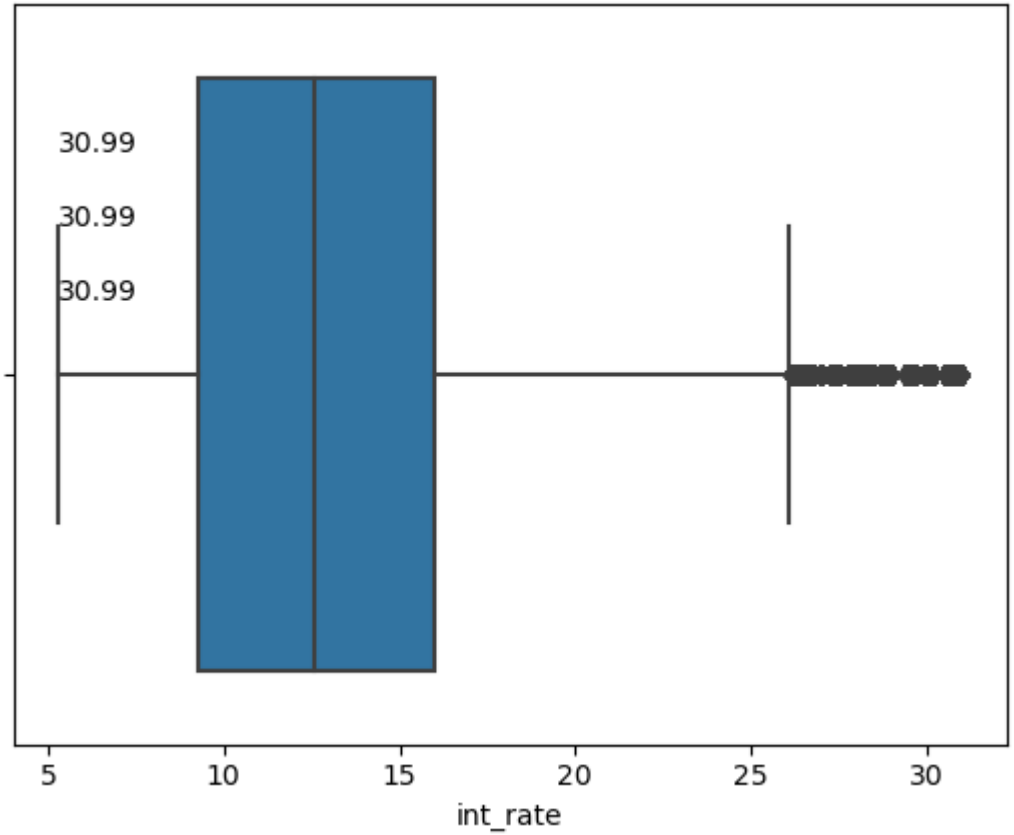

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



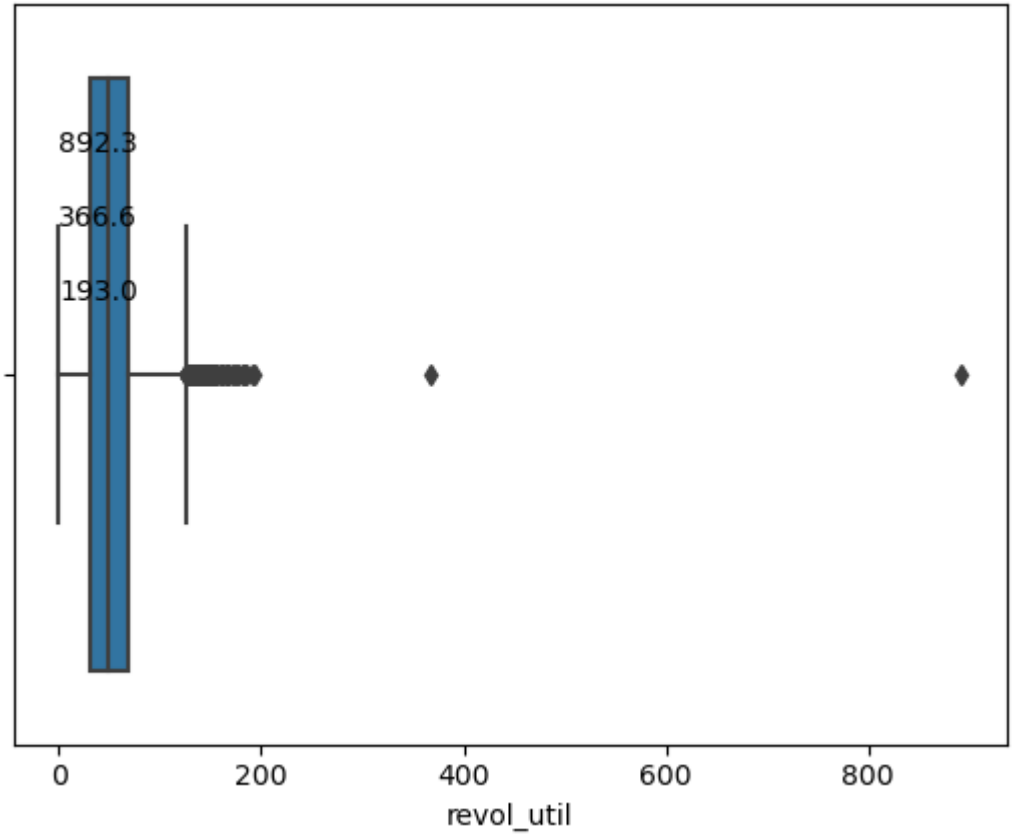

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



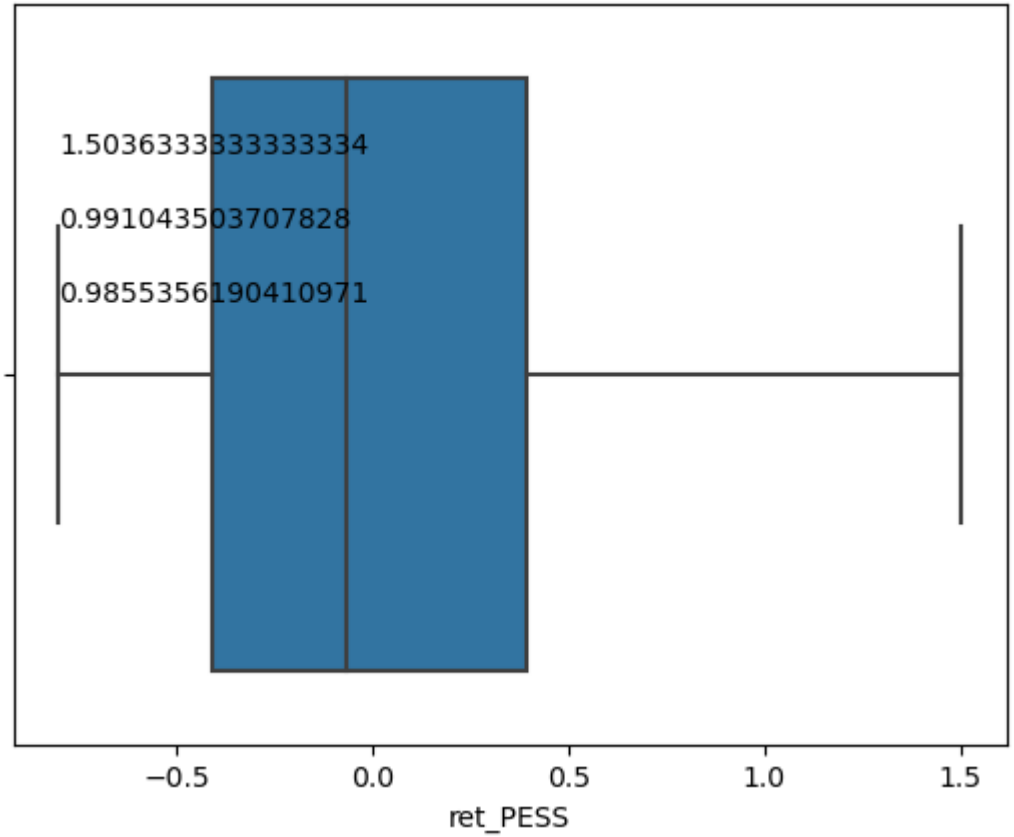

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



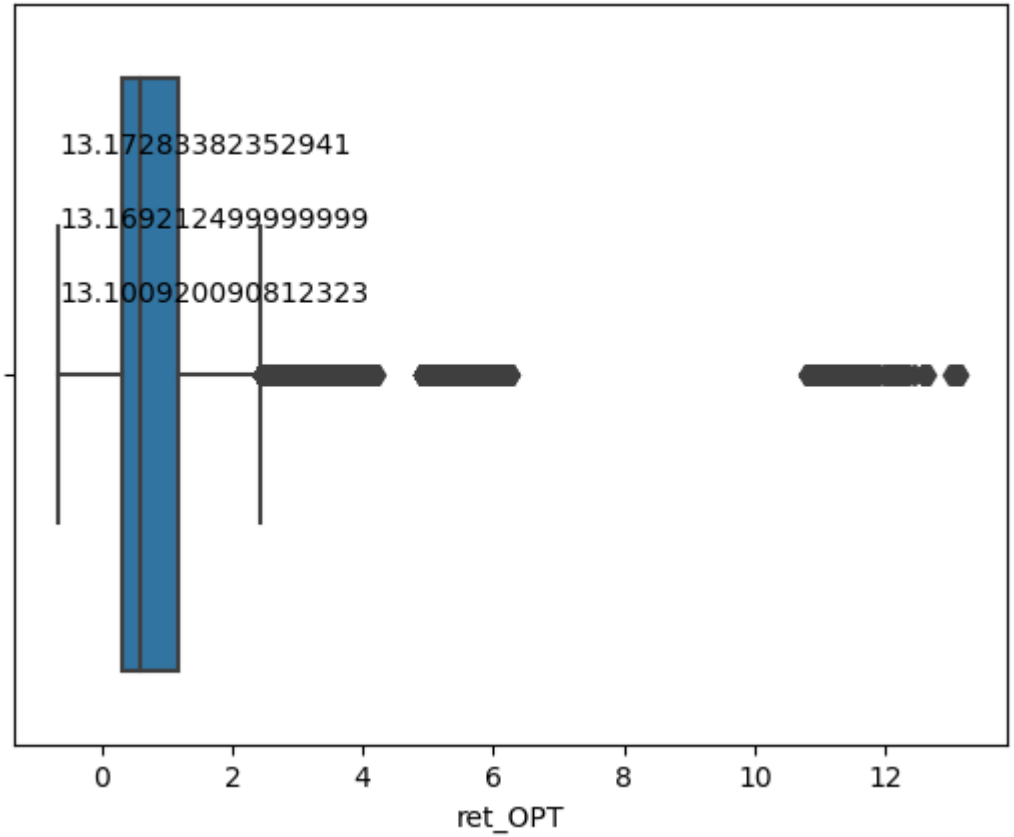

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



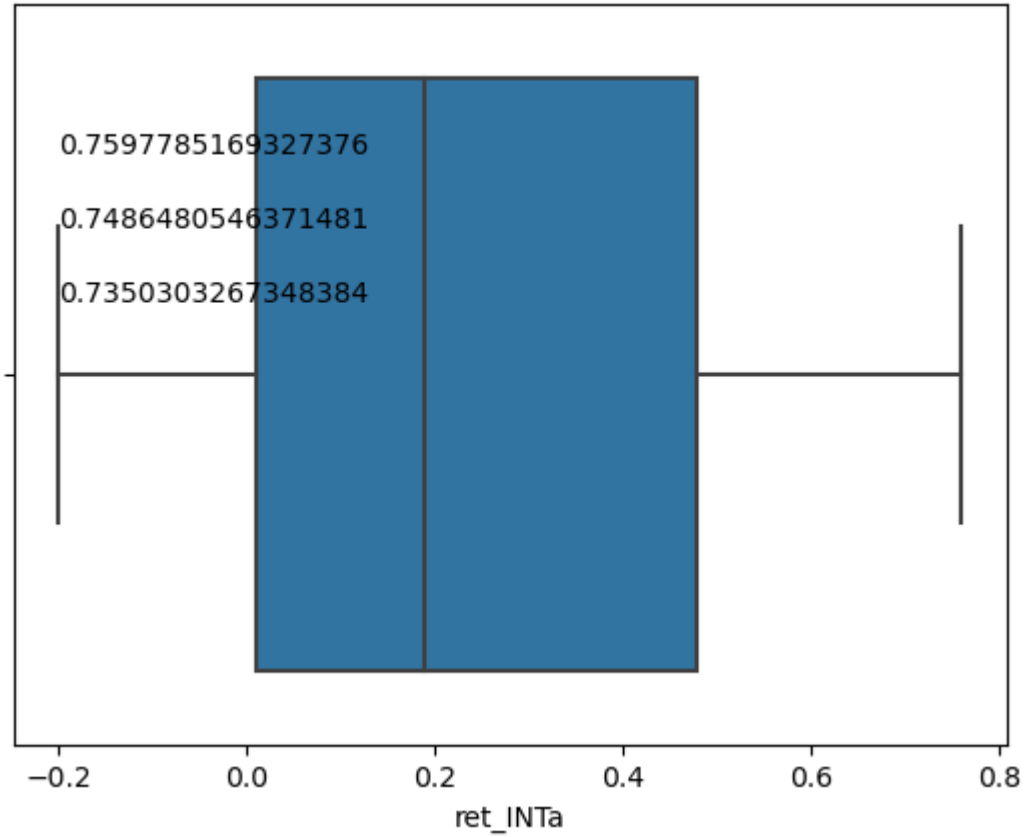

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



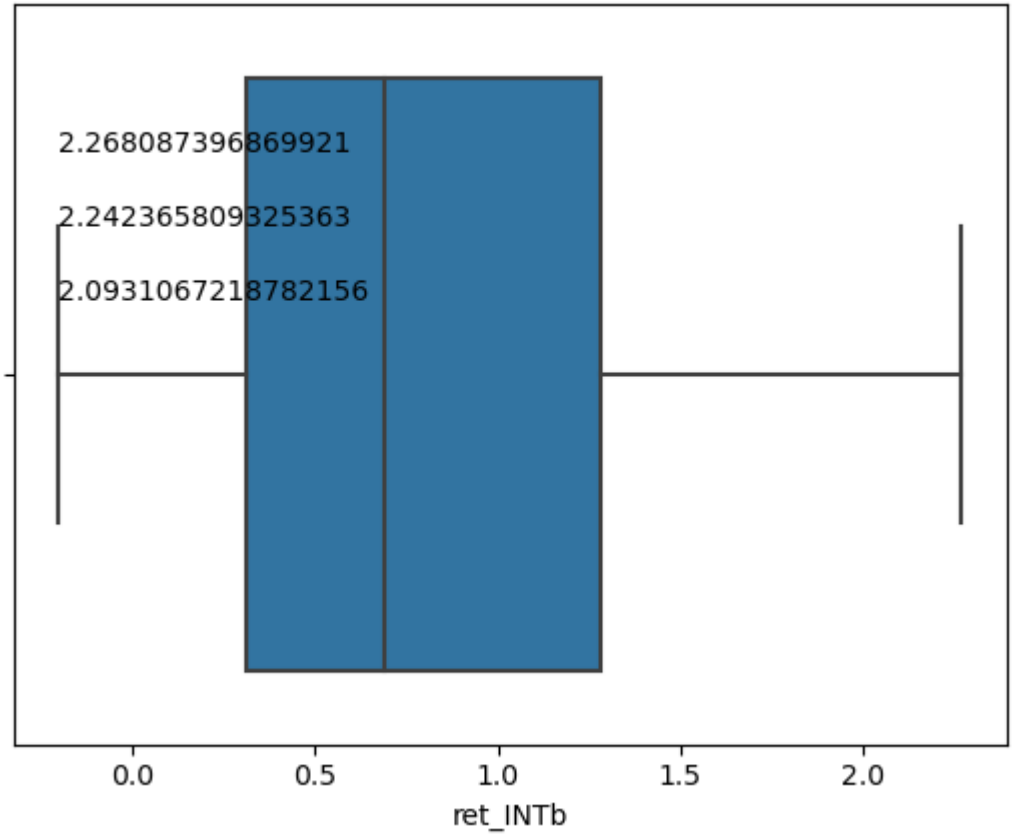

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```




```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

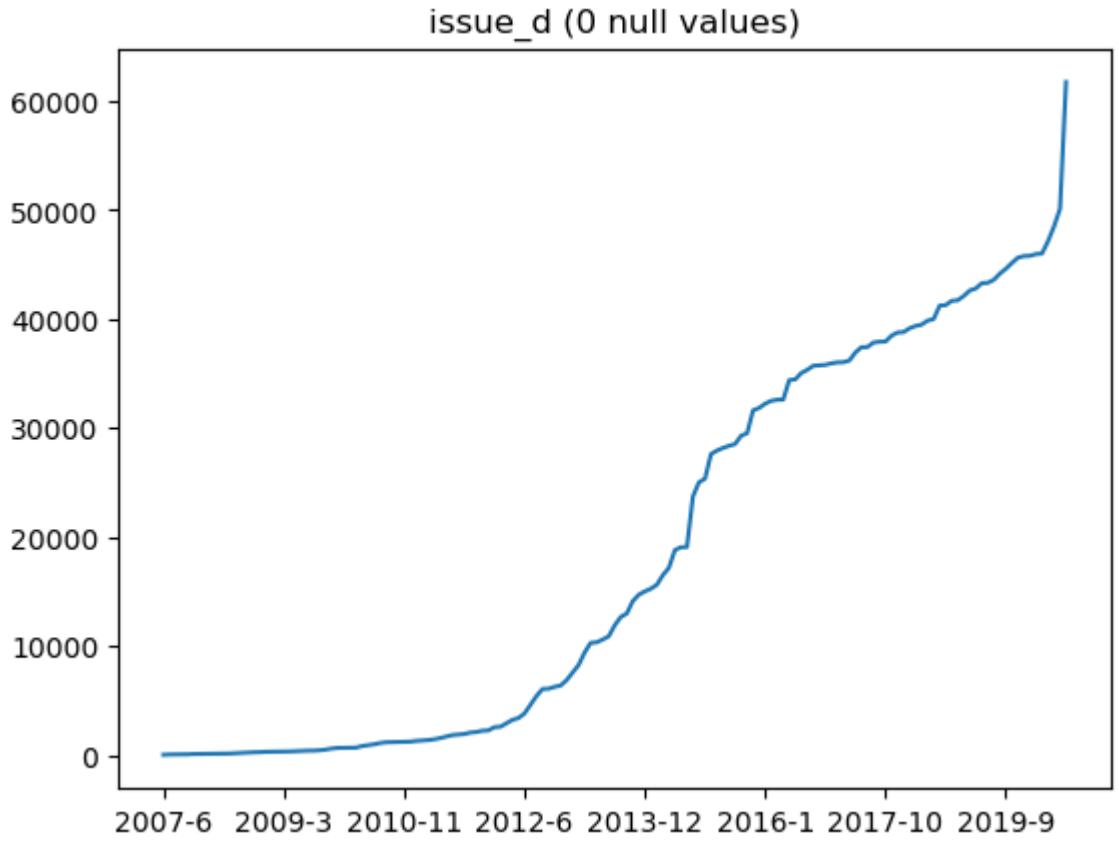


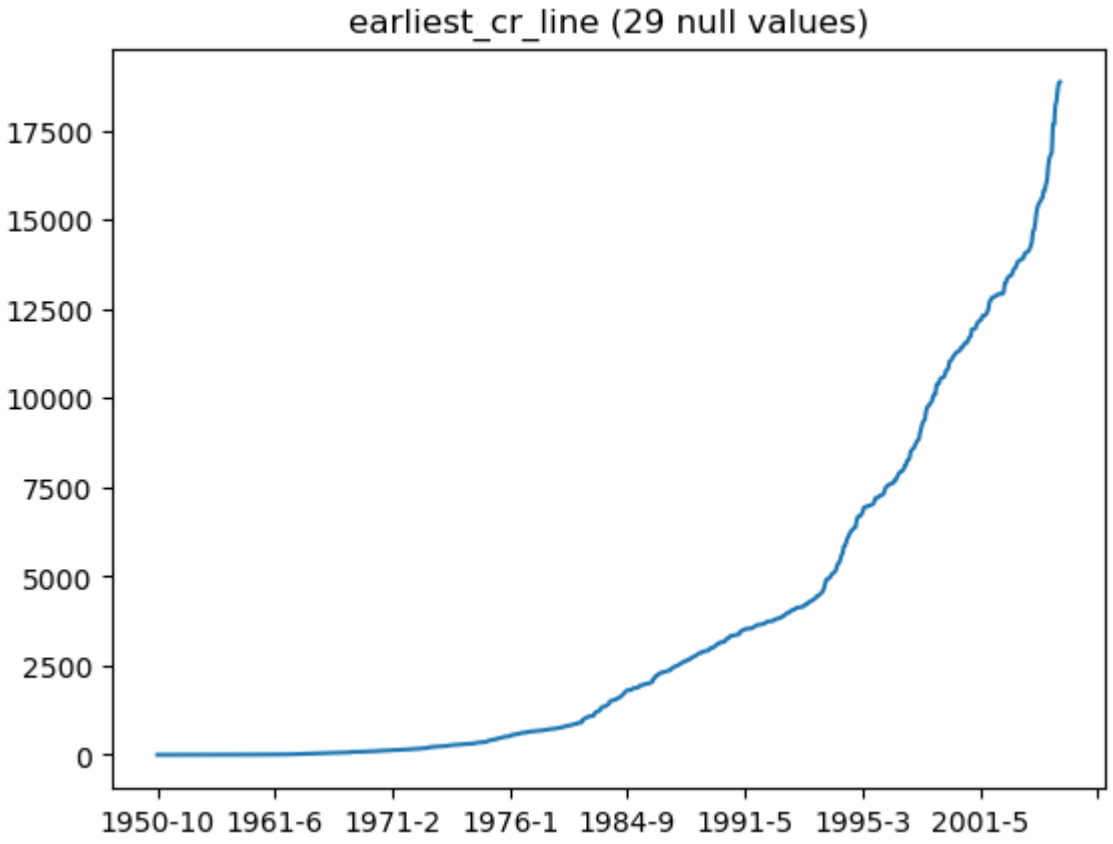

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

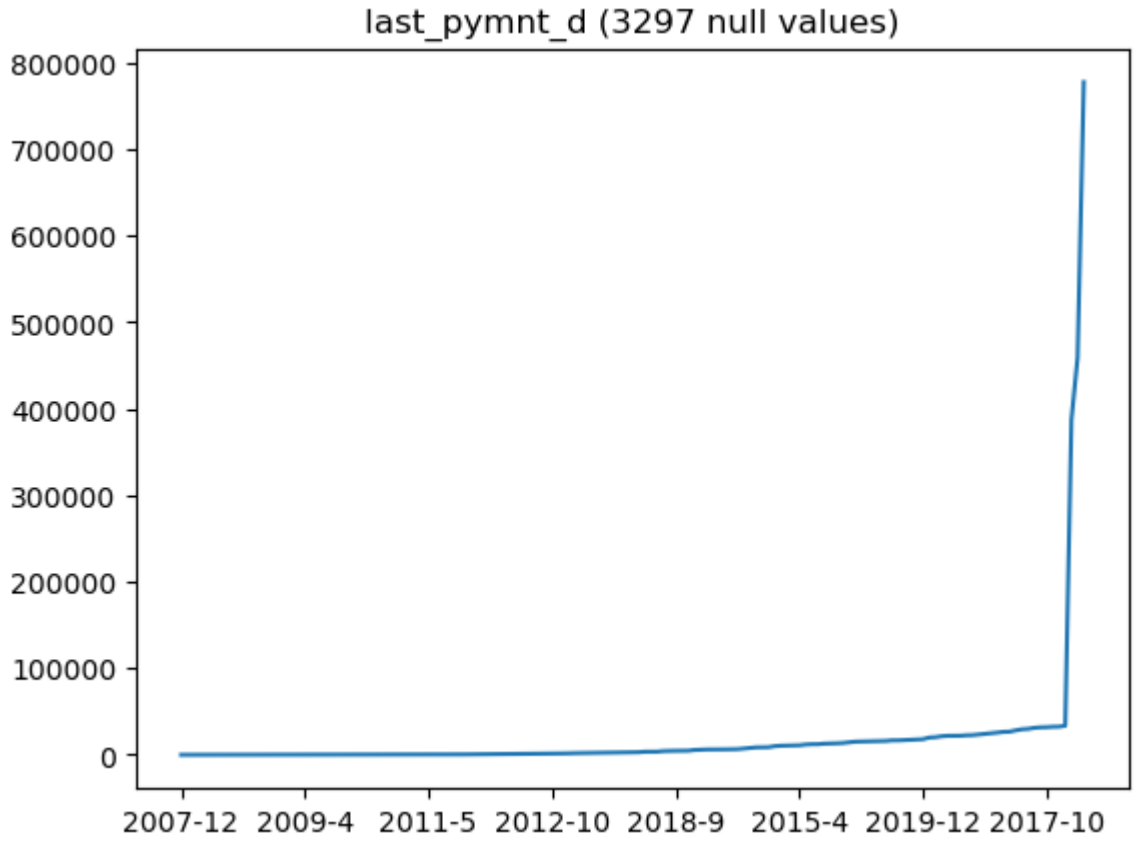

2.268087396869921
2.242365809325363
2.0931067218782156
0.0
0.5
1.0
1.5
2.0
ret_INTb

```
Column name: term
Number of distinct values: 2
 36 months    1949756
 60 months     815659
Name: term, dtype: int64


Column name: grade
Number of distinct values: 7
B    811304
C    768632
A    596277
D    397211
E    138329
F     41596
G     12066
Name: grade, dtype: int64


Column name: emp_length
Number of distinct values: 11
10+ years    898370
< 1 year     253650
2 years      247379
3 years      219762
1 year       183001
5 years      171829
4 years      166566
6 years      123700
7 years      110120
8 years      107507
9 years       91449
Name: emp_length, dtype: int64


Column name: home_ownership
Number of distinct values: 6
MORTGAGE    1359007
RENT        1090600
OWN          312178
ANY            3393
OTHER           182
NONE             55
Name: home_ownership, dtype: int64
```

```
Column name: verification_status
Number of distinct values: 3
Source Verified    1066701
Not Verified        997623
Verified            701091
Name: verification_status, dtype: int64


Column name: loan_status
Number of distinct values: 9
Current                                                1559756
Fully Paid                                              901142
Charged Off                                             238095
Late (31-120 days)                                       34060
In Grace Period                                          19614
Late (16-30 days)                                         9054
Does not meet the credit policy. Status:Fully Paid        1988
Default                                                    945
Does not meet the credit policy. Status:Charged Off        761
Name: loan_status, dtype: int64


Column name: purpose
Number of distinct values: 14
debt_consolidation    1553188
credit_card            653825
home_improvement       181492
other                  167714
major_purchase          59092
medical                 33209
small_business          28590
car                     28186
vacation                19189
house                   18356
moving                  18109
wedding                  2355
renewable_energy         1686
educational               424
Name: purpose, dtype: int64
```


issue_d (0 null values)
60000
50000
40000
30000
20000
10000
0
2007-6
2009-3
2010-11
2012-6
2013-12
2016-1
2017-10
2019-9


earliest_cr_line (29 null values)
17500
15000
12500
10000
7500
5000
2500
0
1950-10
1961-6
1971-2
1976-1
1984-9
1991-5
1995-3
2001-5



In [27]:
```
final_data['dti']
```

Out[27]:
```
0          27.65
1           1.00
2           8.72
3          20.00
4          17.94
           ...
2765410    30.01
2765411    14.18
2765412     6.00
2765413     2.10
2765414     8.76
Name: dti, Length: 2765415, dtype: float64
```

Handle outliers

In [28]:
```
# There are quite a few outliers.
# Please identify top-k (decide this based on the visualization) features where outliers are most obvious
#def iqr(feature):
    #q1=feature.describe()[4]
    #q3=feature.describe()[6]
    #iqr=q3-q1
    #return [q1-1.5*iqr,q3+1.5*iqr]

n_rows = len(final_data)

final_data = final_data[final_data['annual_inc']<10000000] # remove outliers based 1st obvious feature
final_data = final_data[final_data['revol_bal']<2000000]
final_data = final_data[final_data['revol_util']<250]
final_data = final_data[final_data['delinq_2yrs']<31]
final_data = final_data[final_data['pub_rec']<60]

# remove outliers based 2nd obvious feature
...
#final_data = ... # remove outliers based kth obvious feature

print("Removed " + str(n_rows - len(final_data)) + " rows")
```

```
Removed 2479 rows
```

In [29]:
```
# Remove all loans that are still current
n_rows = len(final_data)

final_data = final_data[final_data['loan_status']!='Current']

print("Removed " + str(n_rows - len(final_data)) + " rows")
```

```
Removed 1558250 rows
```

In [30]:
```
# Only include loans isssued since 2010
n_rows = len(final_data)

final_data = final_data[final_data['issue_d']>=datetime.date(2010,1,1)]

print("Removed " + str(n_rows - len(final_data)) + " rows")
```

```
Removed 8217 rows
```

Drop null values

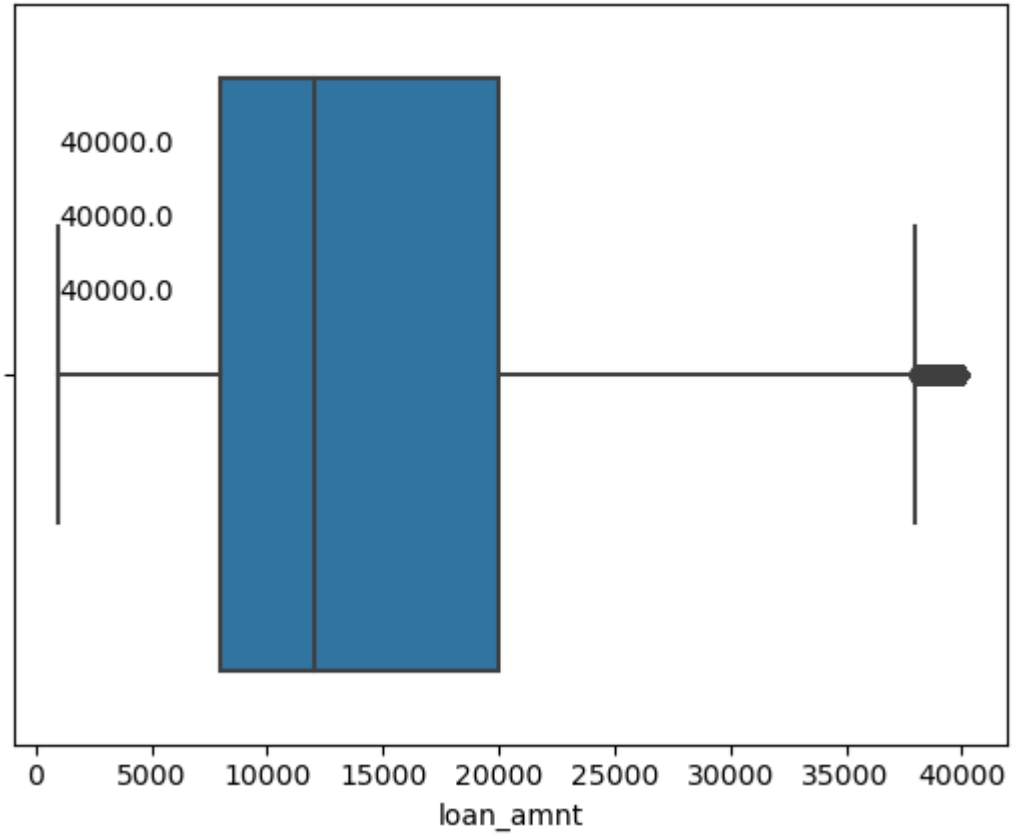
In [31]:
```
# Deal with null values. We allow cateogrical variables to be null
# OTHER than grade, which is a particularly important categorical.
# All non-categorical variables must be non-null, and we drop
# rows that do not meet this requirement

required_cols = set(cols_to_pick) - set(cat_cols) - set(["id"])
required_cols.add("grade")

n_rows = len(final_data)

final_data.dropna(subset=list(required_cols),inplace=True) # drop rows that contain null based only on "required_cols"

print("Removed " + str(n_rows - len(final_data)) + " rows")
```

```
Removed 3963 rows
```

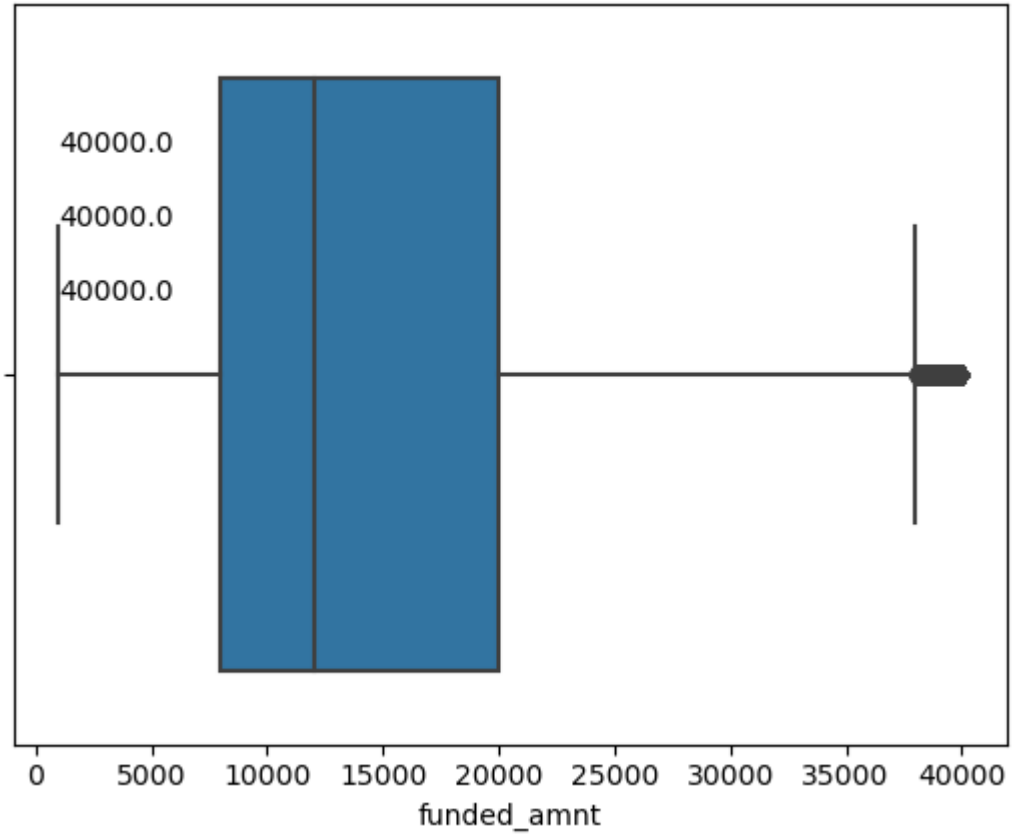
Visualize clean data

In [32]:
```
# Visualize the data again after cleaning
# visualize continuous features
visualize_float_columns()

# visulaize categorical features
visualize_cat_columns()

# visualize date columns
visualize_date_columns()
```

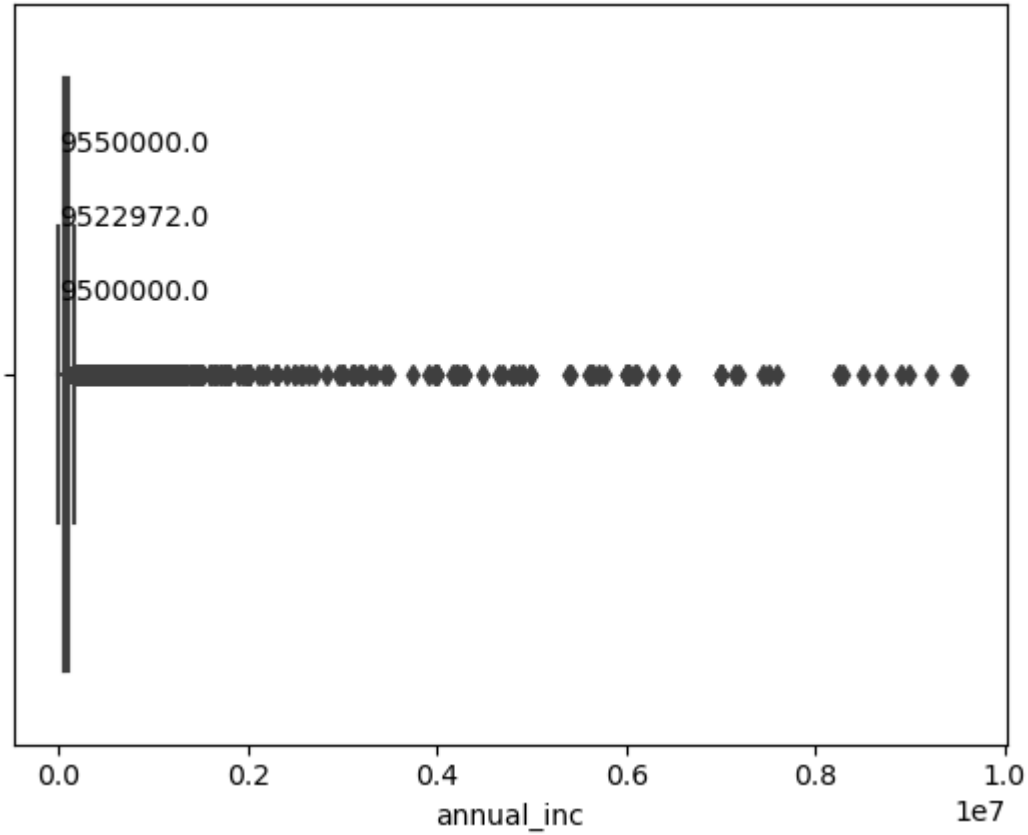
```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



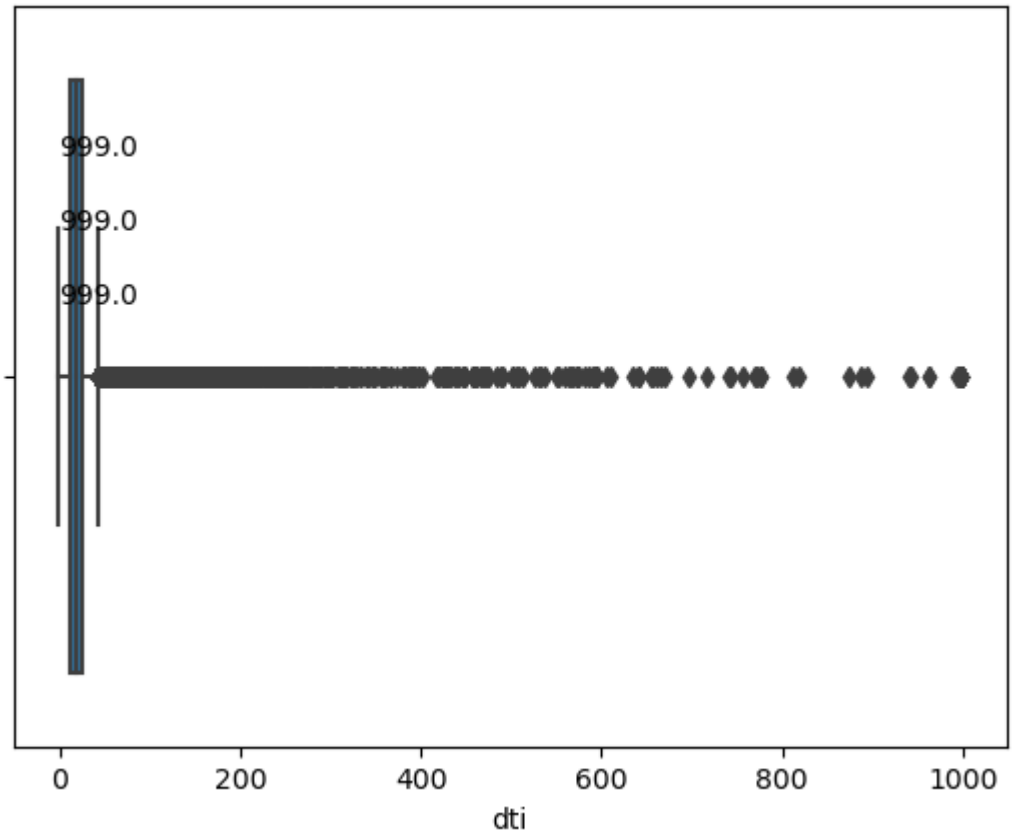

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



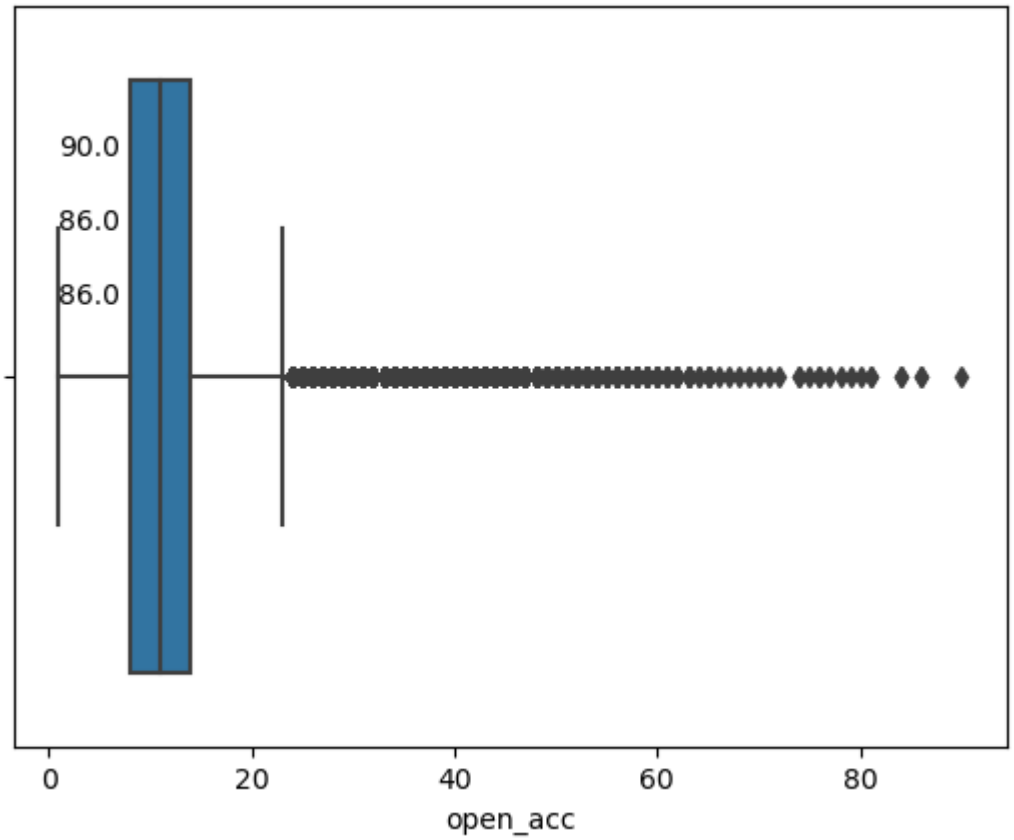

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



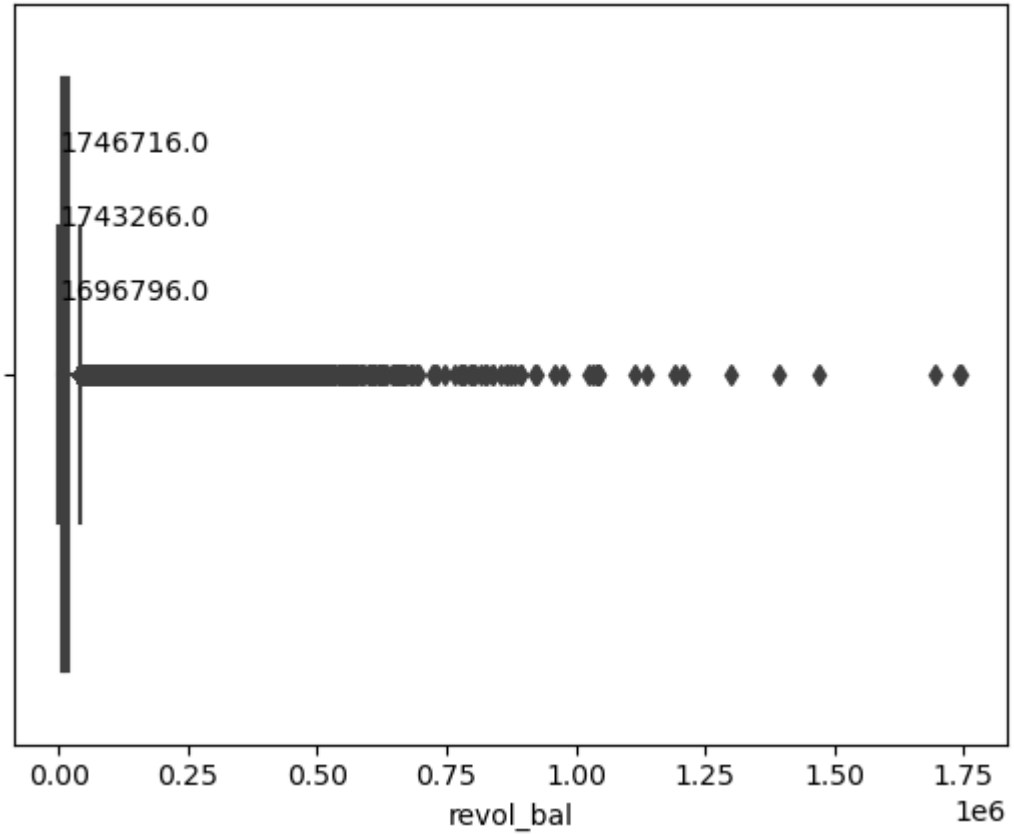

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



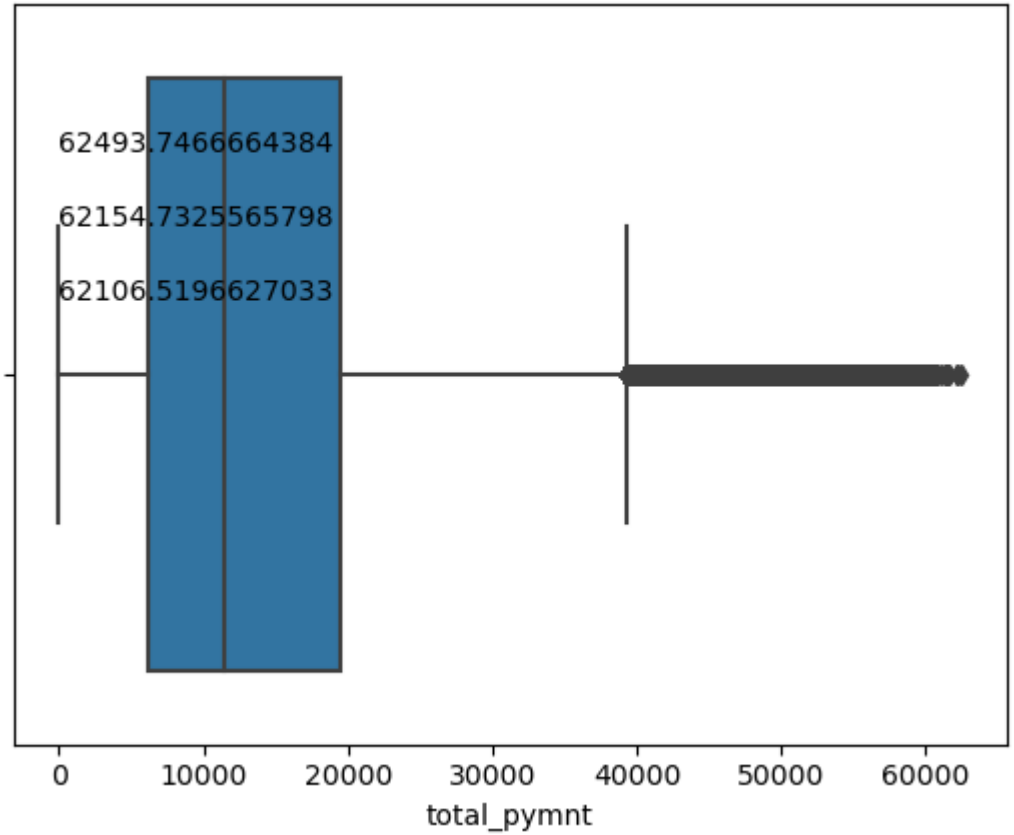

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



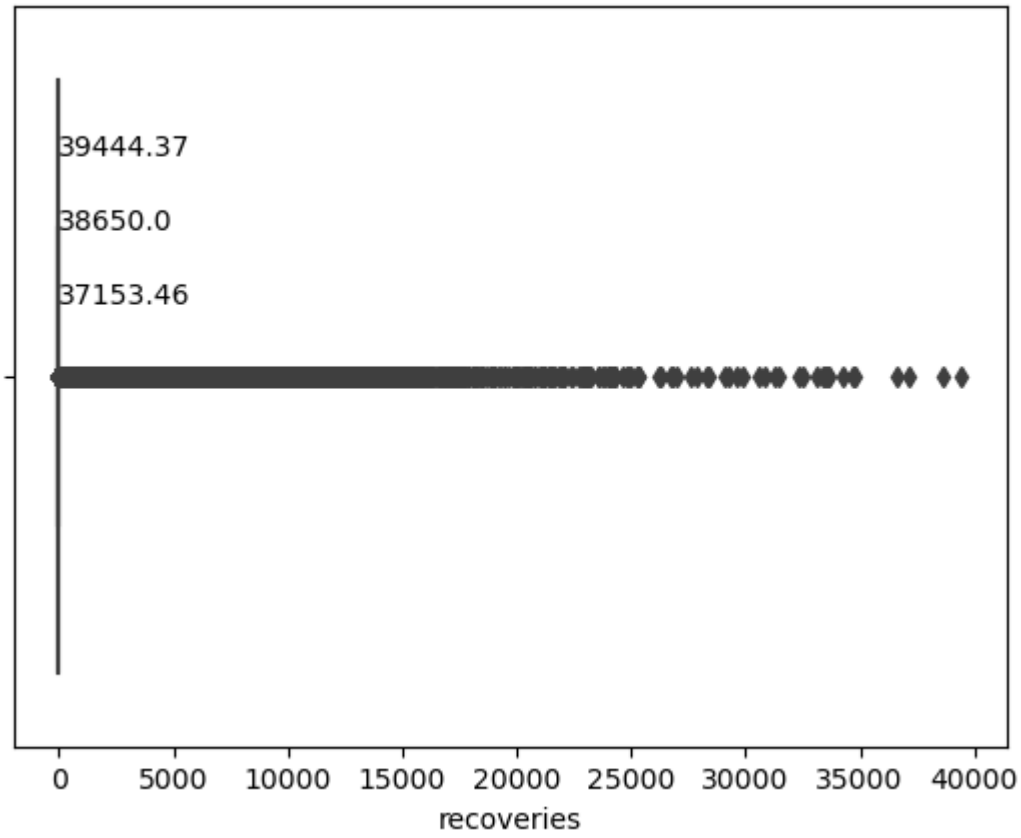

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



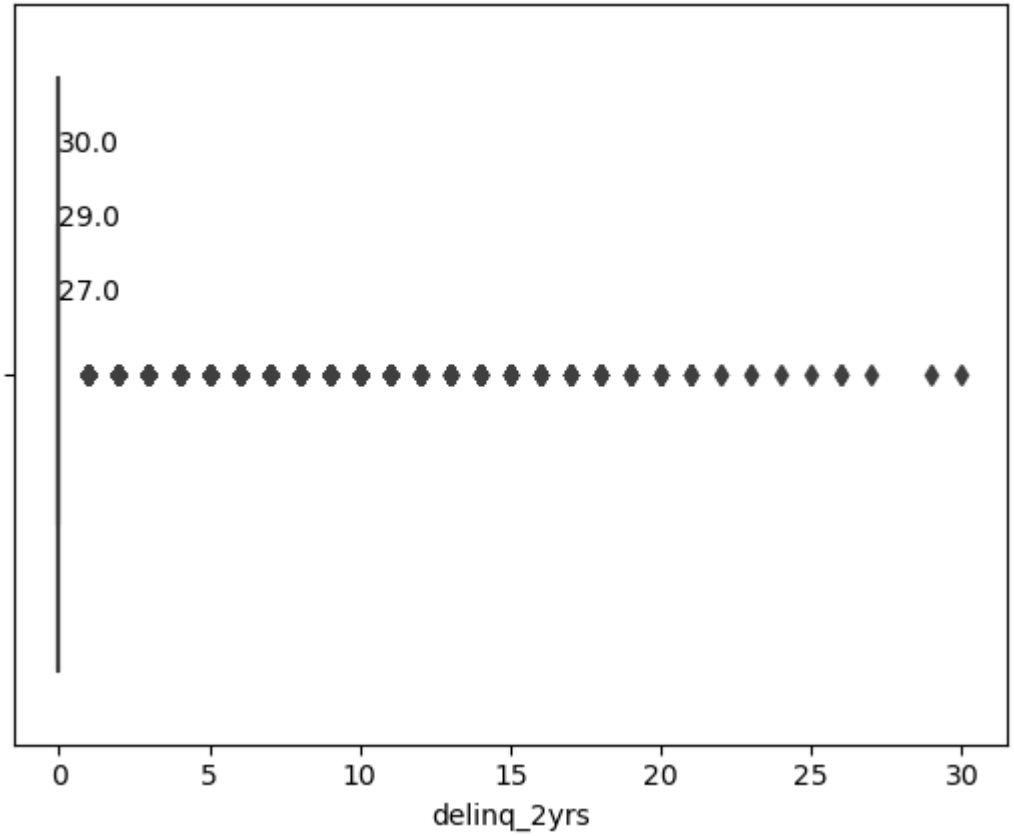

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



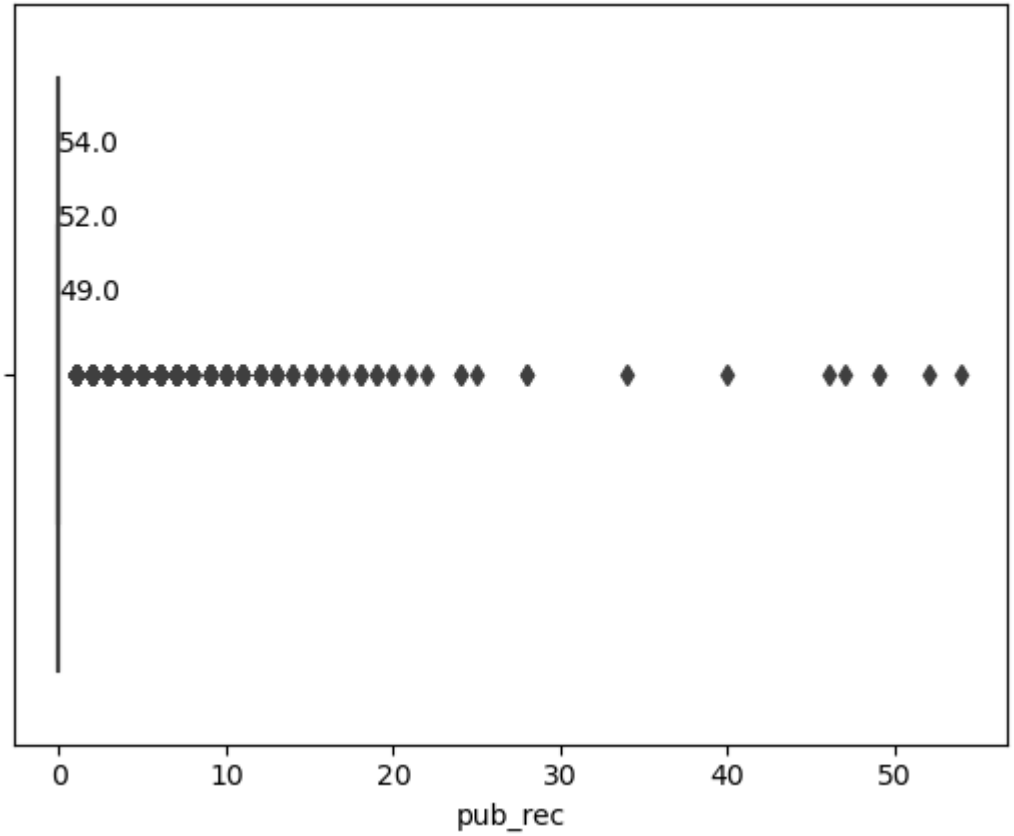

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



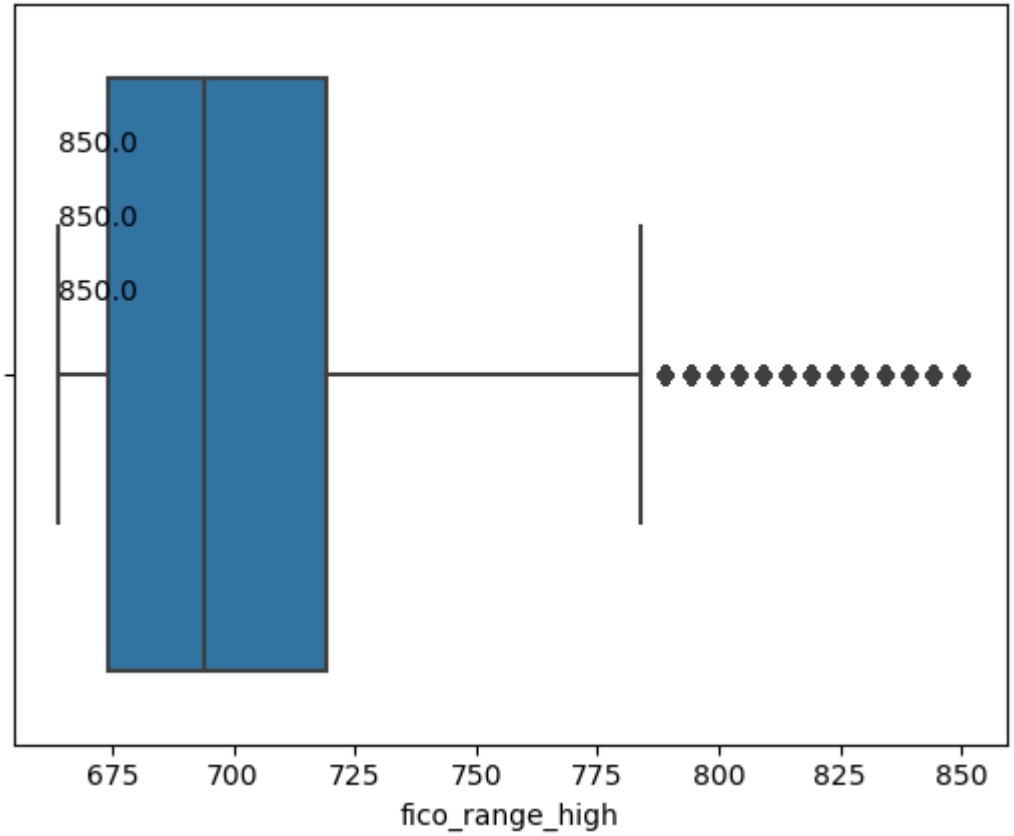

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



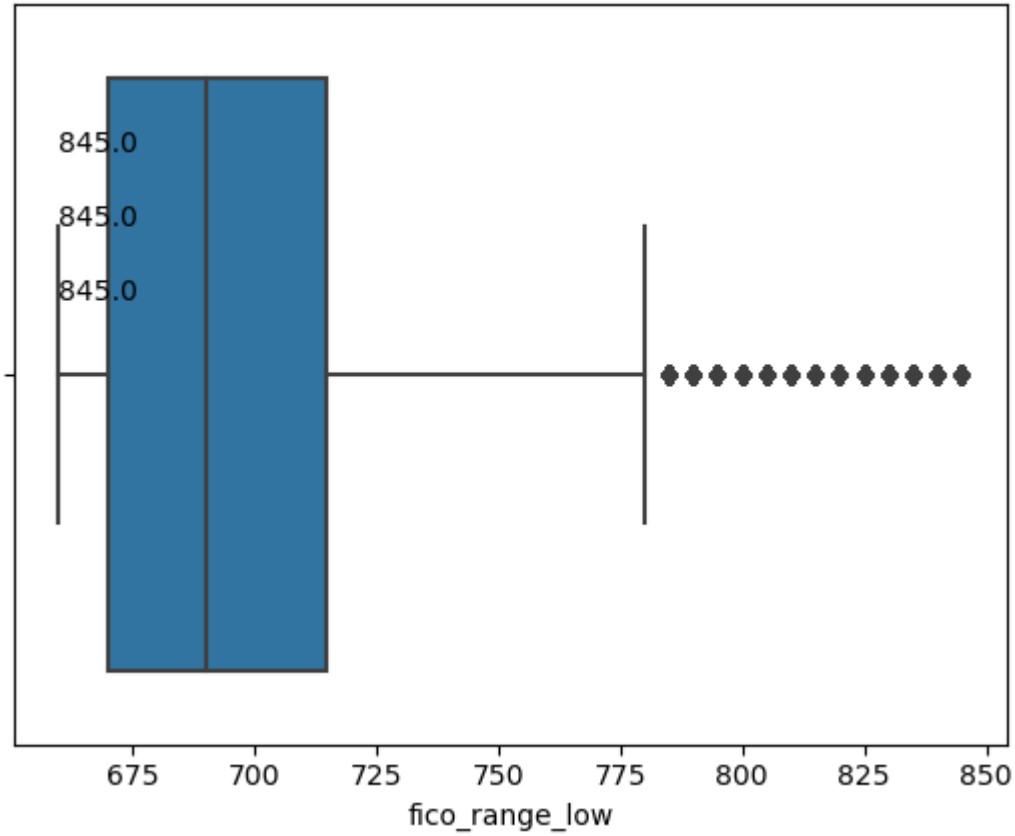

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



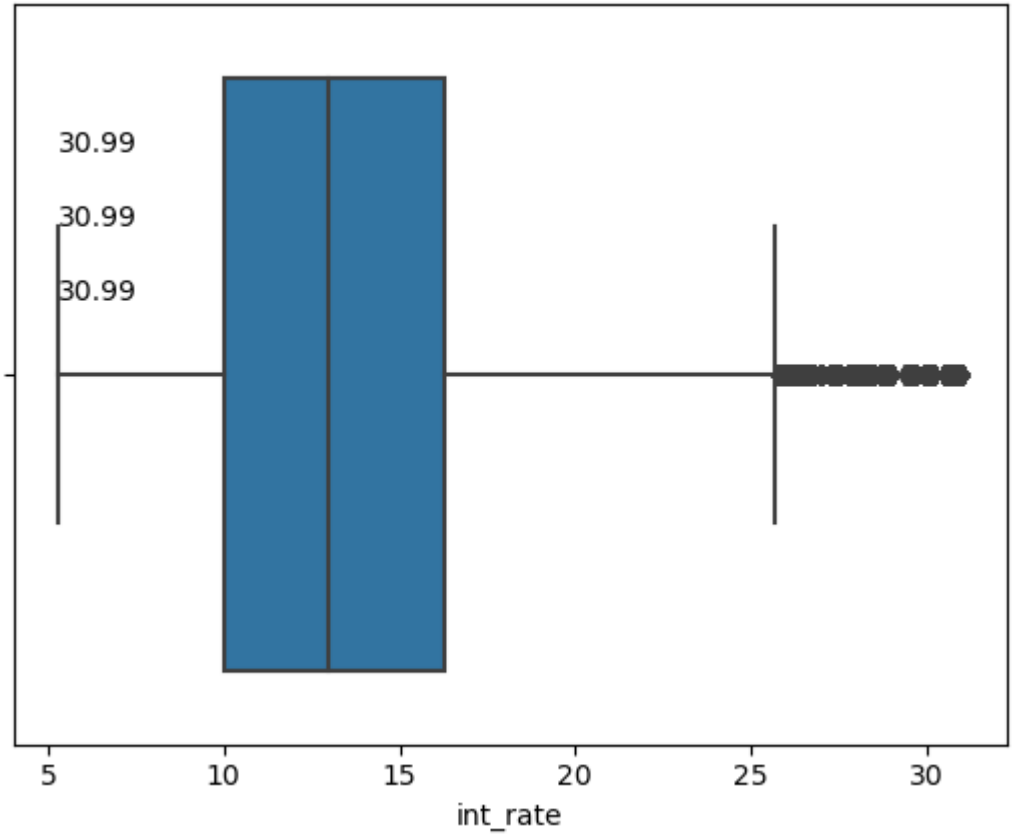

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



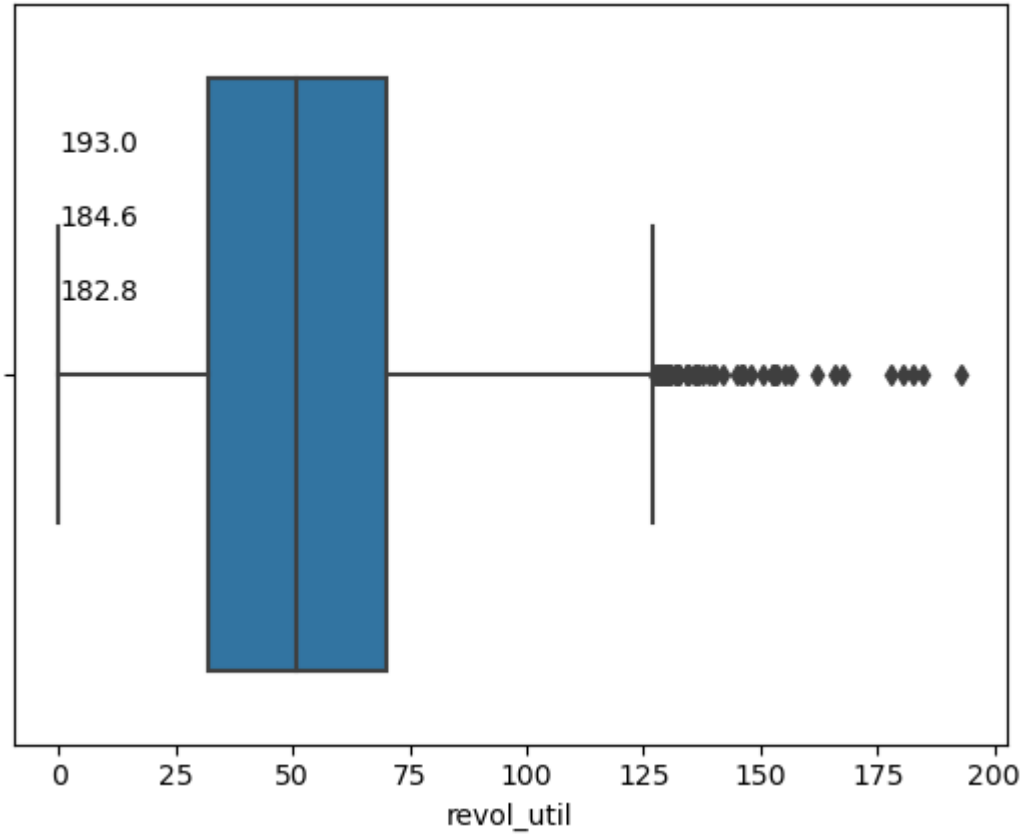

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



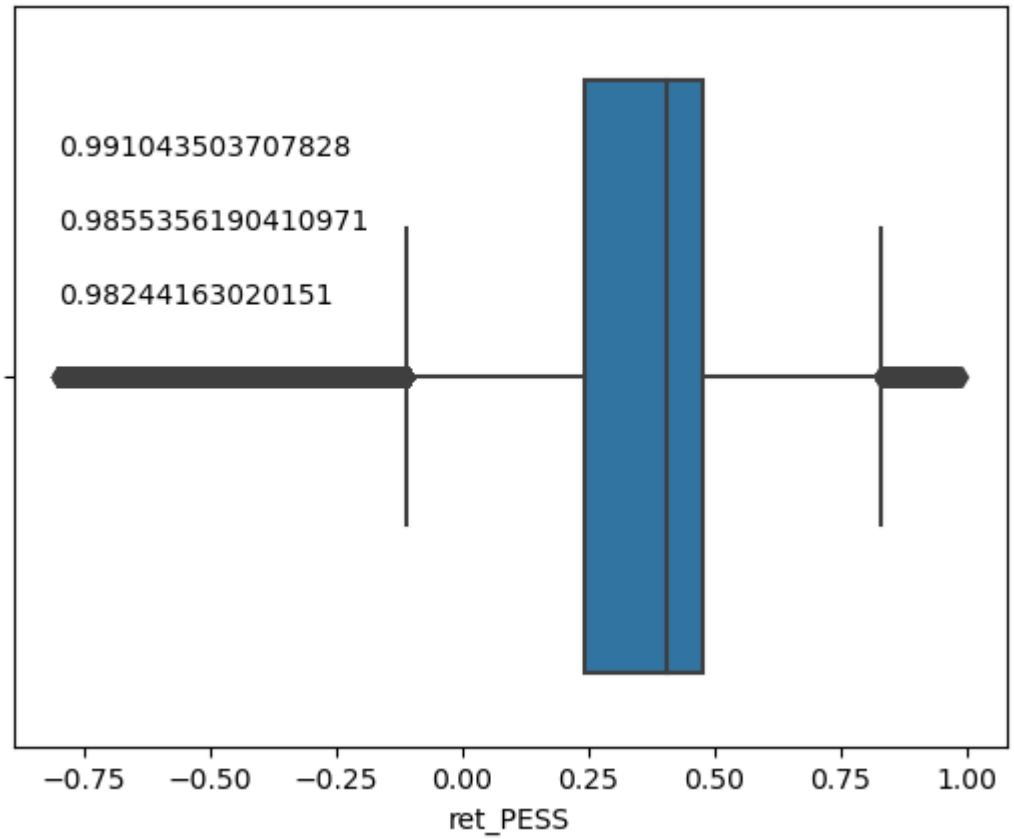

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



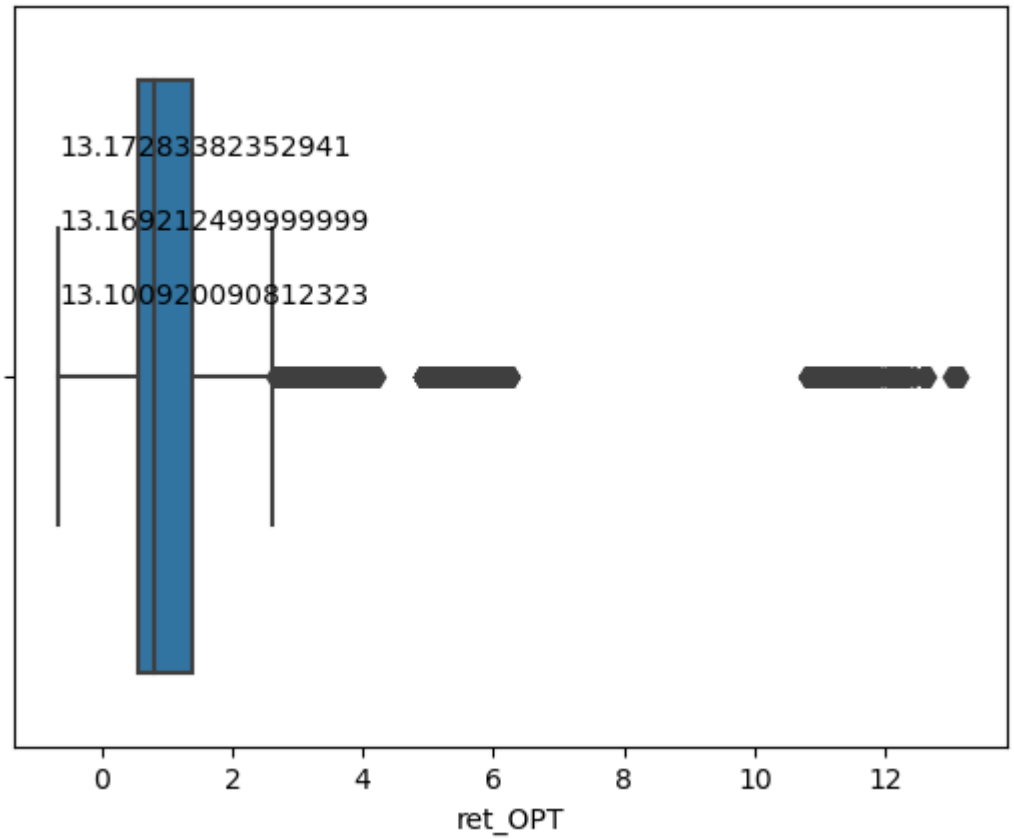

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



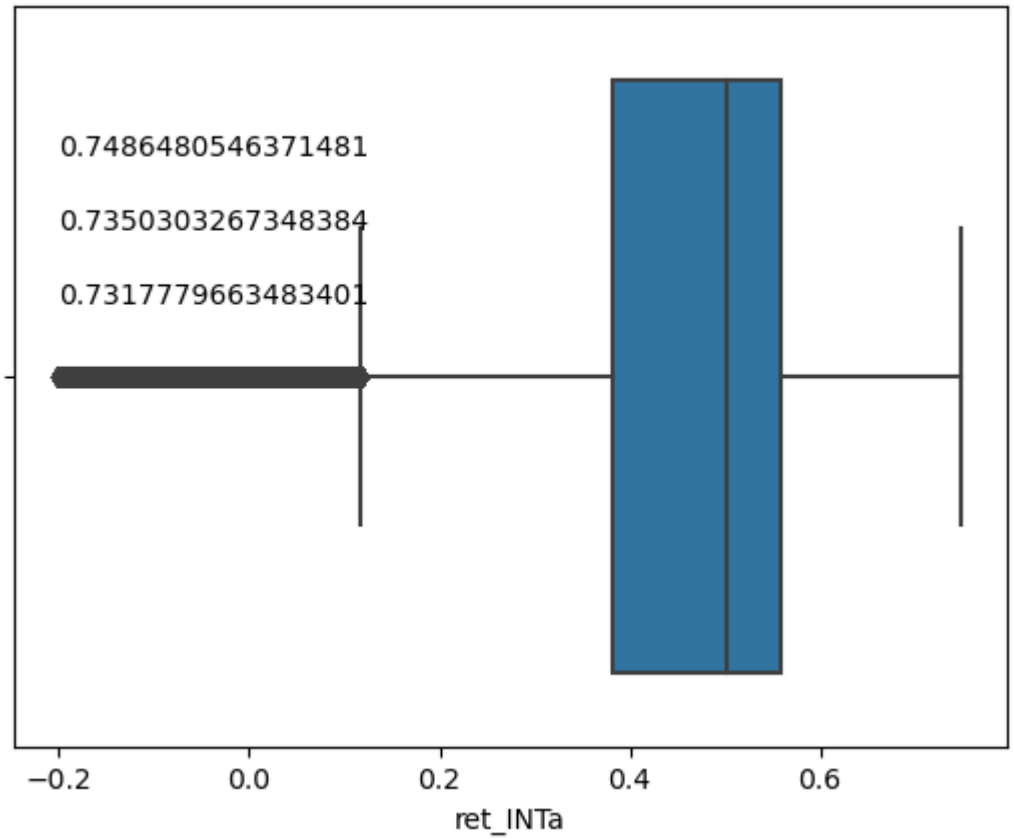

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



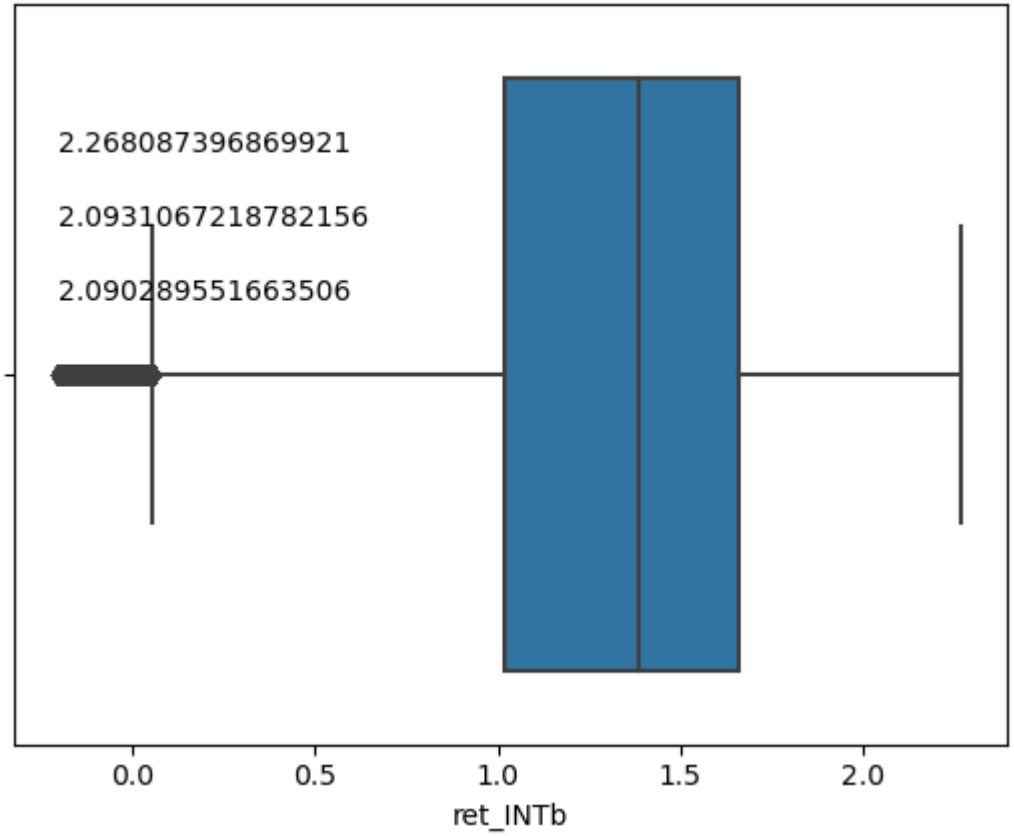

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```




```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

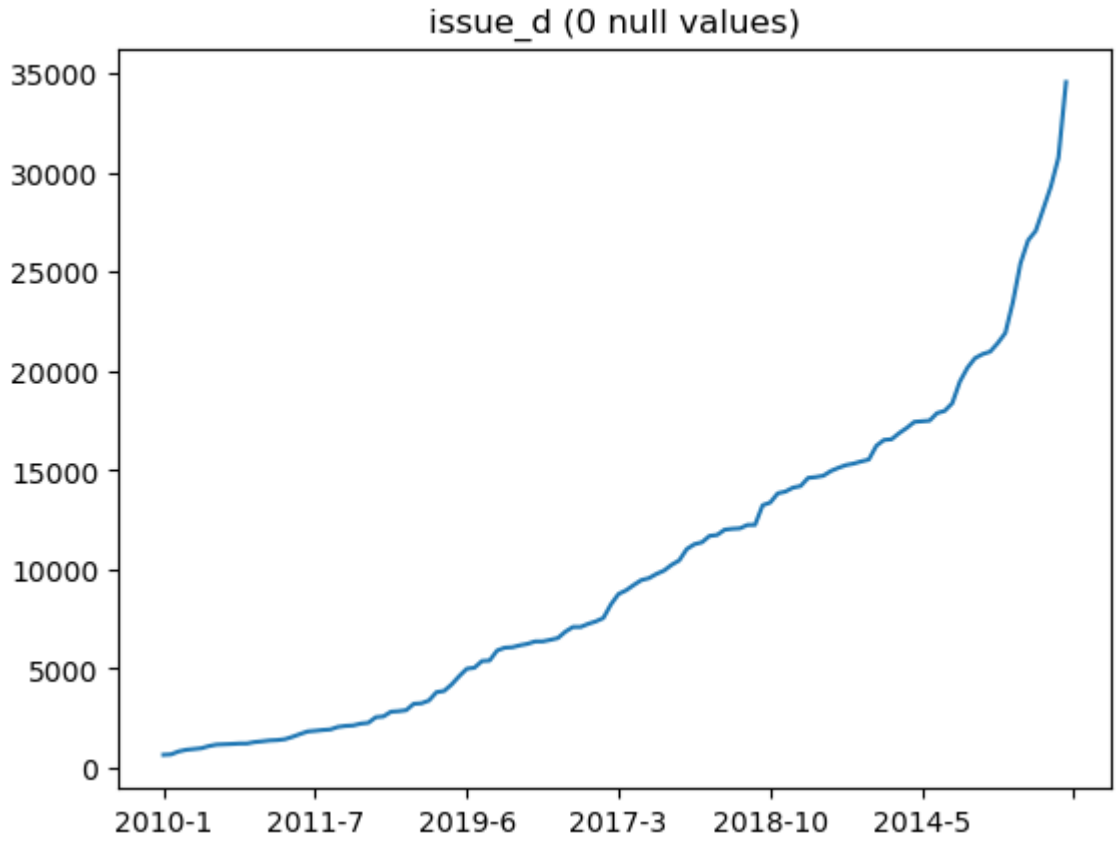


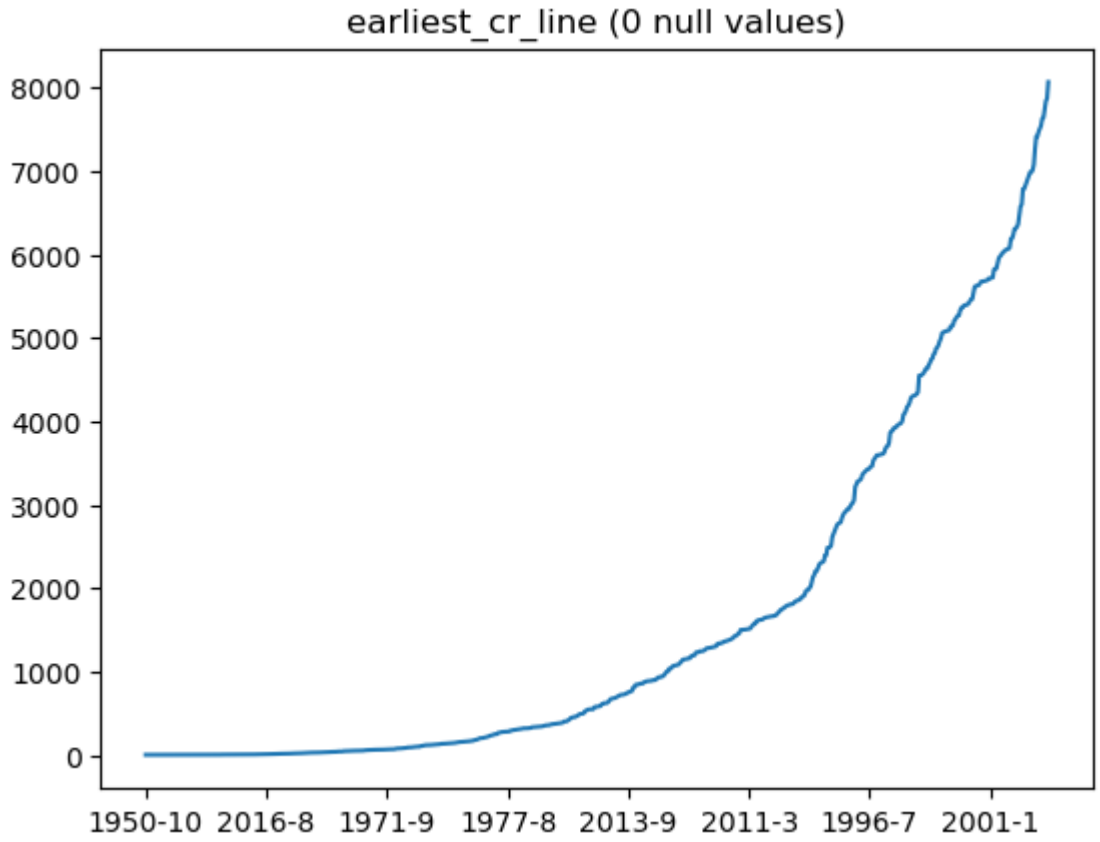

```
C:\Users\panjw\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

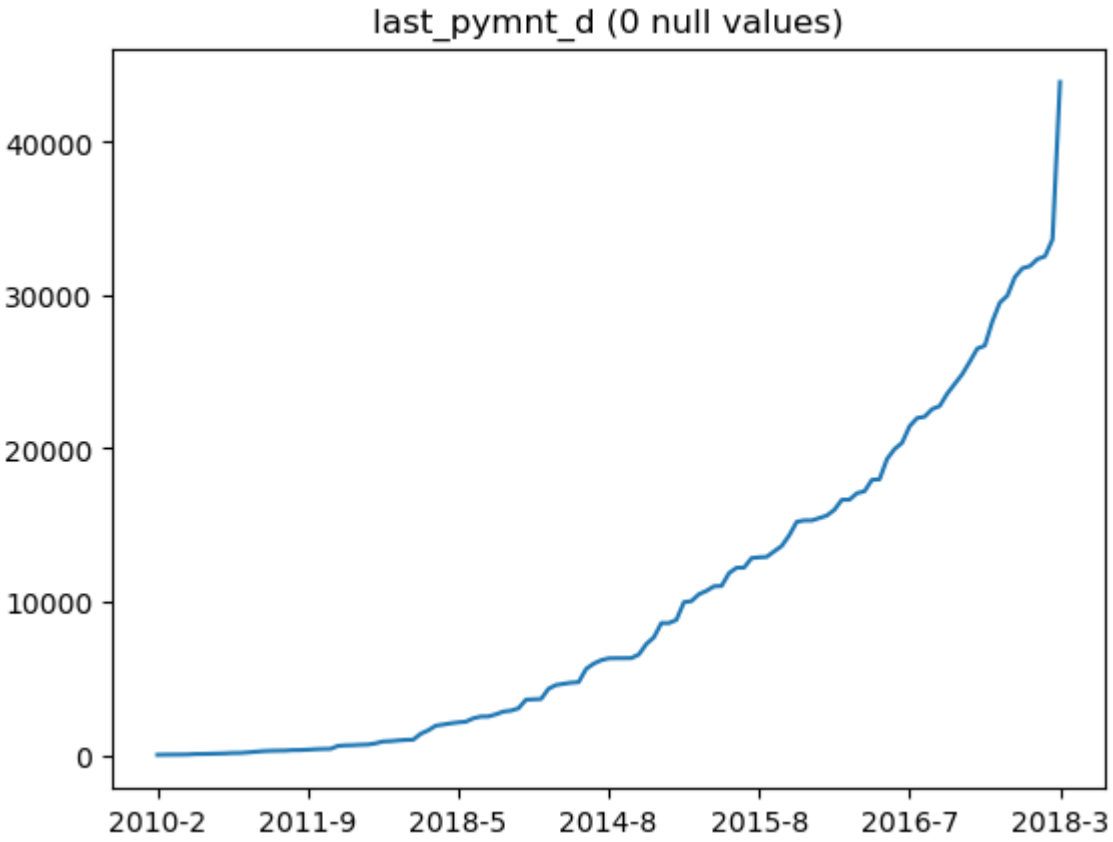

2.268087396869921
2.0931067218782156
2.090289551663506
0.0
0.5
1.0
1.5
2.0
ret_INTb

```
Column name: term
Number of distinct values: 2
 36 months    891199
 60 months    301307
Name: term, dtype: int64


Column name: grade
Number of distinct values: 7
B    336662
C    332342
A    211754
D    191040
E     84575
F     28397
G      7736
Name: grade, dtype: int64


Column name: emp_length
Number of distinct values: 11
10+ years    387212
2 years      107807
< 1 year      97403
3 years       95656
1 year        78618
5 years       75610
4 years       72118
6 years       56778
7 years       53380
8 years       52711
9 years       42956
Name: emp_length, dtype: int64


Column name: home_ownership
Number of distinct values: 6
MORTGAGE    592790
RENT        471298
OWN         127588
ANY            736
OTHER           50
NONE            44
Name: home_ownership, dtype: int64
```

```
Column name: verification_status
Number of distinct values: 3
Source Verified    448168
Not Verified       383056
Verified           361282
Name: verification_status, dtype: int64


Column name: loan_status
Number of distinct values: 8
Fully Paid                                            894335
Charged Off                                           233930
Late (31-120 days)                                     33726
In Grace Period                                        19557
Late (16-30 days)                                       9028
Default                                                  937
Does not meet the credit policy. Status:Fully Paid       730
Does not meet the credit policy. Status:Charged Off      263
Name: loan_status, dtype: int64


Column name: purpose
Number of distinct values: 14
debt_consolidation    693374
credit_card           259903
home_improvement       76629
other                  69544
major_purchase         26042
medical                13672
small_business         13639
car                    12502
moving                  8432
vacation                8064
house                   7645
wedding                 2117
renewable_energy         831
educational              112
Name: purpose, dtype: int64
```


issue_d (0 null values)
35000
30000
25000
20000
15000
10000
5000
0
2010-1
2011-7
2019-6
2017-3
2018-10
2014-5


earliest_cr_line (0 null values)
8000
7000
6000
5000
4000
3000
2000
1000
0
1950-10
2016-8
1971-9
1977-8
2013-9
2011-3
1996-7
2001-1

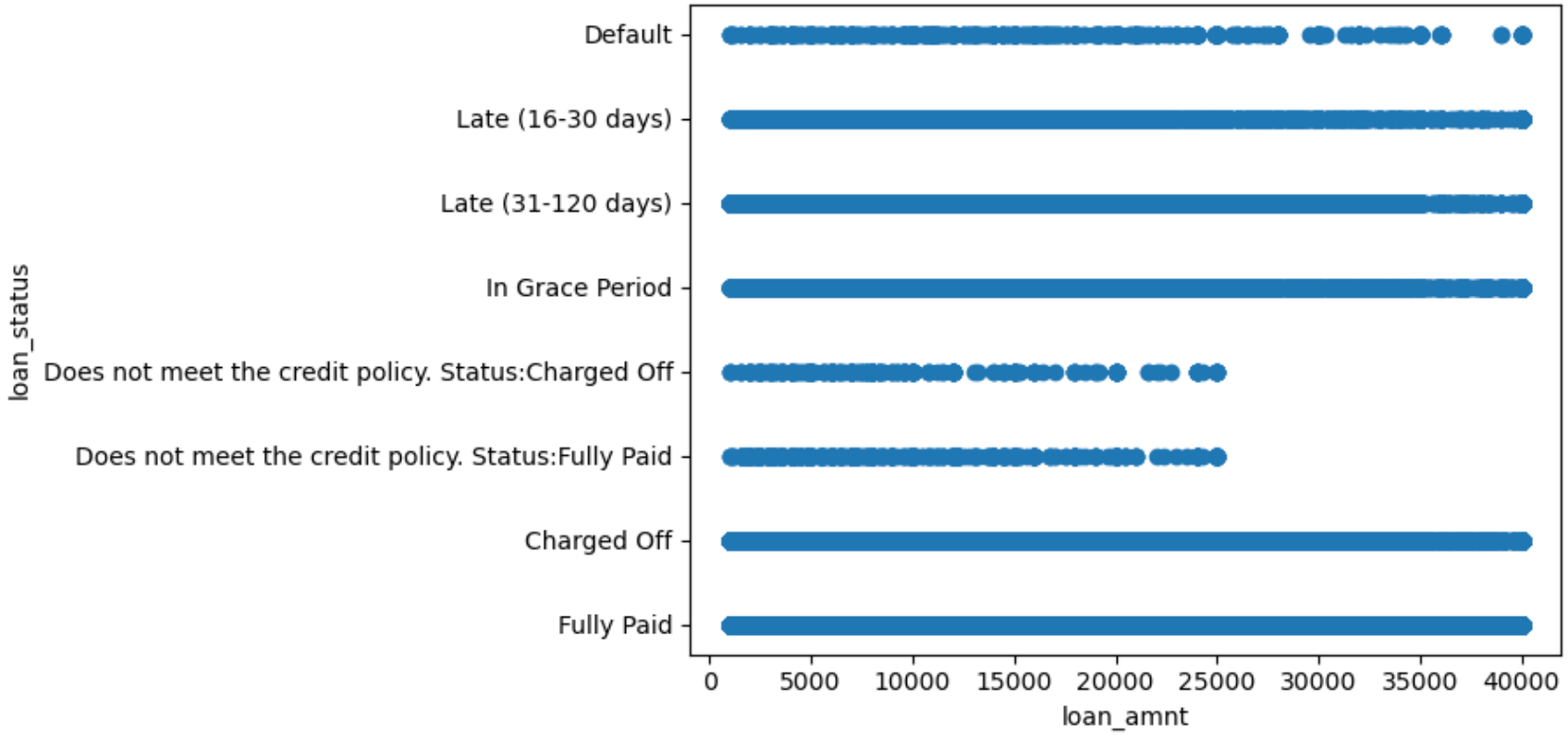


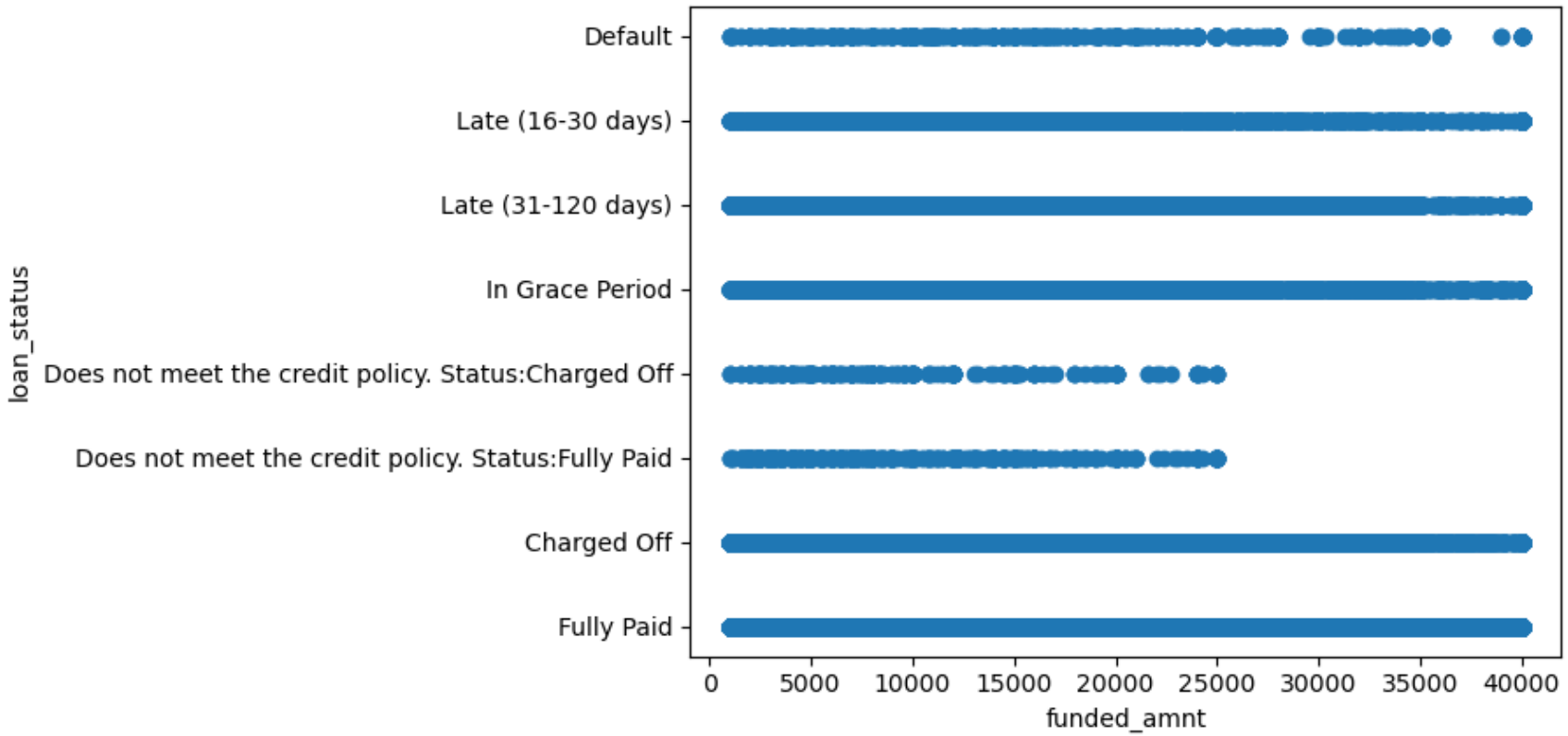
In [33]:
```
# Visualize the feature correlations
    # You can compute the correlation among features and display a heat-map of the matrix
    # OR use sns scatter or pairplot
import seaborn as sns
fig, ax = plt.subplots(figsize=(13, 13))
corr=final_data.corr()
sns.heatmap(corr,
        xticklabels=corr.columns,
        yticklabels=corr.columns,annot=True,ax=ax)
```

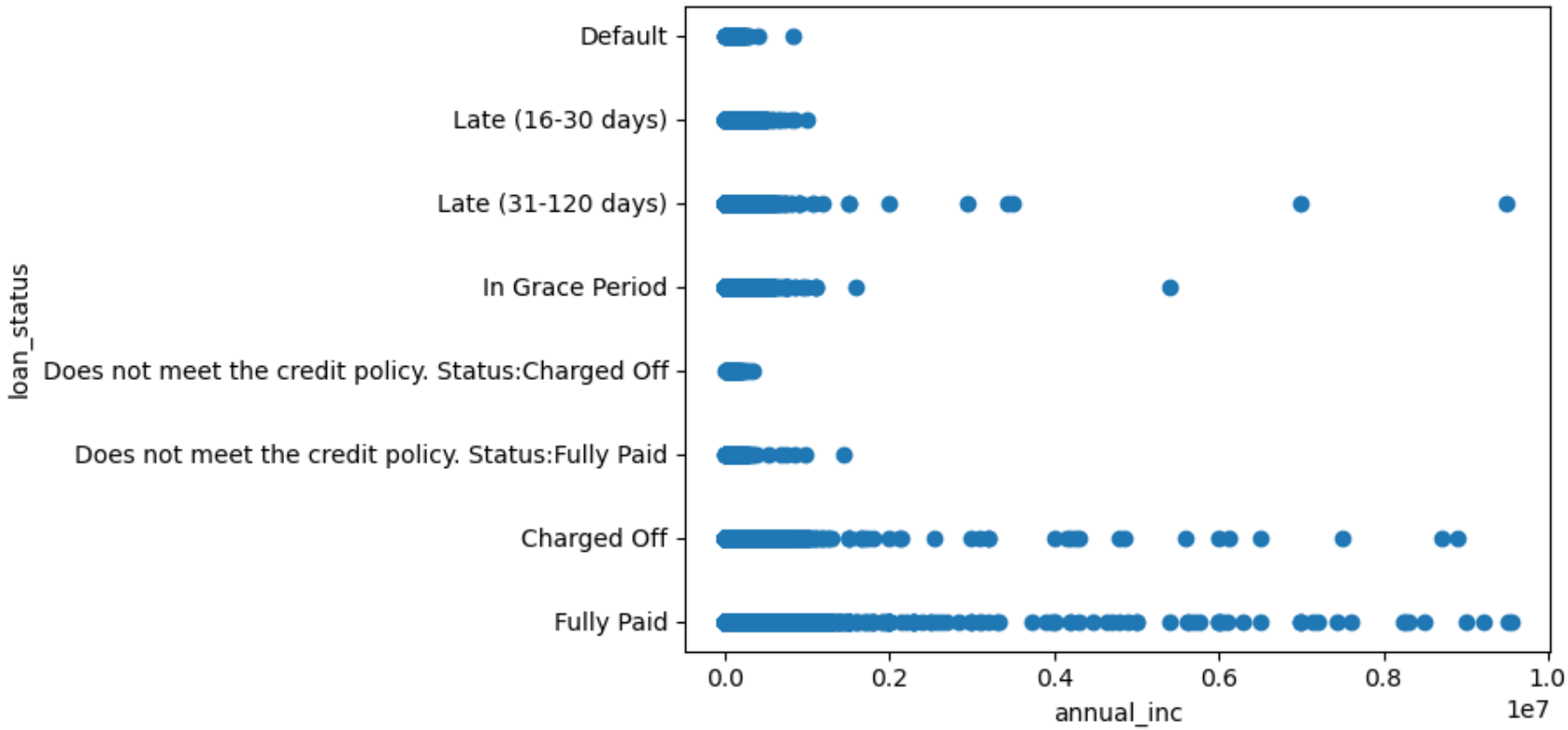
Out[33]:
```
<AxesSubplot:>
```

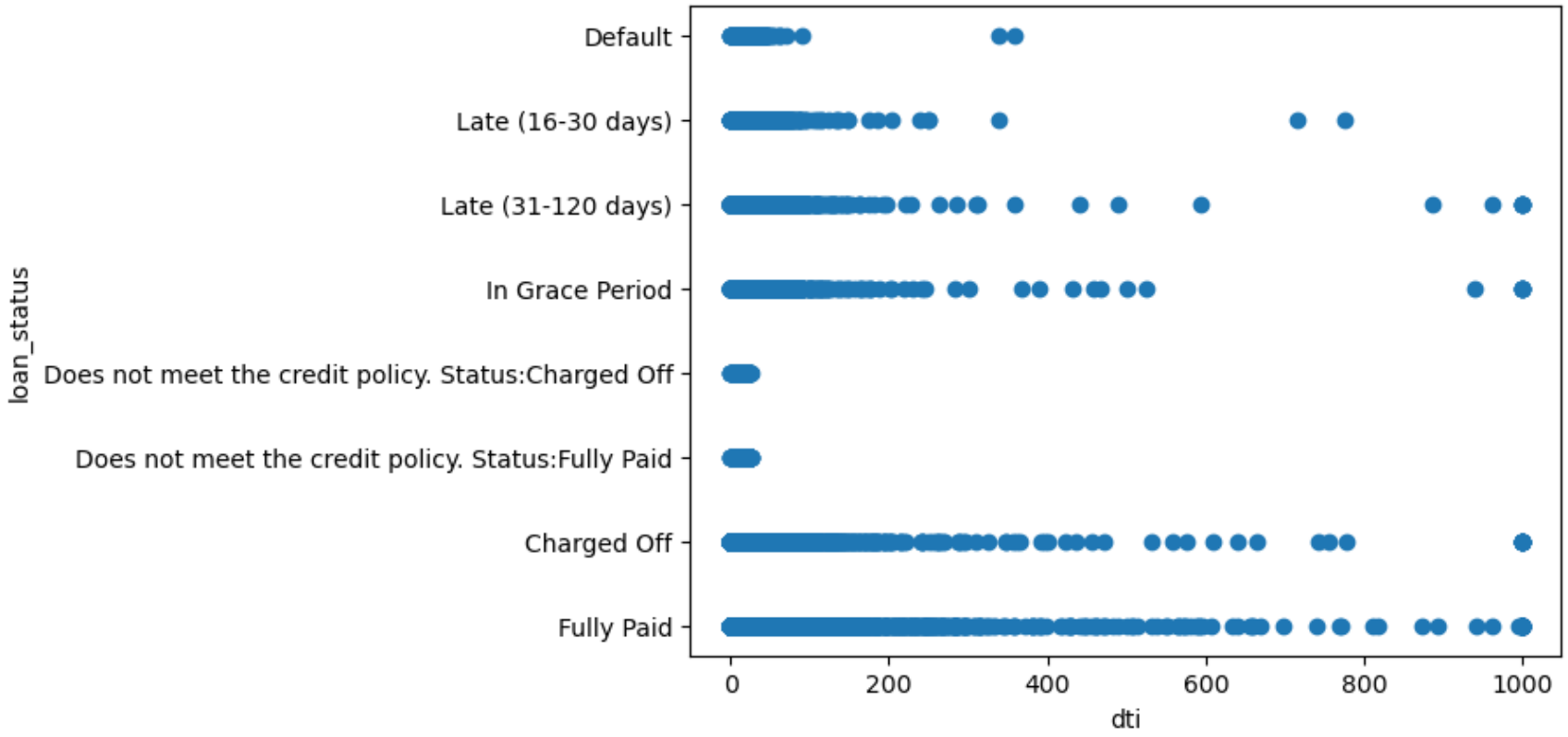

loan_amnt
funded_amnt
int_rate
annual_inc
dti
delinq_2yrs
open_acc
pub_rec
fico_range_high
fico_range_low
revol_bal
revol_util
total_pymnt
recoveries
loan_length
term_num
ret_PESS
ret_OPT
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

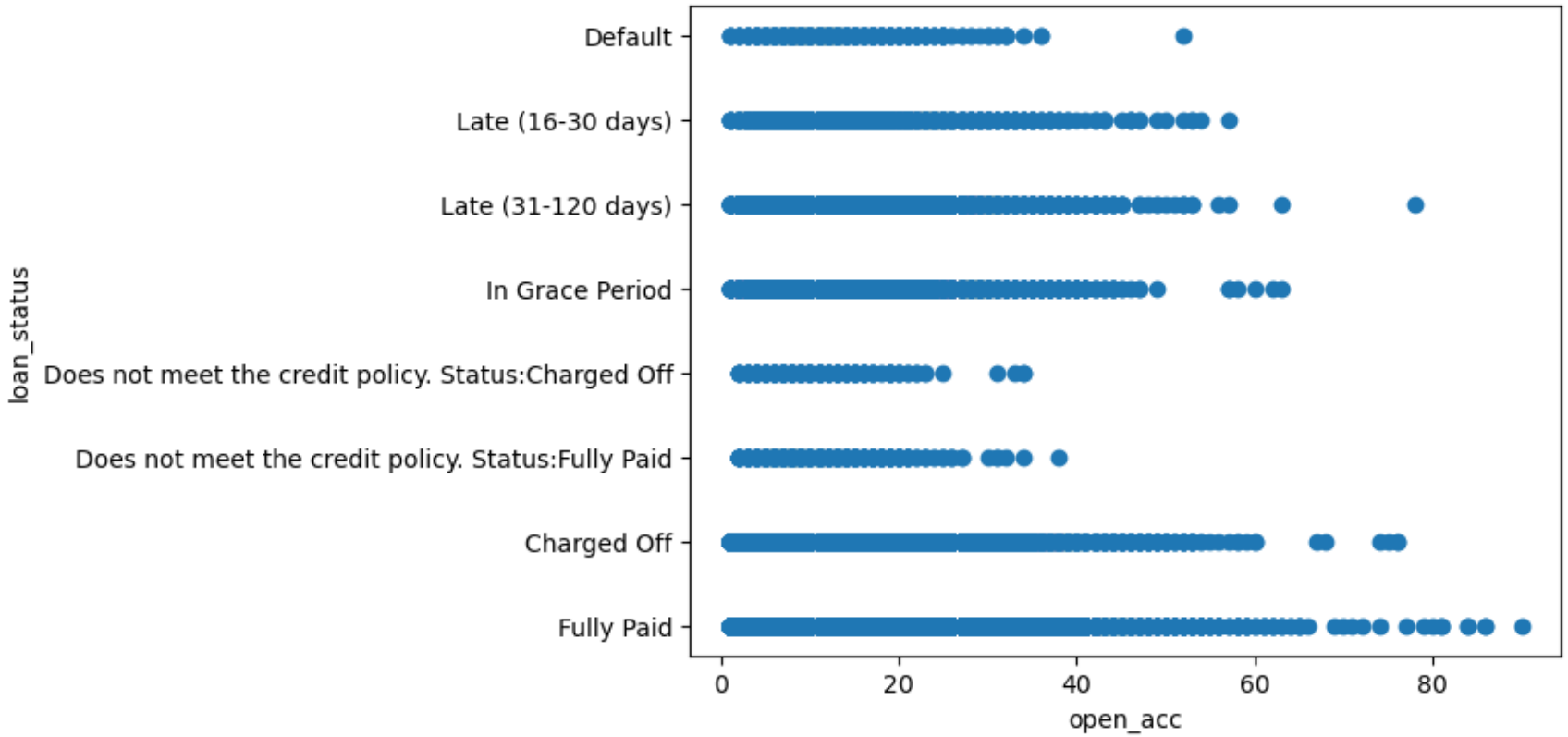
In [34]:
```
# Visualize relation between loan status and features
# sns pairplot or scatter plot. Refer to recitations
for col in float_cols + perc_cols + ret_cols:
    plt.scatter(x=final_data[col],y=final_data['loan_status'])
    plt.xlabel(col)
    plt.ylabel('loan_status')
    plt.show()
```

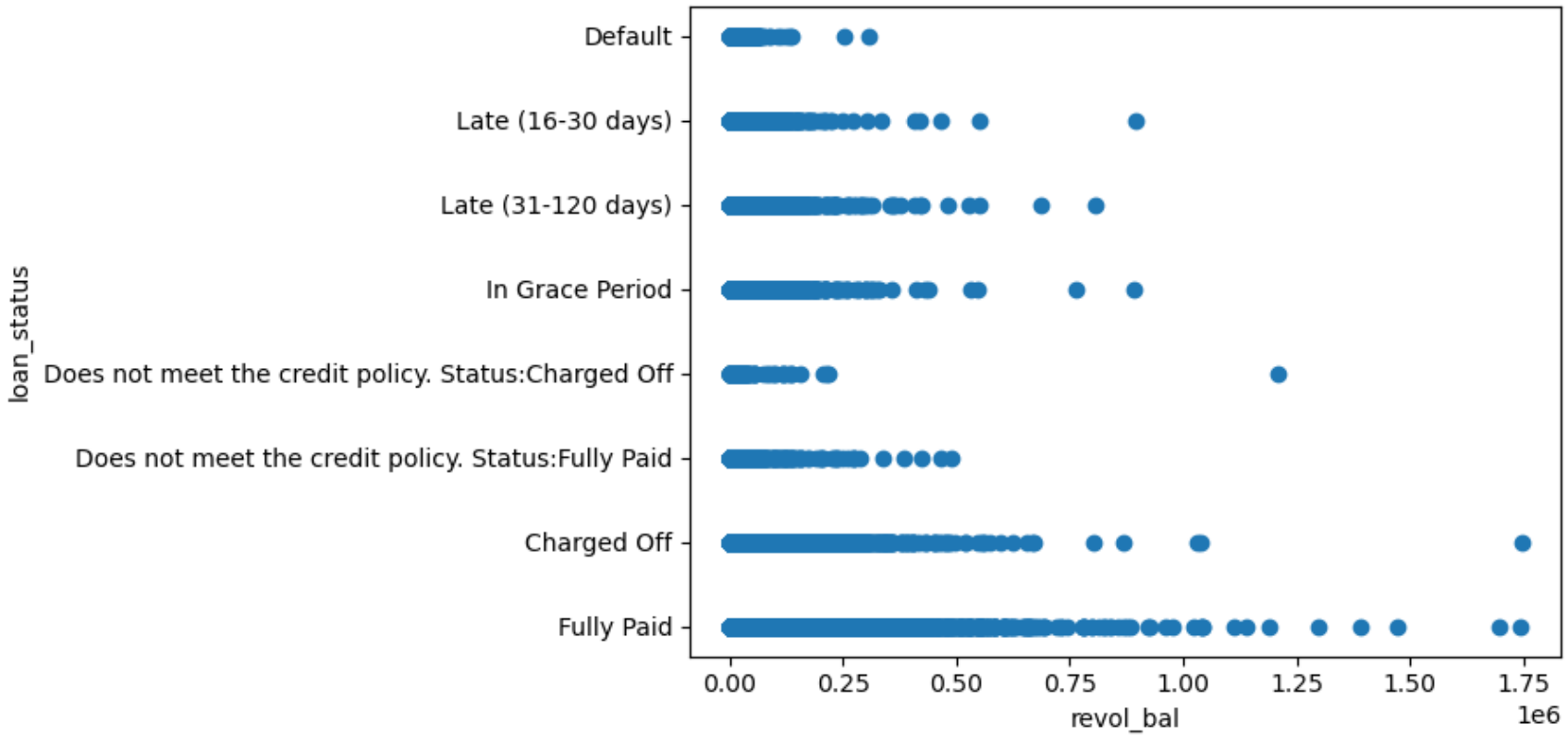


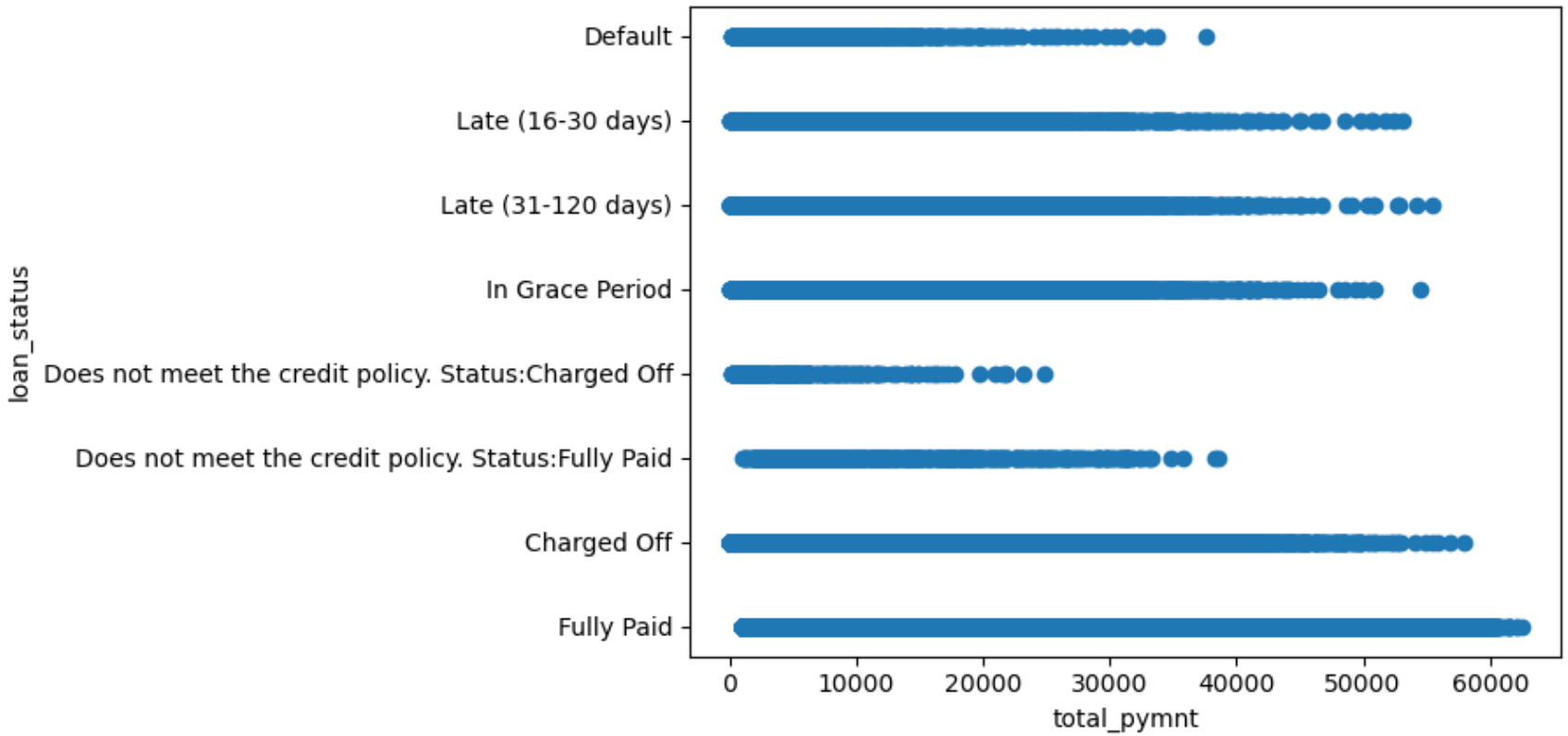

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
5000
10000
15000
20000
25000
30000
35000
40000
funded_amnt

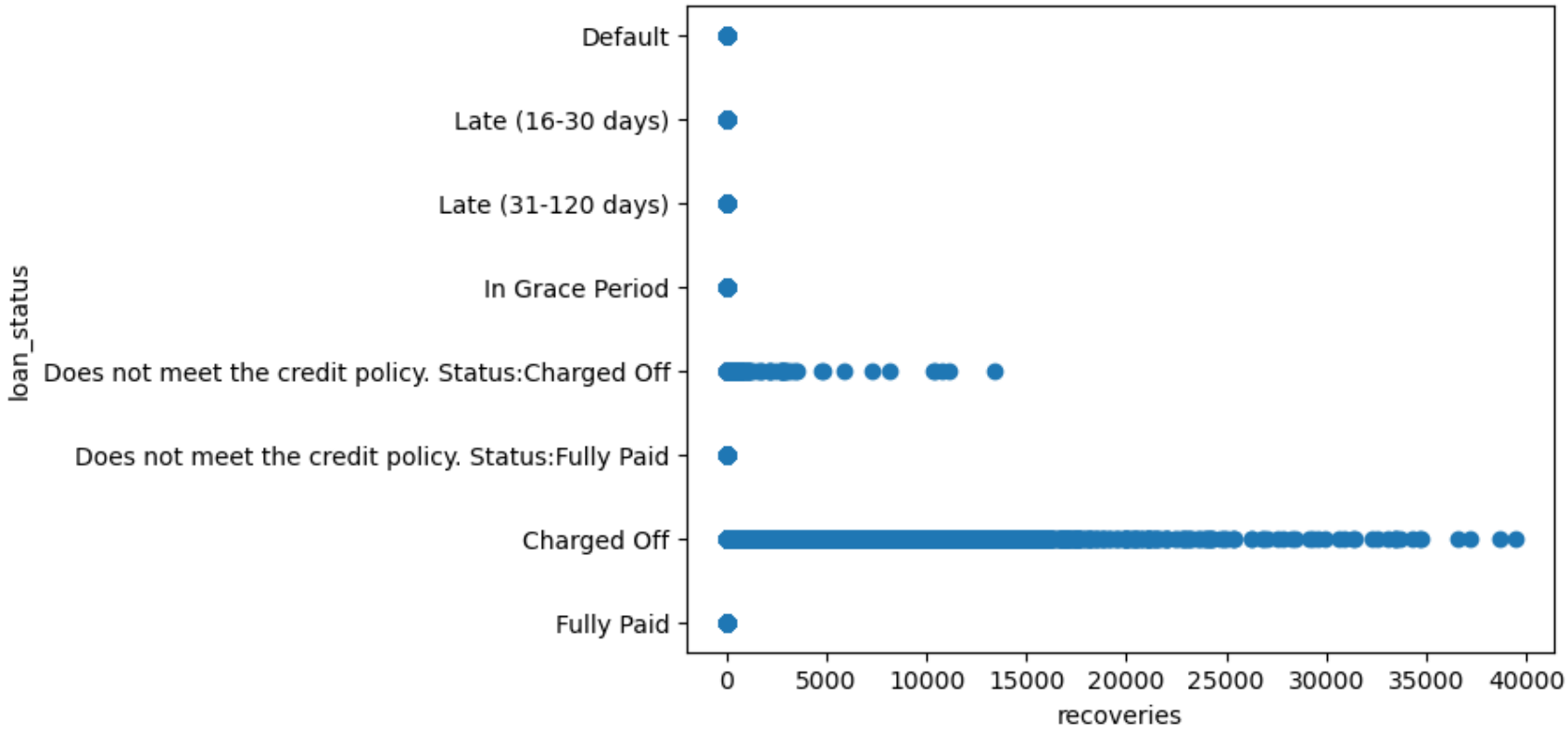

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0.0
0.2
0.4
0.6
0.8
1.0
annual_inc
1e7

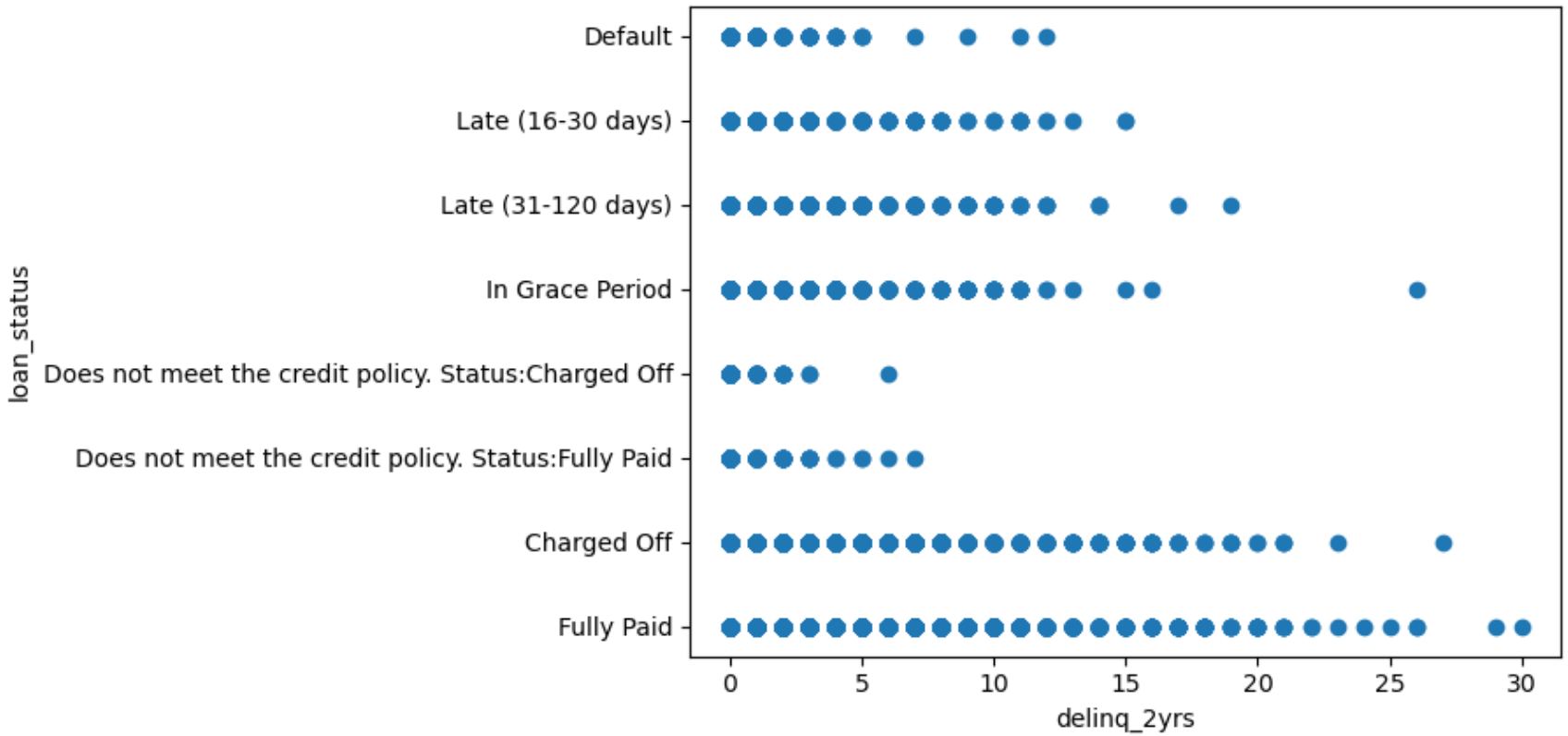

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
200
400
600
800
1000
dti

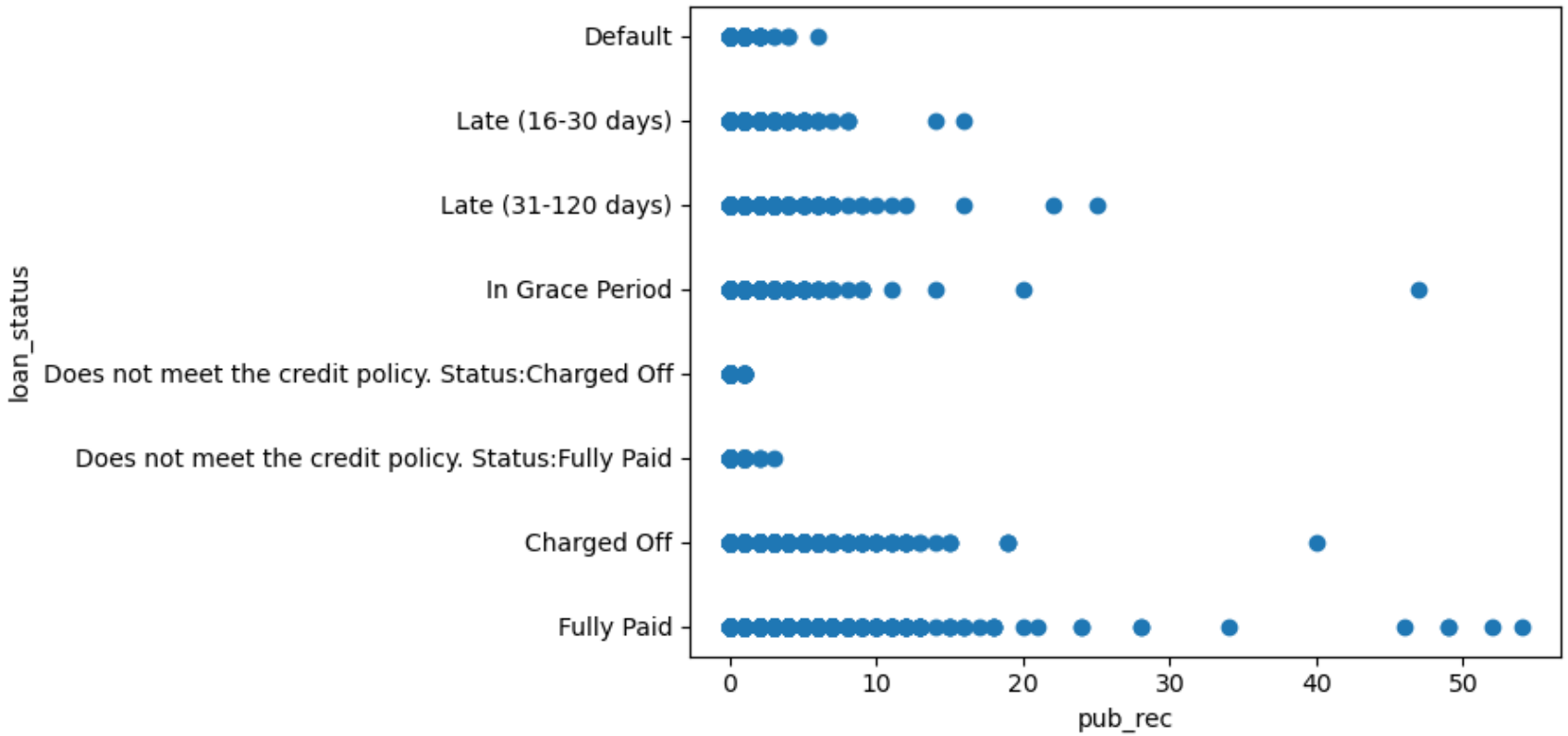

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
20
40
60
80
open_acc

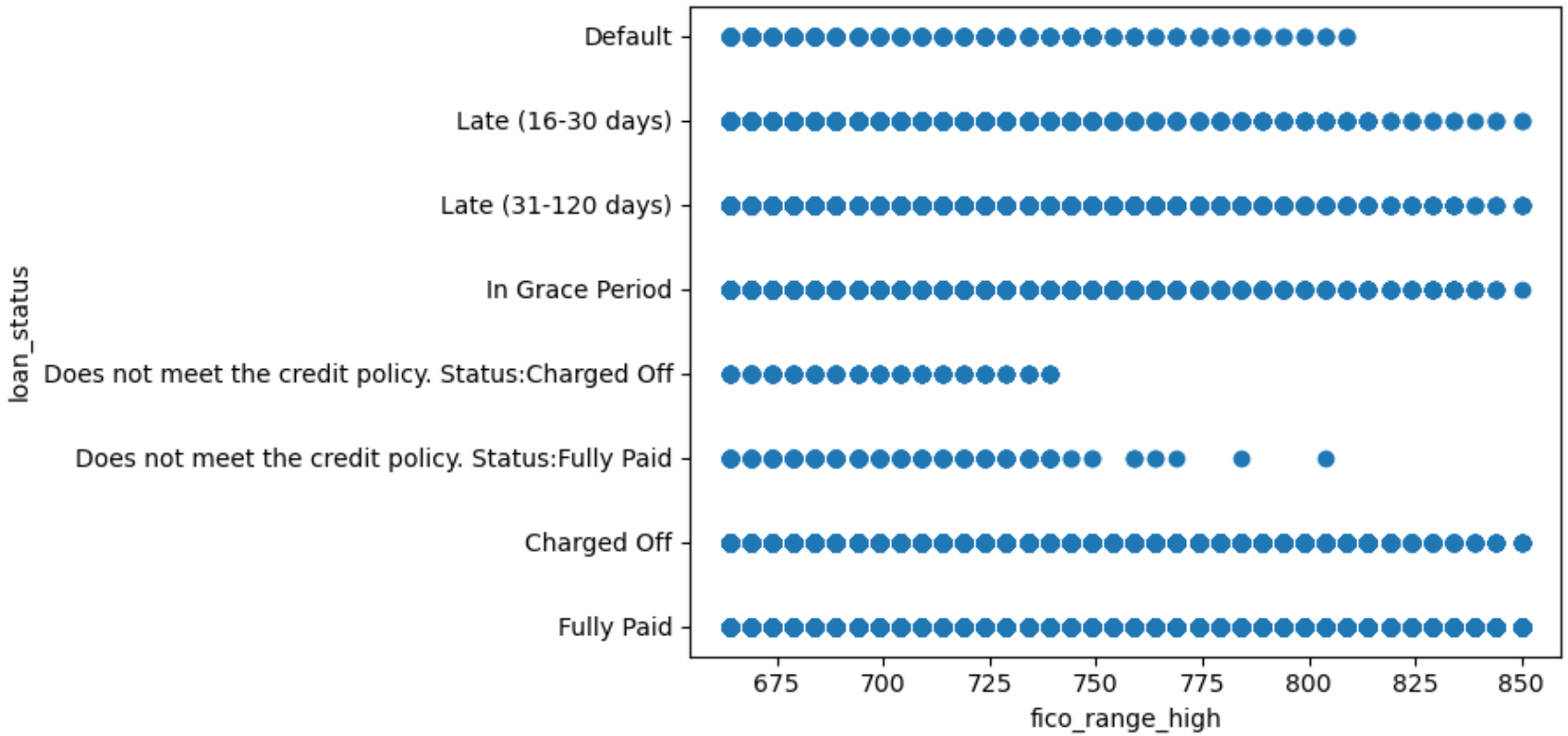

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6
revol_bal

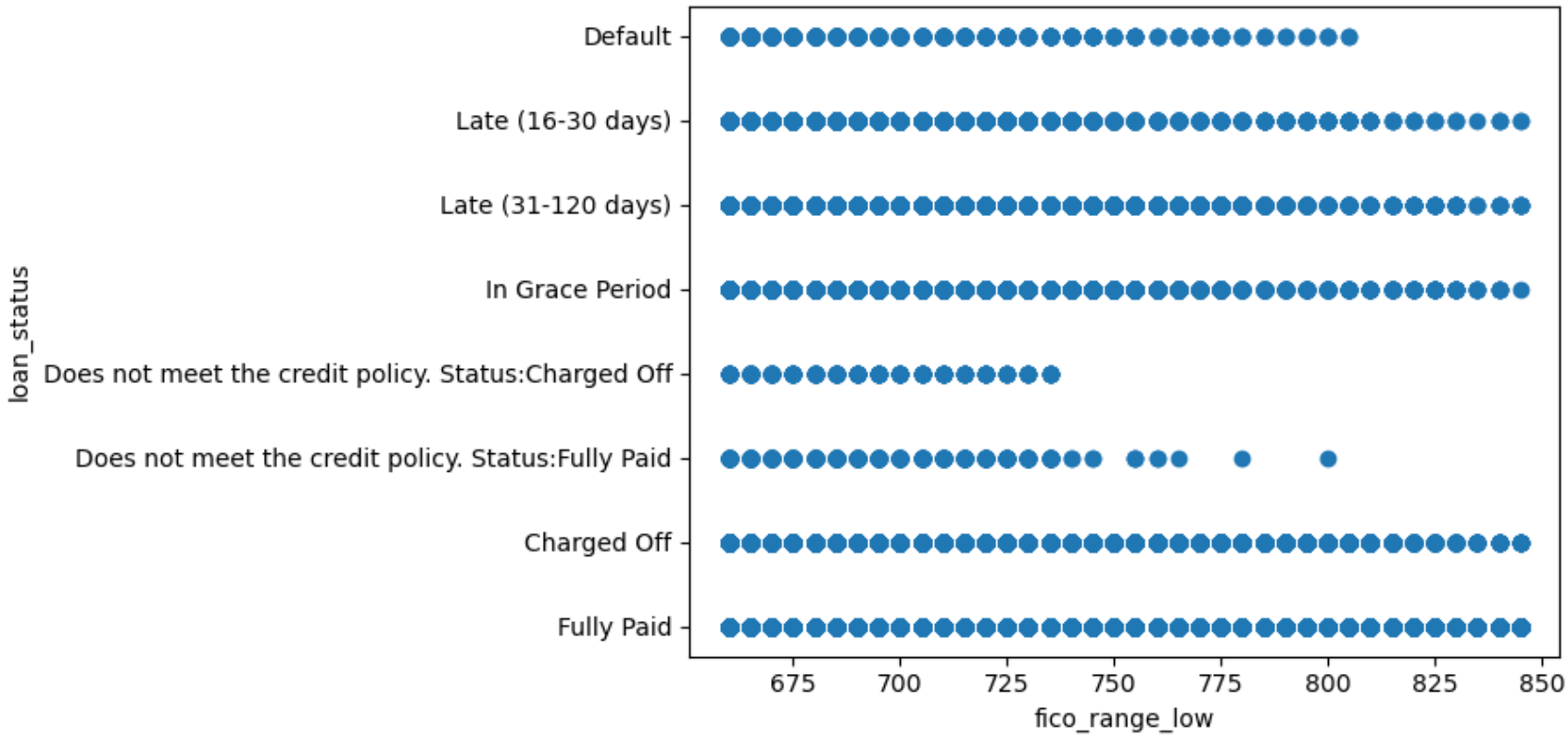

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
10000
20000
30000
40000
50000
60000
total_pymnt

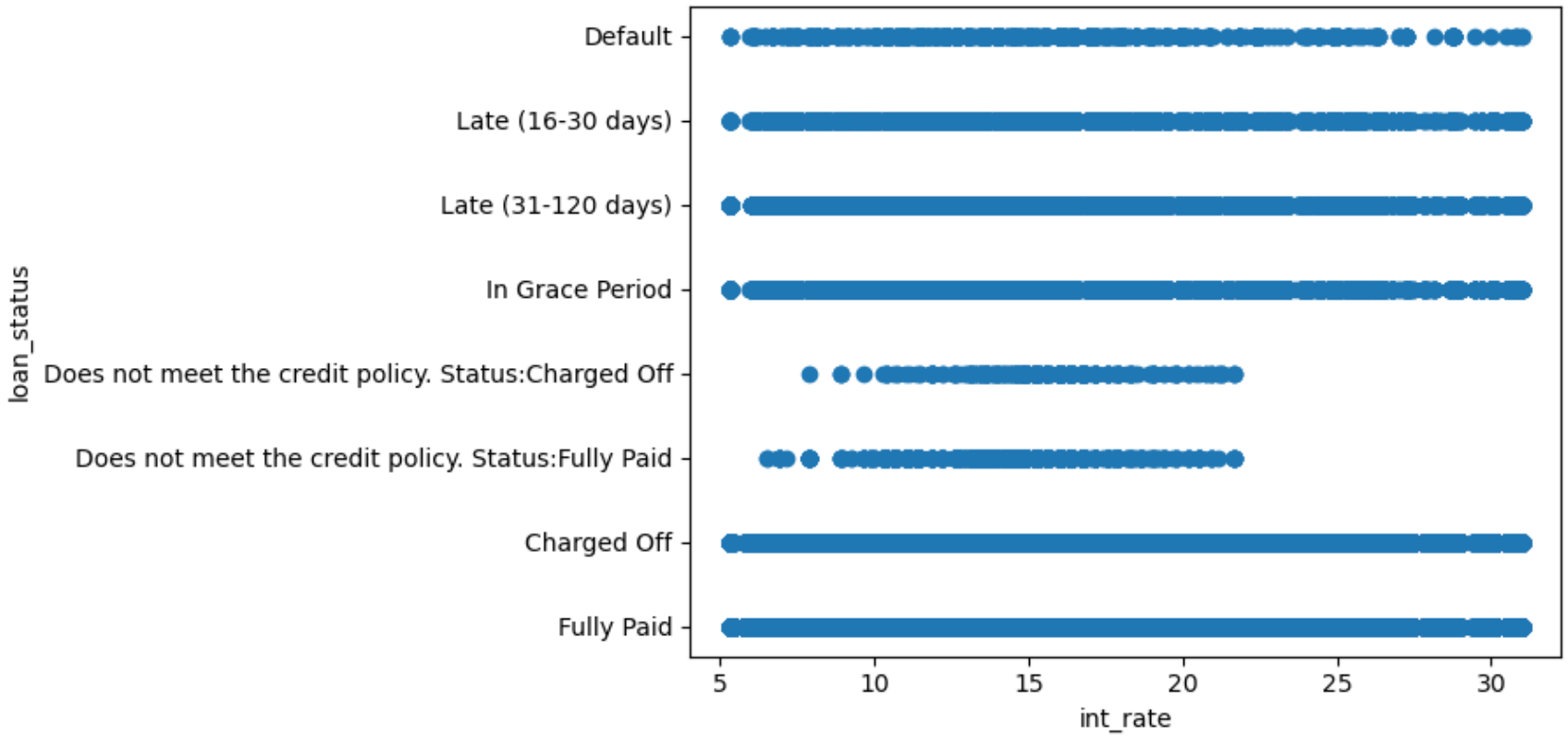

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
5000
10000
15000
20000
25000
30000
35000
40000
recoveries

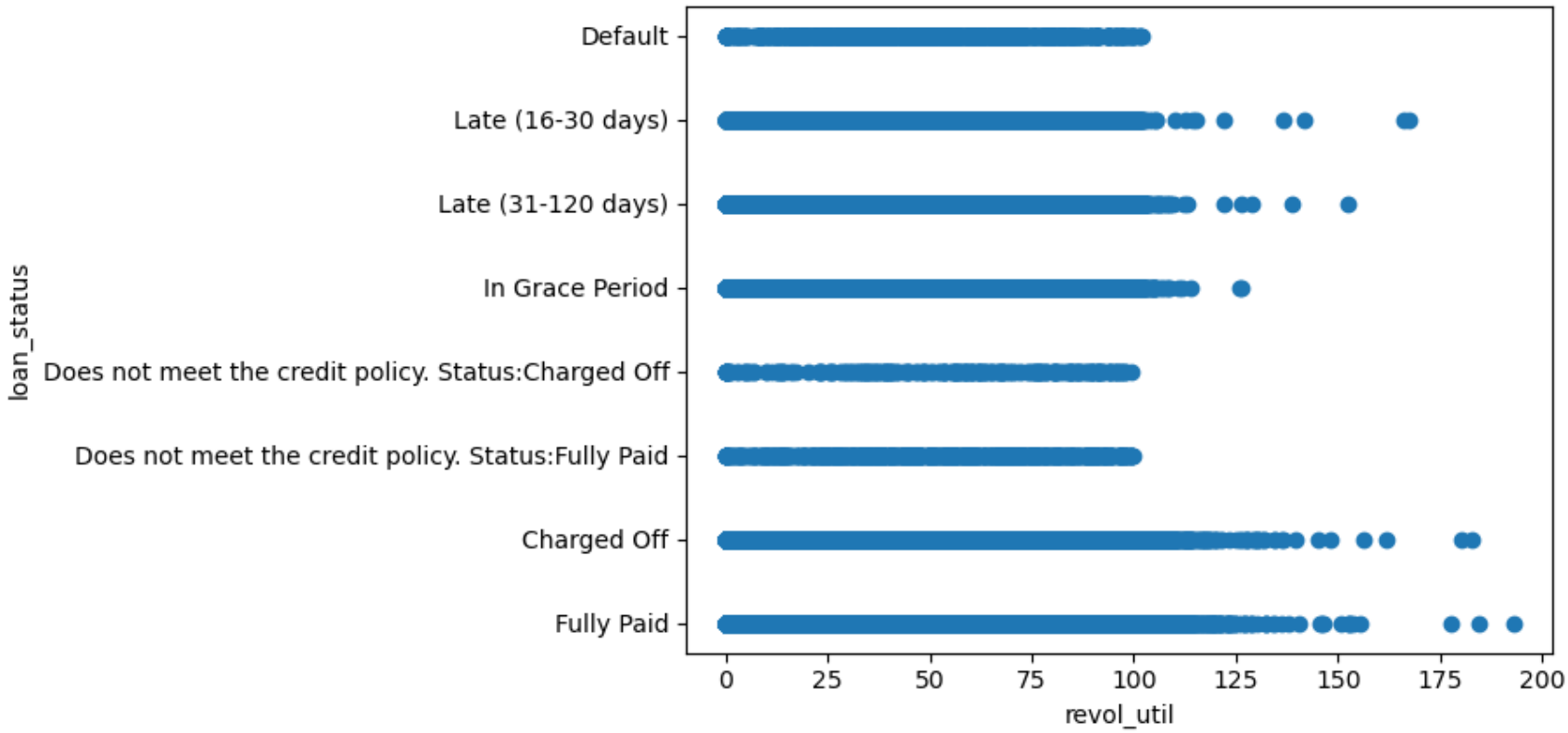

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
5
10
15
20
25
30
delinq_2yrs

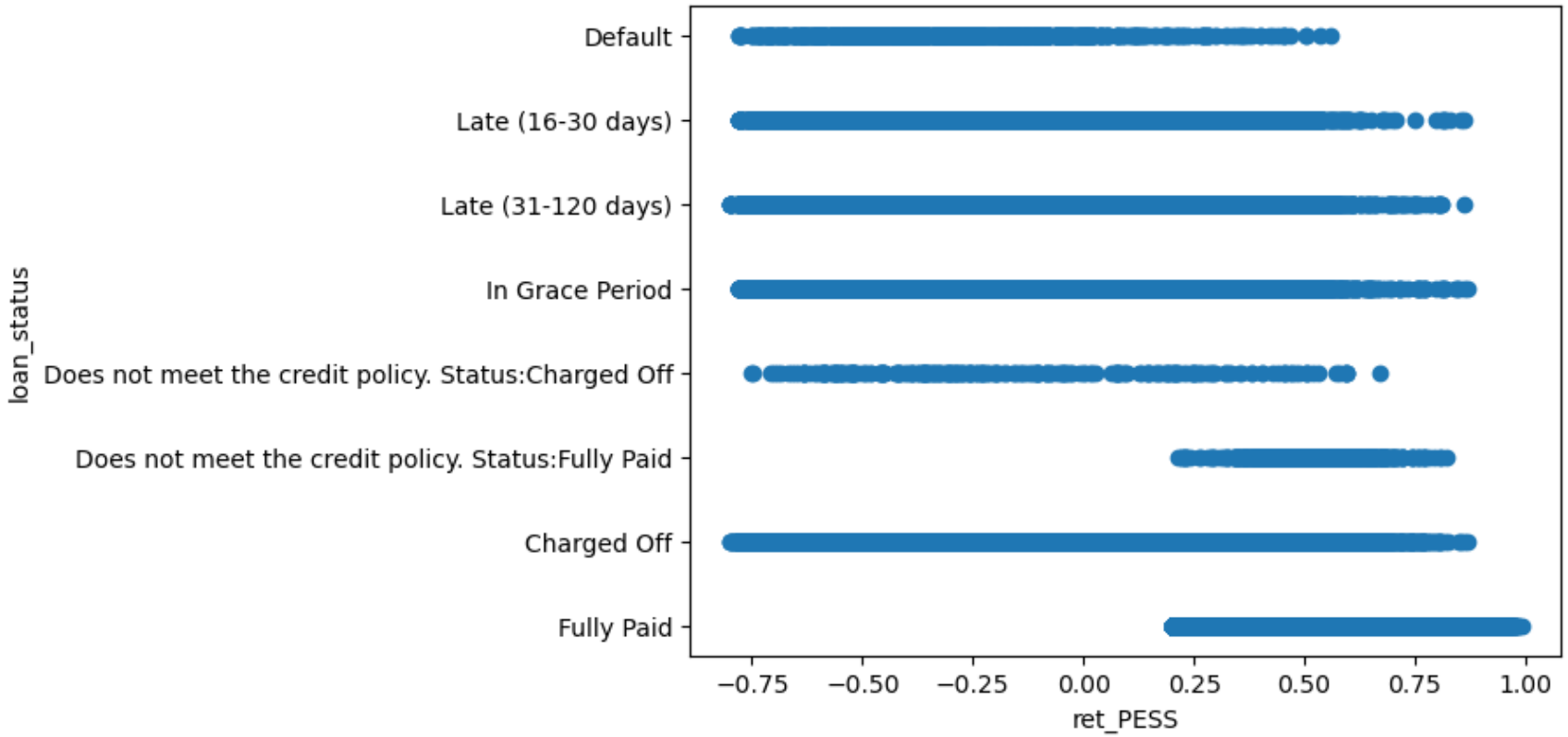

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
10
20
30
40
50
pub_rec

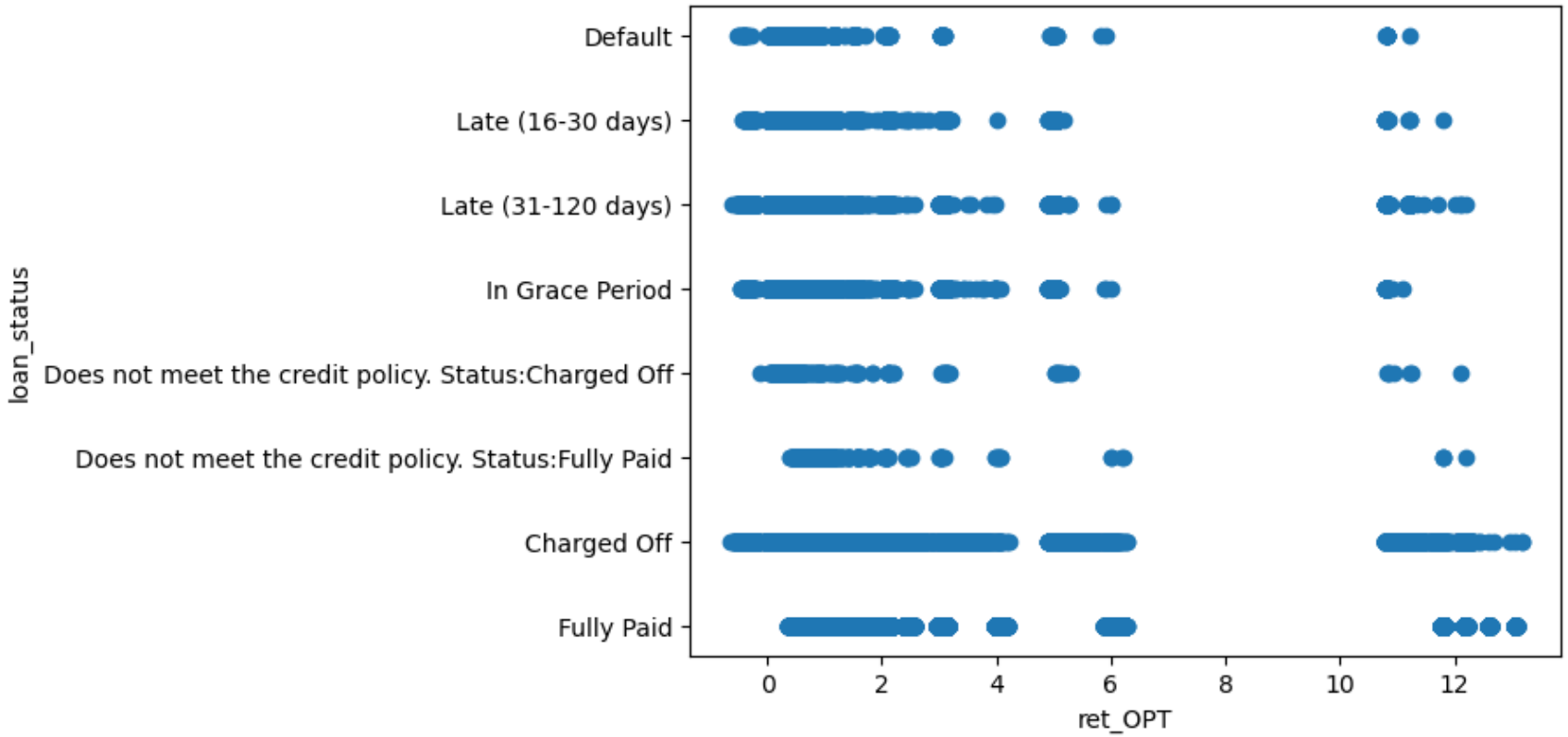

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
675
700
725
750
775
800
825
850
fico_range_high

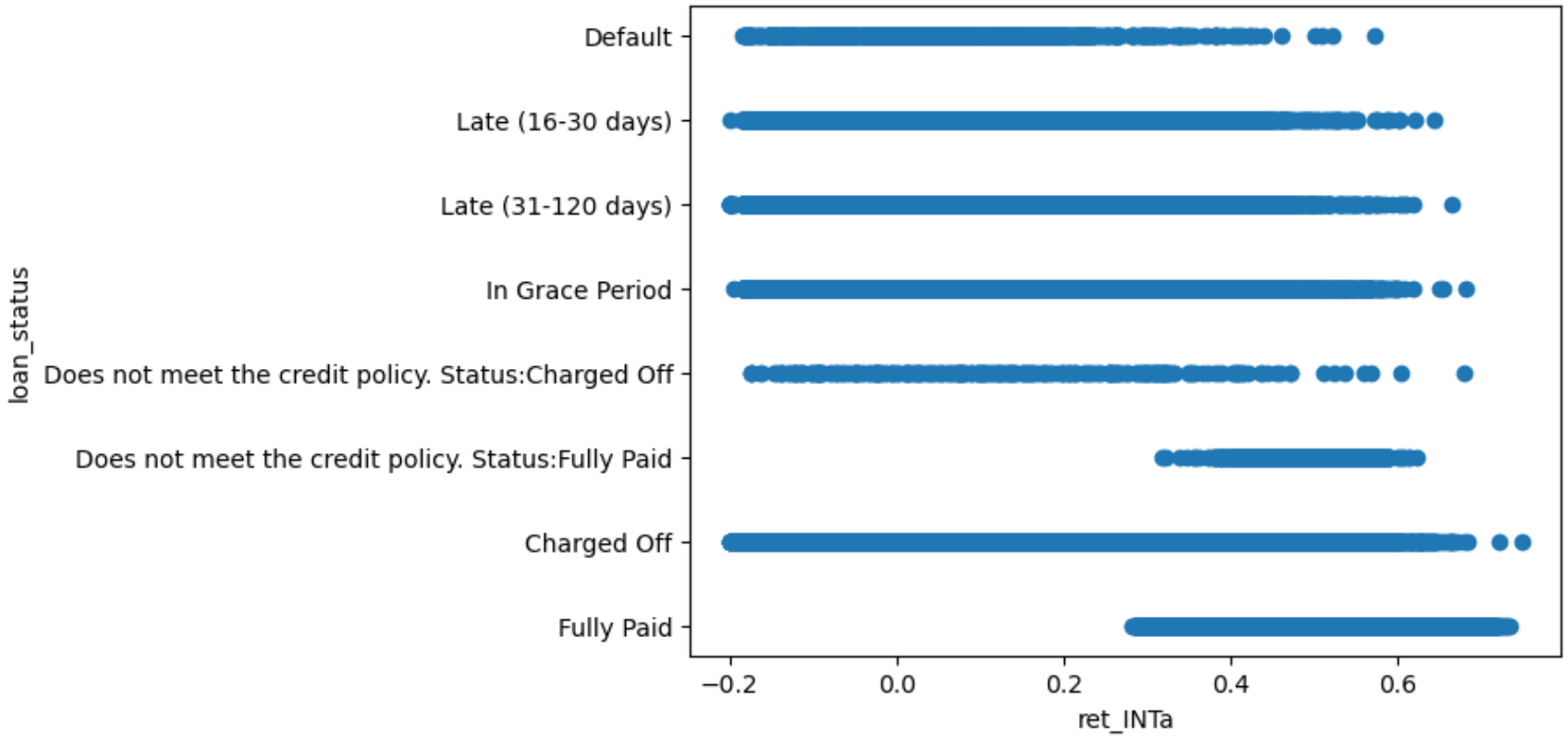

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
675
700
725
750
775
800
825
850
fico_range_low

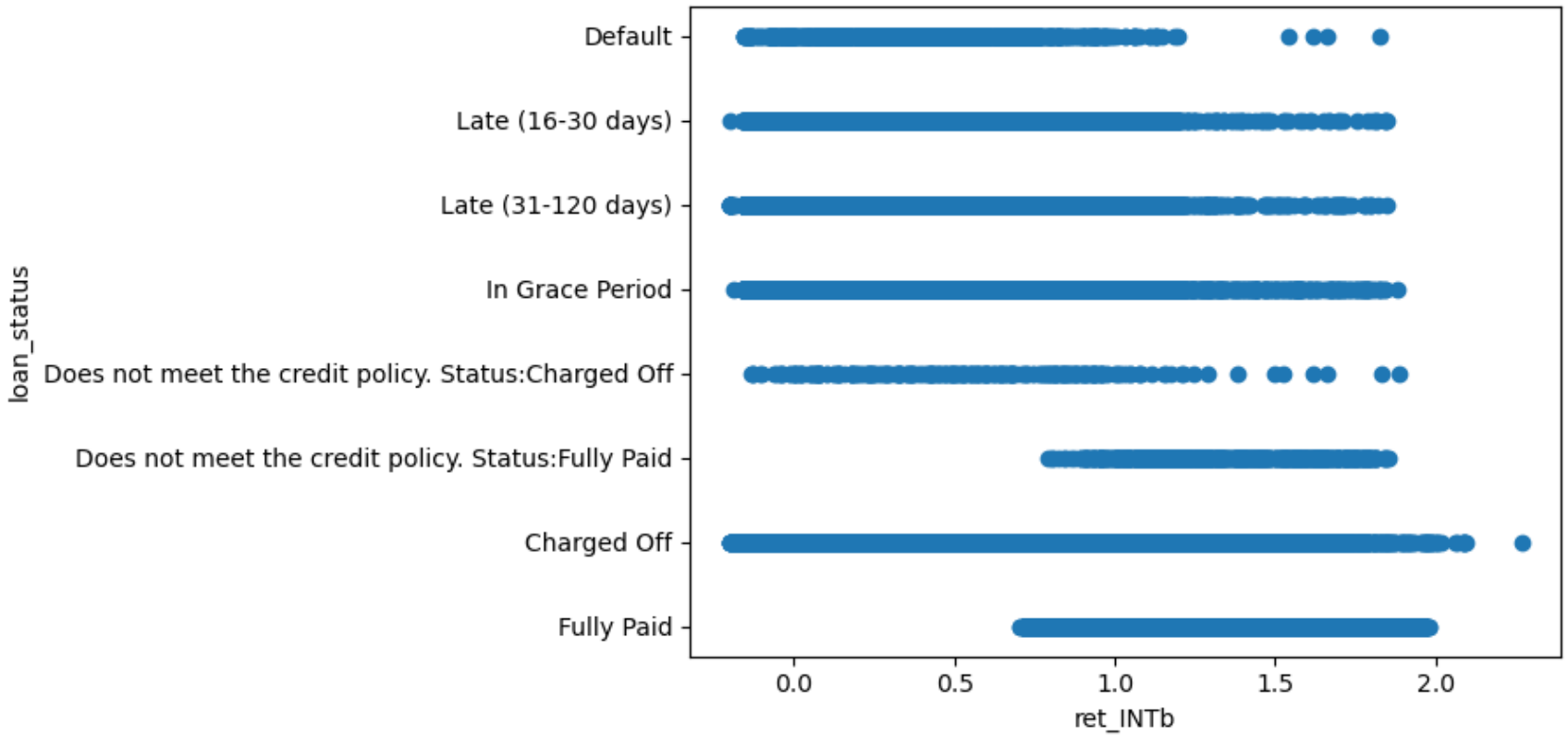

Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
5
10
15
20
25
30
int_rate


Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
25
50
75
100
125
150
175
200
revol_util


Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
ret_PESS


Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
0
2
4
6
8
10
12
ret_OPT


Default
Late (16-30 days)
Late (31-120 days)
In Grace Period
Does not meet the credit policy. Status:Charged Off
Does not meet the credit policy. Status:Fully Paid
Charged Off
Fully Paid
loan_status
−0.2
0.0
0.2
0.4
0.6
ret_INTa



What do you observe after removing the outliers?

Data Exploration

Solution to Q.7 from the handout

In [35]:
```
# Find the percentage of loans by grade, the default by grade,
# and the return of each grade
perc_by_grade = (final_data.grade.value_counts()*100/len(final_data)).sort_index()

default_by_grade = final_data.groupby("grade").apply(lambda x : (x.loan_status != "Fully Paid").sum()*100/len(x) )
ret_by_grade_OPT = final_data.groupby('grade').agg({'ret_OPT':'mean'}) # average return for M2-Optimistic for each loan
ret_by_grade_PESS = final_data.groupby('grade').agg({'ret_PESS':'mean'}) # average return for M1-Pessimistic for each l
ret_by_grade_INTa = final_data.groupby('grade').agg({'ret_INTa':'mean'}) # average return for M3
ret_by_grade_INTb = final_data.groupby('grade').agg({'ret_INTb':'mean'}) # average return for M3
int_rate_by_grade = final_data.groupby('grade').agg({'int_rate':'mean'}) # average interest rate for each grade
```

```
combined = pd.DataFrame(perc_by_grade)
combined.columns = ['perc_of_loans']
combined['perc_default'] = default_by_grade
combined['avg_int_rate'] = int_rate_by_grade
combined['return_OPT'] = ret_by_grade_OPT
combined['return_PESS'] = ret_by_grade_PESS
combined['return_INTa'] = ret_by_grade_INTa
combined['return_INTb'] = ret_by_grade_INTb

combined
```

Out[35]:

	perc_of_loans	perc_default	avg_int_rate	return_OPT	return_PESS	return_INTa	return_INTb
A	17.757060	9.208327	7.221118	1.394685	0.347594	0.445251	1.316969
B	28.231472	17.785197	10.882361	1.369832	0.321003	0.432577	1.280858
C	27.869210	28.227549	14.231611	1.473430	0.258236	0.409517	1.233966
D	16.020045	36.647822	18.109706	1.582821	0.211456	0.388937	1.187303
E	7.092208	43.506946	21.351506	1.521320	0.169627	0.376713	1.152735
F	2.381288	49.308025	24.882461	1.569195	0.130559	0.363299	1.118552
G	0.648718	53.955533	27.473188	1.858501	0.063813	0.337867	1.075296

Based on the output of previous cell, write down your answers to Q.7 from the handout.

Save a Pickle

In [36]:
```
# Remove the "total_pymnt" and "recoveries" from the list of continuous features
continuous_features = list(set(float_cols)-set(['total_pymnt','recoveries']))
```

Why did we remove `total_pymt` and `recoveries` from the data for the task of predicting whether to give loan or not, although these are highly predictive features?

In [37]:
```
# save the prepared data for modeling in next Phase.
pickle.dump( [final_data, discrete_features, continuous_features, ret_cols], open(pickle_file, "wb") )
```

In [ ]:
```

```

Q7)

(i) What percentage of loans are in each grade

A : 17.7
B : 28.23
C: 27.86
D: 16.02
E: 7.09

	perc_of_loans	perc_default	avg_int_rate	return_OPT	return_PESS	return_INTa	return_INTb
A	17.757060	9.208327	7.221118	1.394685	0.347594	0.445251	1.316969
B	28.231472	17.785197	10.882361	1.369832	0.321003	0.432577	1.280858
C	27.869210	28.227549	14.231611	1.473430	0.258236	0.409517	1.233966
D	16.020045	36.647822	18.109706	1.582821	0.211456	0.388937	1.187303
E	7.092208	43.506946	21.351506	1.521320	0.169627	0.376713	1.152735
F	2.381288	49.308025	24.882461	1.569195	0.130559	0.363299	1.118552
G	0.648718	53.955533	27.473188	1.858501	0.063813	0.337867	1.075296

F: 2.38
G: 0.64

(ii) What is the default rate in each grade? How do you interpret those numbers?

A : 9.2
B : 17.78
C: 28.22
D: 36.64
E: 43.50
F: 49
G: 27.4

All the loans that belong to grade A, 9.2% of them had defaulted. Similarly, for the others, the numbers represent the percentage of loans which defaulted.

(iii) What is the average interest rate in each grade? How do you interpret those numbers?

A : 7.22
B : 10.88
C: 14.23
D: 18.1
E: 21.35

F: 24.88
G: 27.47

The average interest rate of grade A is 7.22%. Similarly, for the others, the numbers represent the average rate of interest of that particular grade.

(iv) What is the average percentage (annual) return per grade (as calculated using the three methods in part 6.)? (Assume two different yearly rates for M3: (i = 2.3) and (i = 4.0))

M3: (i = 2.3)

A : 0.44
B : 0.43
C: 0.40
D: 0.38
E: 0.37
F: 0.36
G: 0.33

M3: (i = 4.0)

A : 1.31
B : 1.28
C: 1.23
D: 1.18
E: 1.15
F: 1.11
G: 1.07

(v) Do these numbers surprise you? If you had to invest in one grade only, which loans would you invest in?

Based on the information provided, it appears that there is a significant increase in default rates between loan grades A and G, despite a relatively small increase in the average interest rate. As such, if one were to invest in a grade of loan, it may be wise to consider a medium risk, medium reward option such as grade B. While this would likely result in a lower return compared to higher risk loans, it would also offer a lower chance of default and potential loss of investment.