Loan Default Prediction

April 16, 2024

1 Loan Default Prediction

1.0.1 Run imports

```
[ ]: import warnings
from numba.core.errors import NumbaDeprecationWarning,␣
 ↪NumbaPendingDeprecationWarning

from _util.custom_plotting import corr_heatmap, histogram_boxplot,␣
 ↪horizontal_bar, heatmap_boxplot, simple_bar
from _util.k2_iv_woe_function import iv
from _util.make_confusion_matrix import make_cm
from _util.model_comparisons import *
from _util.custom_mem_opt import custom_mem_opt

import pandas as pd
import numpy as np
import pickle
import time
import shap
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, make_scorer
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
from summarytools import dfSummary
from pprint import PrettyPrinter
import torch
from tqdm import tqdm
import torch.nn as nn

pp = PrettyPrinter(width=40, compact=True)
```

1.0.2 Set some options

```
[ ]: warnings.simplefilter(action='ignore', category=NumbaDeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
np.seterr(divide = 'ignore')

# Ensure that the current MacOS version is at least 12.3+, and
# the current current PyTorch installation was built with MPS activated.
print(torch.backends.mps.is_available(), ", ", torch.backends.mps.is_built())

pd.options.mode.chained_assignment = None  # default='warn'
%matplotlib inline
```

```
pp = pprint.PrettyPrinter(width=100)
```

```
True ,  True
```

[ ]:
```
!du -sh ../../_data/loan.csv
```

```
1.1G    ../../_data/loan.csv
```

[ ]:
```
%time

df = pd.read_csv("../../_data/loan.csv", low_memory=False)
```

```
CPU times: user 1e+03 ns, sys: 0 ns, total: 1e+03 ns
Wall time: 3.1 µs
```

[ ]:
```
df = custom_mem_opt(df)
```

```
Memory usage of properties dataframe is : 2500.8916931152344  MB

___MEMORY USAGE AFTER COMPLETION:___
Memory usage is:  1205.1711463928223  MB
This is  48.189657701313784 % of the initial size
```

[ ]:
```
print(np.corrcoef(df['loan_amnt'], df['funded_amnt']))
```

```
[[1.         0.99975527]
 [0.99975527 1.        ]]
```

[ ]:
```
print(df.loan_amnt.corr(df.funded_amnt))
```

```
0.9997552688416222
```

[ ]:
```
pp.pprint(df.head())
```

```
   id  member_id  loan_amnt  funded_amnt  funded_amnt_inv        term  \
0 NaN        NaN       2500         2500           2500.0   36 months
1 NaN        NaN      30000        30000          30000.0   60 months
2 NaN        NaN       5000         5000           5000.0   36 months
3 NaN        NaN       4000         4000           4000.0   36 months
4 NaN        NaN      30000        30000          30000.0   60 months

    int_rate  installment grade sub_grade  … hardship_payoff_balance_amount  \
0  13.562500      84.9375     C        C1  …                            NaN
1  18.937500     777.0000     D        D2  …                            NaN
2  17.968750     180.7500     D        D1  …                            NaN
3  18.937500     146.5000     D        D2  …                            NaN
4  16.140625     732.0000     C        C4  …                            NaN

  hardship_last_payment_amount disbursement_method  debt_settlement_flag  \
0                          NaN                Cash                     N
```

```
1                           NaN                 Cash                         N
2                           NaN                 Cash                         N
3                           NaN                 Cash                         N
4                           NaN                 Cash                         N

  debt_settlement_flag_date settlement_status settlement_date  \
0                        NaN               NaN             NaN
1                        NaN               NaN             NaN
2                        NaN               NaN             NaN
3                        NaN               NaN             NaN
4                        NaN               NaN             NaN

  settlement_amount  settlement_percentage settlement_term
0               NaN                    NaN             NaN
1               NaN                    NaN             NaN
2               NaN                    NaN             NaN
3               NaN                    NaN             NaN
4               NaN                    NaN             NaN

[5 rows x 145 columns]
```

[ ]:
```
nulls = pd.DataFrame({'Count': df.isnull().sum(), 'Percent': 100 * df.isnull().
↪sum()/len(df)})
```

[ ]:
```
print(nulls[nulls['Count']>0].sort_values(by='Percent', ascending=False))
```

```
                                              Count     Percent
id                                          2260668  100.000000
url                                         2260668  100.000000
member_id                                   2260668  100.000000
orig_projected_additional_accrued_interest  2252242   99.627278
hardship_length                             2250055   99.530537
…                                                …          …
delinq_amnt                                      29    0.001283
acc_now_delinq                                   29    0.001283
pub_rec                                          29    0.001283
annual_inc                                        4    0.000177
zip_code                                          1    0.000044

[113 rows x 2 columns]
```

Dropping the variables with more than 80% of Na Values

[ ]:
```
df1 = df.dropna(axis = 1, thresh = int(0.80 * len(df)))
```

[ ]:
```
print(df1['loan_status'].value_counts())
```

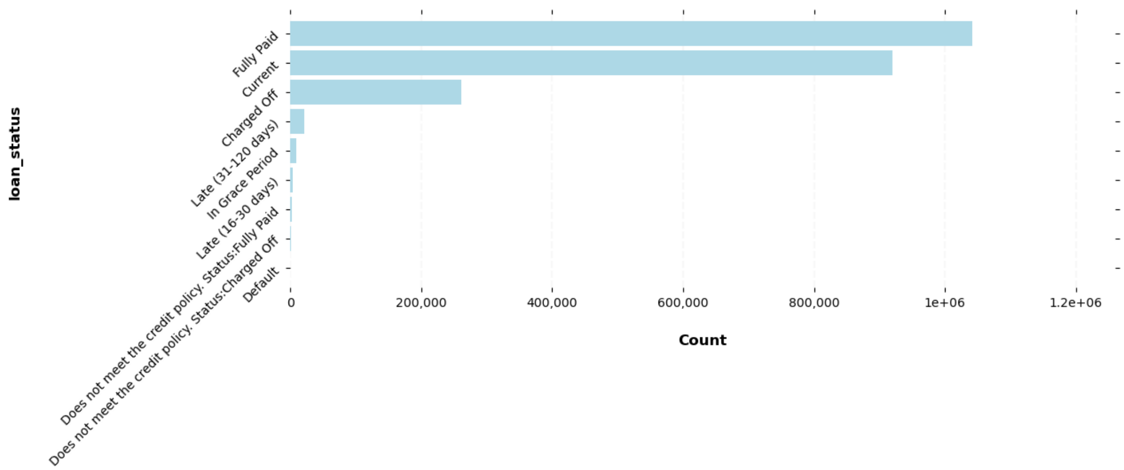
```
loan_status
Fully Paid                                               1041952
```

```
Current                                                919695
Charged Off                                            261655
Late (31-120 days)                                      21897
In Grace Period                                          8952
Late (16-30 days)                                        3737
Does not meet the credit policy. Status:Fully Paid       1988
Does not meet the credit policy. Status:Charged Off       761
Default                                                    31
Name: count, dtype: int64
```

[ ]:
```
print(df1.shape)
```

```
(2260668, 87)
```

[ ]:
```
horizontal_bar(df1['loan_status'], sort_by='x', color='lightblue')
```



1.1 Target Column

[ ]:
```
print(df1.loan_status.value_counts())
```

```
loan_status
Fully Paid                                            1041952
Current                                                919695
Charged Off                                            261655
Late (31-120 days)                                      21897
In Grace Period                                          8952
Late (16-30 days)                                        3737
Does not meet the credit policy. Status:Fully Paid       1988
Does not meet the credit policy. Status:Charged Off       761
Default                                                    31
Name: count, dtype: int64
```

[ ]:
```
df2 = df1.copy()
conditions = [
    (df1['loan_status']=='Fully Paid')
    , (df1['loan_status']=='Charged Off')
    , (~df1['loan_status'].isin(['Default', 'Fully Paid']))
]

values = [0, 1, 2]
df2['default_flag'] = np.select(conditions, values)
print(df2['default_flag'].value_counts().sort_values(ascending = False))
```

```
default_flag
0    1041983
2     957030
1     261655
Name: count, dtype: int64
```

1.2 Mostly current or fully paid, let's look at closed loan counts

[ ]:
```
loan_data = df2[(df2['loan_status'] == "Fully Paid") | (df2['loan_status'] ==␣
 ↪"Charged Off")]
```

[ ]:
```
print(loan_data['default_flag'].value_counts().sort_values(ascending = False))
```

```
default_flag
0    1041952
1     261655
Name: count, dtype: int64
```

[ ]:
```
print(loan_data.shape)
```

```
(1303607, 88)
```

[ ]:
```
print(loan_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1303607 entries, 100 to 2260664
Data columns (total 88 columns):
 #   Column                      Non-Null Count    Dtype
---  ------                      --------------    -----
 0   loan_amnt                   1303607 non-null  int32
 1   funded_amnt                 1303607 non-null  int32
 2   funded_amnt_inv             1303607 non-null  float16
 3   term                        1303607 non-null  object
 4   int_rate                    1303607 non-null  float16
 5   installment                 1303607 non-null  float16
 6   grade                       1303607 non-null  object
 7   sub_grade                   1303607 non-null  object
 8   emp_title                   1221028 non-null  object
```

```
9   emp_length                   1228153 non-null  object
10  home_ownership               1303607 non-null  object
11  annual_inc                   1303607 non-null  float32
12  verification_status          1303607 non-null  object
13  issue_d                      1303607 non-null  object
14  loan_status                  1303607 non-null  object
15  pymnt_plan                   1303607 non-null  object
16  purpose                      1303607 non-null  object
17  title                        1288180 non-null  object
18  zip_code                     1303606 non-null  object
19  addr_state                   1303607 non-null  object
20  dti                          1303295 non-null  float16
21  delinq_2yrs                  1303607 non-null  float16
22  earliest_cr_line             1303607 non-null  object
23  inq_last_6mths               1303606 non-null  float16
24  open_acc                     1303607 non-null  float16
25  pub_rec                      1303607 non-null  float16
26  revol_bal                    1303607 non-null  int32
27  revol_util                   1302797 non-null  float16
28  total_acc                    1303607 non-null  float16
29  initial_list_status          1303607 non-null  object
30  out_prncp                    1303607 non-null  float16
31  out_prncp_inv                1303607 non-null  float16
32  total_pymnt                  1303607 non-null  float16
33  total_pymnt_inv              1303607 non-null  float16
34  total_rec_prncp              1303607 non-null  float16
35  total_rec_int                1303607 non-null  float16
36  total_rec_late_fee           1303607 non-null  float16
37  recoveries                   1303607 non-null  float16
38  collection_recovery_fee      1303607 non-null  float16
39  last_pymnt_d                 1301347 non-null  object
40  last_pymnt_amnt              1303607 non-null  float16
41  last_credit_pull_d           1303553 non-null  object
42  collections_12_mths_ex_med   1303551 non-null  float16
43  policy_code                  1303607 non-null  int8
44  application_type             1303607 non-null  object
45  acc_now_delinq               1303607 non-null  float16
46  tot_coll_amt                 1236080 non-null  float32
47  tot_cur_bal                  1236080 non-null  float32
48  total_rev_hi_lim             1236080 non-null  float32
49  acc_open_past_24mths         1256326 non-null  float16
50  avg_cur_bal                  1236059 non-null  float32
51  bc_open_to_buy               1242968 non-null  float32
52  bc_util                      1242221 non-null  float16
53  chargeoff_within_12_mths     1303551 non-null  float16
54  delinq_amnt                  1303607 non-null  float32
55  mo_sin_old_il_acct           1199312 non-null  float16
56  mo_sin_old_rev_tl_op         1236079 non-null  float16
```

```
 57  mo_sin_rcnt_rev_tl_op       1236079 non-null  float16
 58  mo_sin_rcnt_tl              1236080 non-null  float16
 59  mort_acc                    1256326 non-null  float16
 60  mths_since_recent_bc        1243866 non-null  float16
 61  mths_since_recent_inq       1134058 non-null  float16
 62  num_accts_ever_120_pd       1236080 non-null  float16
 63  num_actv_bc_tl              1236080 non-null  float16
 64  num_actv_rev_tl             1236080 non-null  float16
 65  num_bc_sats                 1247766 non-null  float16
 66  num_bc_tl                   1236080 non-null  float16
 67  num_il_tl                   1236080 non-null  float16
 68  num_op_rev_tl               1236080 non-null  float16
 69  num_rev_accts               1236079 non-null  float16
 70  num_rev_tl_bal_gt_0         1236080 non-null  float16
 71  num_sats                    1247766 non-null  float16
 72  num_tl_120dpd_2m            1188037 non-null  float16
 73  num_tl_30dpd                1236080 non-null  float16
 74  num_tl_90g_dpd_24m          1236080 non-null  float16
 75  num_tl_op_past_12m          1236080 non-null  float16
 76  pct_tl_nvr_dlq              1235926 non-null  float16
 77  percent_bc_gt_75            1242560 non-null  float16
 78  pub_rec_bankruptcies        1302910 non-null  float16
 79  tax_liens                   1303568 non-null  float16
 80  tot_hi_cred_lim             1236080 non-null  float32
 81  total_bal_ex_mort           1256326 non-null  float32
 82  total_bc_limit              1256326 non-null  float32
 83  total_il_high_credit_limit  1236080 non-null  float32
 84  hardship_flag               1303607 non-null  object
 85  disbursement_method         1303607 non-null  object
 86  debt_settlement_flag        1303607 non-null  object
 87  default_flag                1303607 non-null  int64
dtypes: float16(50), float32(11), int32(3), int64(1), int8(1), object(22)
memory usage: 433.9+ MB
None
```

2 To eval imbalance let's look at the c/o rate.

[ ]:
```python
print(loan_data['default_flag'].mean())
```

```
0.2007161667588468
```

2.0.1 ~20% c/o rate. Imbalanced if <10-20%, so we're probably good.

[ ]:
```python
## IV filtering on the data
```

[ ]:
```python
print(loan_data.emp_title.value_counts().nlargest(15))
```

```
emp_title
Teacher              20496
Manager              18704
Owner                 9803
Registered Nurse      8477
RN                    8253
Supervisor            8012
Driver                7230
Sales                 7213
Project Manager       6154
Office Manager        5345
General Manager       5013
Director              4861
owner                 4405
manager               4378
Engineer              4134
Name: count, dtype: int64
```

[ ]:
```
loan_data.emp_title.value_counts().nlargest(10)
conditions = [
    (loan_data['emp_title']=='Teacher')
    , (loan_data['emp_title']=='Manager')
    , (loan_data['emp_title']=='Owner')
    , (loan_data['emp_title']=='Registered Nurse')
    , (loan_data['emp_title']=='RN')
    , (loan_data['emp_title']=='Supervisor')
    , (loan_data['emp_title']=='Driver')
    , (loan_data['emp_title']=='Sales')
    , (loan_data['emp_title']=='Project Manager')
    , (loan_data['emp_title']=='Office Manager')
]

values = ['1','2','3','4','5','6','7','8','9','10']

loan_data['emp_title_cat'] = np.select(conditions, values, default='999')
```

[ ]:
```
loan_data.emp_title.value_counts().nlargest(15)
loan_data.drop('emp_title', axis=1, inplace=True)
```

[ ]:
```
conditions = [
    (loan_data['title']=='Debt consolidation')
    , (loan_data['title']=='Credit card refinancing')
    , (loan_data['title']=='Home improvement')
    , (loan_data['title']=='Other')
    , (loan_data['title']=='Major purchase')
    , (loan_data['title']=='Debt Consolidation')
    , (loan_data['title']=='Medical expenses')
```

```
    , (loan_data['title']=='Business')
    , (loan_data['title']=='Car financing')
    , (loan_data['title']=='Vacation')
]

values = ['1','2','3','4','5','6','7','8','9','10']

loan_data['title_cat'] = np.select(conditions, values, default='999')
```

[ ]:
```
loan_data.drop('title', axis=1, inplace=True)
```

[ ]:
```
iv = iv(loan_data, 'default_flag')
```

[ ]:
```
up_bound = .9
low_bound = .025

iv_filtered = iv['Var'][(iv['IV']<up_bound) & (iv['IV']>=low_bound)].to_list()
iv_f_df = iv[(iv['IV']<up_bound) & (iv['IV']>=low_bound)]

print(iv_f_df.head(n=20))
```

```
                      Var        IV
15              sub_grade  0.494601
16                  grade  0.460762
17               int_rate  0.446919
18                   term  0.173454
19      total_rec_late_fee  0.095761
20                    dti  0.074849
21    acc_open_past_24mths  0.060938
22          bc_open_to_buy  0.059494
23     verification_status  0.056366
24               mort_acc  0.055486
25            avg_cur_bal  0.055073
26         tot_hi_cred_lim  0.050655
27       num_tl_op_past_12m  0.045485
28            tot_cur_bal  0.044746
29          total_bc_limit  0.042863
30   mths_since_recent_inq  0.037495
31          mo_sin_rcnt_tl  0.035737
32        percent_bc_gt_75  0.033379
33            funded_amnt  0.032163
34        funded_amnt_inv  0.032124
```

[ ]:
```
features = [k for k in loan_data.keys() if k!='default_flag']
```

2.0.2 Lets' look at cardinality

```
[ ]: distinct_vals = loan_data.nunique().sort_values(ascending=False)
     print(distinct_vals)
```

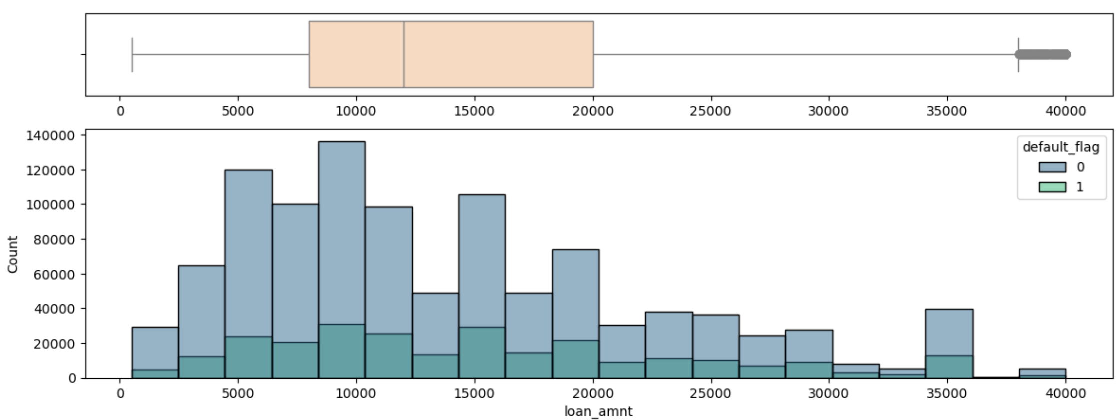
```
tot_hi_cred_lim              421293
tot_cur_bal                  395106
total_bal_ex_mort            176071
total_il_high_credit_limit   160792
revol_bal                     82819
                              …
application_type                  2
pymnt_plan                        1
out_prncp_inv                     1
policy_code                       1
out_prncp                         1
Length: 88, dtype: int64
```

2.1 Data Processing and Data Cleaning

2.1.1 loan_amnt :

The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

```
[ ]: histogram_boxplot(data=loan_data, x='loan_amnt', bins=20, figsize=(14,5),␣
     ↪hue='default_flag')
```



2.1.2 funded_amnt and funded_amnt_inv

funded_amnt: The total amount committed to that loan at that point in time. funded_amnt_inv: The total amount committed by investors for that loan at that point in time.

```
[ ]: dfSummary(loan_data[['funded_amnt', 'funded_amnt_inv']], is_collapsible = True)
```

[ ]: <IPython.core.display.HTML object>

```
[ ]: print(np.corrcoef(loan_data.funded_amnt, loan_data.funded_amnt_inv))
```

```
[[1.         0.99906272]
 [0.99906272 1.        ]]
```

```
[ ]: print(np.corrcoef(loan_data["funded_amnt_inv"], loan_data["loan_amnt"]))
```

```
[[1.         0.99851546]
 [0.99851546 1.        ]]
```

```
[ ]: loan_data[['funded_amnt','funded_amnt_inv','loan_amnt']].head()
```

[ ]:

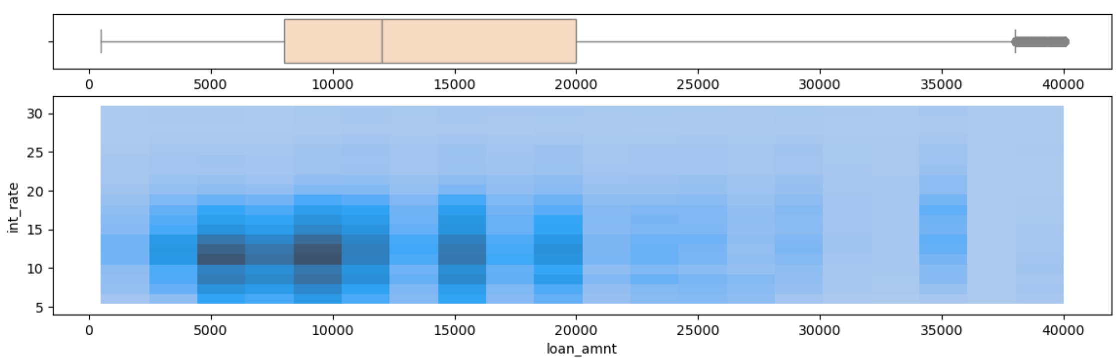
	funded_amnt	funded_amnt_inv	loan_amnt
100	30000	30000.0	30000
152	40000	40000.0	40000
170	20000	20000.0	20000
186	4500	4500.0	4500
215	8425	8424.0	8425

While multicollinearity doesn't affect prediction, it can impact inference via inflated standard errors and low t-obs values and the corresponding statistical significance of regressors. We'll drop funding variables.

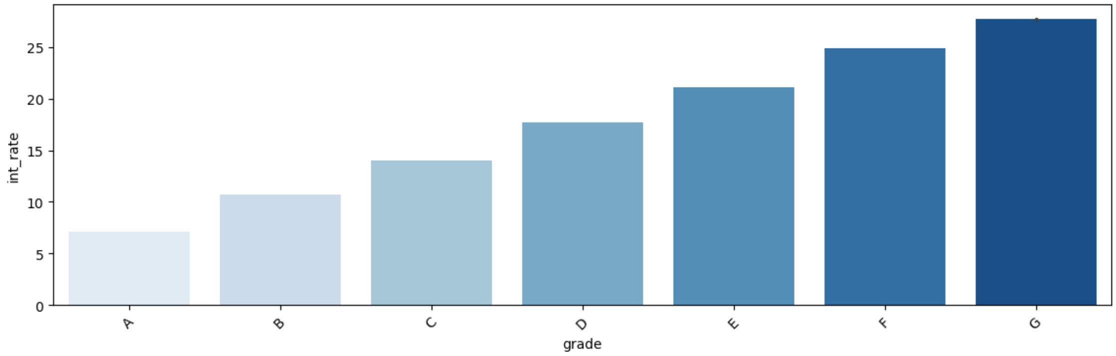
```
[ ]: loan_data = loan_data.drop(["funded_amnt", "funded_amnt_inv"], axis=1)
```

2.1.3 int_rate, grade and sub grade

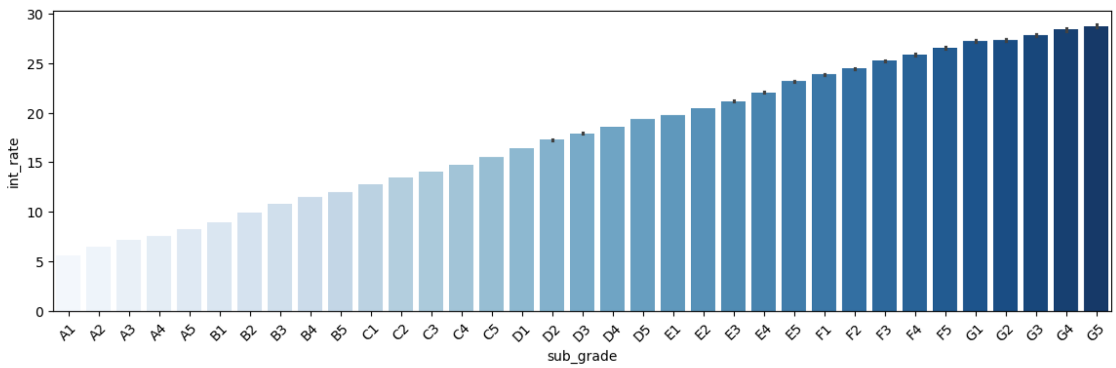
```
[ ]: heatmap_boxplot(data=loan_data, x='loan_amnt', y='int_rate', bins=20,␣
     ↪figsize=(14, 4))
```




```
[ ]: simple_bar(data=loan_data, x='grade', y='int_rate', sort_by='grade',␣
     ↪figsize=(14,4), n=1)
```




```
[ ]: simple_bar(data=loan_data, x='sub_grade', y='int_rate', sort_by='sub_grade',␣
     ↪figsize=(14,4), n=1)
```



As we can see grade and sub-grade are given based on the int_rate so we can drop both of these variables.

```
[ ]: loan_data = loan_data.drop(["grade","sub_grade"], axis = 1)
```

2.1.4 emp_title

The job title supplied by the Borrower when applying for the loan.

2.1.5 zip_code

Dropping zip and going to use state to reduce complexity.

```
[ ]: loan_data = loan_data.drop("zip_code", axis= 1)
```

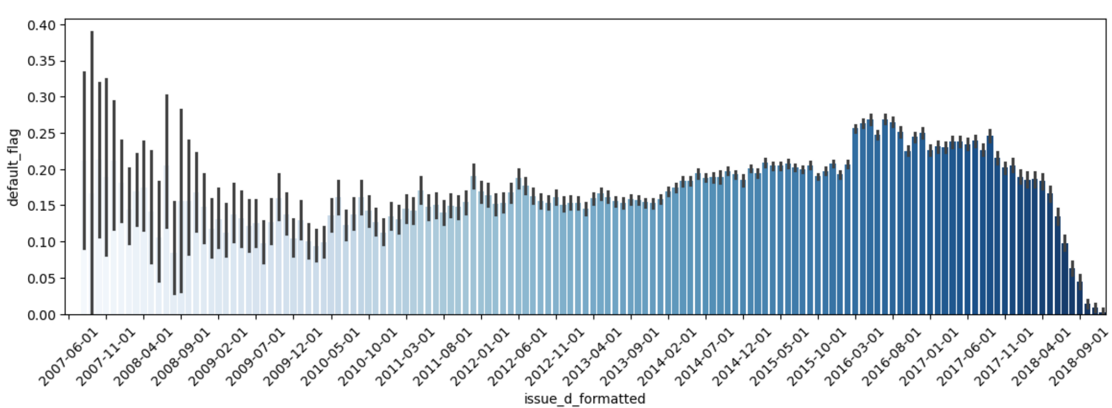
2.1.6 issue_d

An alternative strategy for defining the target could be using a cash flow based response variable. In other words, one could evaluate a cumulative distribution of months to positive cash (e.g. after 9 months 95% of loans have positive cash flow), and then flag good or bad loans with more granular rules.

The logic is that someone may default on a loan yet still generate positive cash flow - you could go this route with the issue date and cash flow information. This would allow you to retain loans that have not yet charged off as well, but in the interest of time for this project we're not going that route.

[ ]:
```
loan_data["id_mon"]=loan_data["issue_d"].str.split("-", n=1, expand=True)[0]
loan_data["id_year"]=loan_data["issue_d"].str.split("-", n=1, expand=True)[1].
↪astype(int)
conditions=[
    (loan_data['id_mon']=='Jan')
    , (loan_data['id_mon']=='Feb')
    , (loan_data['id_mon']=='Mar')
    , (loan_data['id_mon']=='Apr')
    , (loan_data['id_mon']=='May')
    , (loan_data['id_mon']=='Jun')
    , (loan_data['id_mon']=='Jul')
    , (loan_data['id_mon']=='Aug')
    , (loan_data['id_mon']=='Sep')
    , (loan_data['id_mon']=='Oct')
    , (loan_data['id_mon']=='Nov')
    , (loan_data['id_mon']=='Dec')
]

values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
loan_data['id_mon_num'] = np.select(conditions, values)
loan_data['issue_d_formatted'] = pd.to_datetime(dict(year=loan_data.id_year,␣
↪month=loan_data.id_mon_num, day=1))
```

[ ]:
```
simple_bar(data=loan_data, x='issue_d_formatted', y='default_flag',␣
↪sort_by='issue_d_formatted', figsize=(14,4), n=5)
```



```
[ ]: loan_data = loan_data.drop(["issue_d", "issue_d_formatted", "id_mon",␣
     ↪"id_mon_num", "id_year"], axis= 1)
```

2.1.7 out_prncp, out_prncp_inv

Both of this are related to remaining outstanding principal. It is about the future so they are of no use.

```
[ ]: loan_data = loan_data.drop(["out_prncp","out_prncp_inv"], axis = 1)
```

2.1.8 total_pymnt, total_pymnt_inv, total_rec_prncp

- total_pymnt: Payments received to date for total amount funded
- total_pymnt_inv: Payments received to date for portion of total amount funded by investors
- total_rec_prncp: Principal received to date

```
[ ]: print(np.corrcoef(loan_data['total_pymnt'], loan_data['total_pymnt_inv']))
```

```
[[1.         0.99926777]
 [0.99926777 1.        ]]
```

```
[ ]: print(np.corrcoef(loan_data['total_pymnt'], loan_data['total_rec_prncp']))
```

```
[[1.         0.96746846]
 [0.96746846 1.        ]]
```

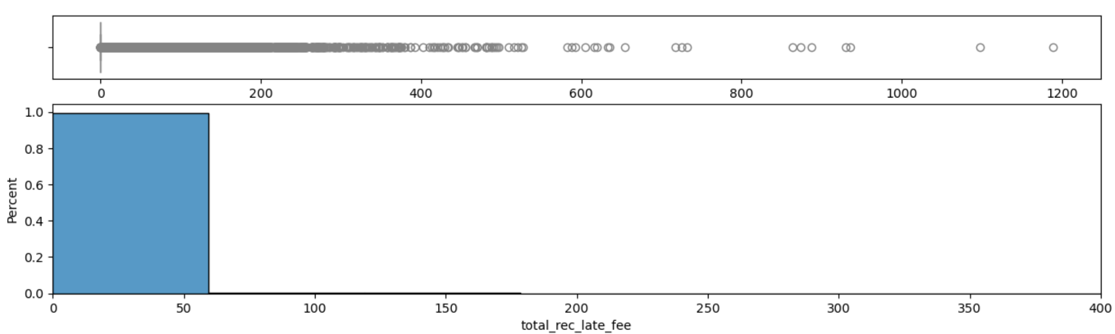
removing total_payment_inv and total_rec_prncp

```
[ ]: loan_data = loan_data.drop(["total_pymnt_inv","total_rec_prncp"], axis = 1)
```

2.1.9 total_rec_late_fee

Late fees received to date - going to round and drop original.

```
[ ]: histogram_boxplot(loan_data, "total_rec_late_fee", figsize=(15,4), bins=20,␣
     ↪xlabel = None, title = None, font_size=12, hist=True, kde=False,␣
     ↪boxplot=True, use_pct=True, xlim=(0,400))
```

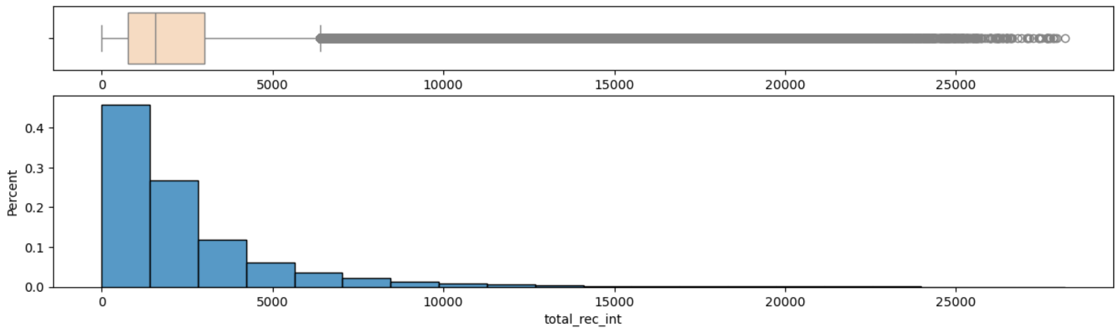


```
[ ]: loan_data = loan_data.drop("total_rec_late_fee", axis = 1)
```

2.1.10 total_rec_int

Interest received to date - as mentioned above, not going the cash flow route so dropping.

```
[ ]: histogram_boxplot(loan_data, "total_rec_int", figsize=(15,4), bins=20,␣
     ↪hist=True, kde=False, boxplot=True, use_pct=True)
```

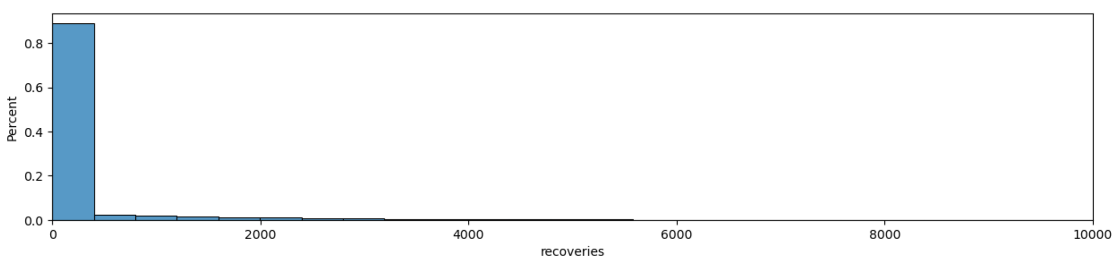



```
[ ]: loan_data = loan_data.drop("total_rec_int", axis = 1)
```

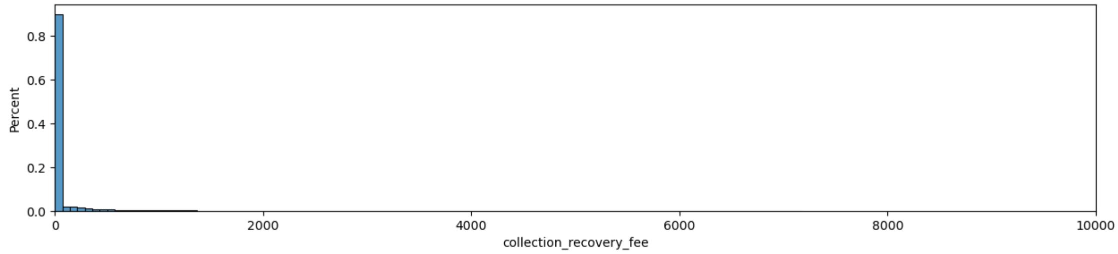
2.1.11 recoveries, collection_recovery_fee, last_pymnt_d, last_pymnt_amnt

- recoveries, collection_recovery_fee, last_pymnt_amnt: Variability in the target captured by total amount paid reflects information in these variables. Given redundancy, going to drop.
- last_pymnt_d with low IV and no cash flow strategy, dropping.

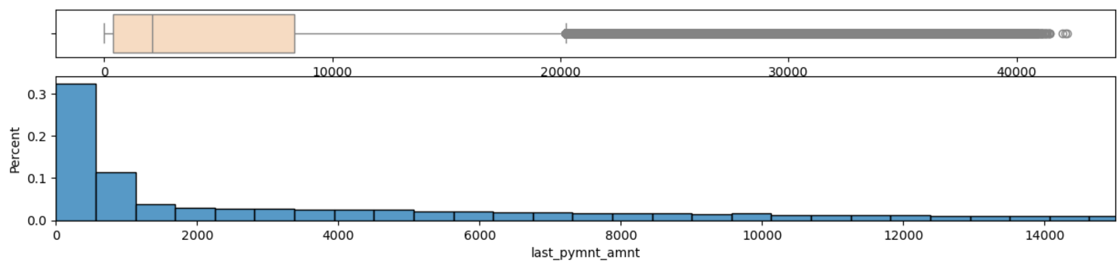
```
[ ]: histogram_boxplot(loan_data, x="recoveries", hue=None, figsize=(15,3),␣
     ↪bins=100, font_size=12, hist=True, boxplot=False, use_pct=True,␣
     ↪xlim=(0,10000))
```




```
[ ]: histogram_boxplot(loan_data, x="collection_recovery_fee", hue=None,␣
     ↪figsize=(15,3), bins=100, font_size=12, hist=True, boxplot=False,␣
     ↪use_pct=True, xlim=(0,10000))
```




```
[ ]: histogram_boxplot(loan_data, x="last_pymnt_amnt", hue=None, figsize=(15,3),␣
     ↪bins=75, font_size=12, hist=True, boxplot=True, use_pct=True, xlim=(0,15000))
```




```
[ ]: loan_data = loan_data.drop(['recoveries', 'collection_recovery_fee',
                                 'last_pymnt_d', 'last_pymnt_amnt'], axis =1)
```

```
[ ]: print(loan_data.shape)
```

```
(1303607, 72)
```

```
[ ]: pp.pprint(loan_data.columns)
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'loan_status',
       'pymnt_plan', 'purpose', 'addr_state', 'dti', 'delinq_2yrs',
       'earliest_cr_line', 'inq_last_6mths', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'initial_list_status',
       'total_pymnt', 'last_credit_pull_d', 'collections_12_mths_ex_med',
       'policy_code', 'application_type', 'acc_now_delinq', 'tot_coll_amt',
       'tot_cur_bal', 'total_rev_hi_lim', 'acc_open_past_24mths',
       'avg_cur_bal', 'bc_open_to_buy', 'bc_util', 'chargeoff_within_12_mths',
       'delinq_amnt', 'mo_sin_old_il_acct', 'mo_sin_old_rev_tl_op',
       'mo_sin_rcnt_rev_tl_op', 'mo_sin_rcnt_tl', 'mort_acc',
       'mths_since_recent_bc', 'mths_since_recent_inq',
       'num_accts_ever_120_pd', 'num_actv_bc_tl', 'num_actv_rev_tl',
       'num_bc_sats', 'num_bc_tl', 'num_il_tl', 'num_op_rev_tl',
       'num_rev_accts', 'num_rev_tl_bal_gt_0', 'num_sats', 'num_tl_120dpd_2m',
```

```
        'num_tl_30dpd', 'num_tl_90g_dpd_24m', 'num_tl_op_past_12m',
        'pct_tl_nvr_dlq', 'percent_bc_gt_75', 'pub_rec_bankruptcies',
        'tax_liens', 'tot_hi_cred_lim', 'total_bal_ex_mort', 'total_bc_limit',
        'total_il_high_credit_limit', 'hardship_flag', 'disbursement_method',
        'debt_settlement_flag', 'default_flag', 'emp_title_cat', 'title_cat'],
       dtype='object')
```

2.1.12 policy_code, application_type, pymnt_plan

```
[ ]: print(loan_data.policy_code.value_counts())
```

```
policy_code
1    1303607
Name: count, dtype: int64
```

```
[ ]: print(loan_data.application_type.value_counts())
```

```
application_type
Individual    1280370
Joint App       23237
Name: count, dtype: int64
```

```
[ ]: print(loan_data.pymnt_plan.value_counts())
```

```
pymnt_plan
n    1303607
Name: count, dtype: int64
```

```
[ ]: print(loan_data.shape[0])
```

```
1303607
```

Single category variables, dropping.

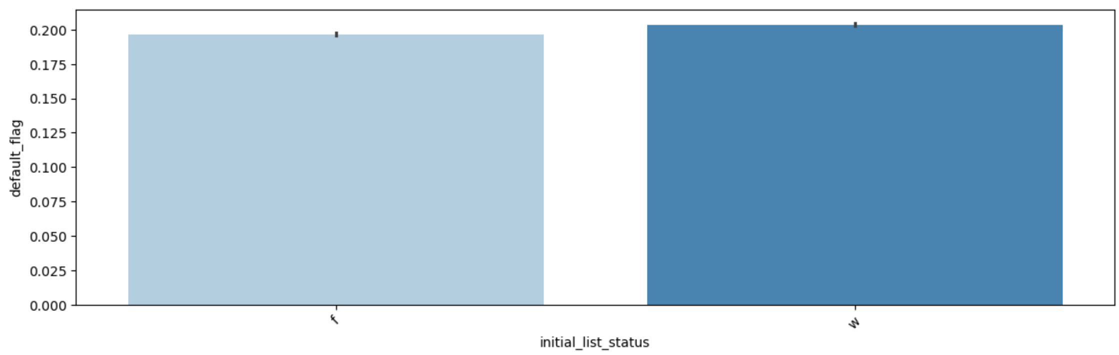
```
[ ]: loan_data = loan_data.drop(["policy_code", "application_type", "pymnt_plan"],␣
      ↪axis = 1)
```

2.1.13 initial_list_status

```
[ ]: print(loan_data.initial_list_status.value_counts())
```

```
initial_list_status
w    751214
f    552393
Name: count, dtype: int64
```

```
[ ]: simple_bar(data=loan_data, x='initial_list_status', y='default_flag',␣
      ↪sort_by='initial_list_status', figsize=(14,4), n=1)
```



Initial list status does not matter whether it is a whole or fractional - dropping

[ ]:
```
loan_data = loan_data.drop("initial_list_status", axis = 1)
```

2.1.14 last_credit_pull_d

Unnecessary complexity added to model - dropping.

[ ]:
```
loan_data = loan_data.drop("last_credit_pull_d", axis = 1)
```

2.1.15 purpose

[ ]:
```
print(loan_data.purpose.value_counts())
```

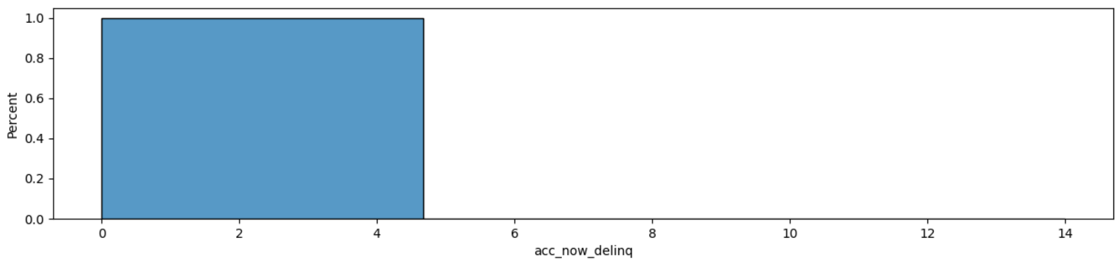
```
purpose
debt_consolidation    757591
credit_card           285704
home_improvement       84495
other                  74934
major_purchase         28328
medical                15023
small_business         15010
car                    14120
moving                  9172
vacation                8732
house                   6967
wedding                 2294
renewable_energy         911
educational              326
Name: count, dtype: int64
```

2.1.16 acc_now_delinq

[ ]:
```
print(loan_data.acc_now_delinq.value_counts())
```

```
acc_now_delinq
0.0     1297441
1.0        5809
2.0         301
3.0          41
4.0          10
5.0           3
6.0           1
14.0          1
Name: count, dtype: int64
```

[ ]:
```
histogram_boxplot(loan_data, x="acc_now_delinq", hue=None, figsize=(15,3),
    bins=3, font_size=12, hist=True, boxplot=False, use_pct=True)
```



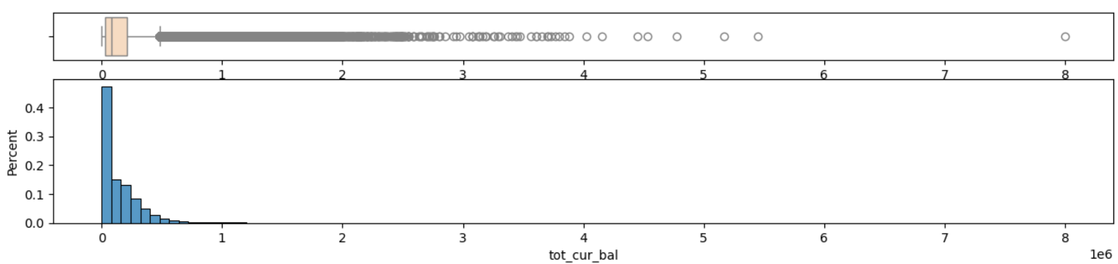
Removing it as most of the values are 0

[ ]:
```
loan_data= loan_data.drop("acc_now_delinq", axis=1)
```

2.1.17 tot_cur_bal

Total current balance of all accounts

[ ]:
```
histogram_boxplot(loan_data, x="tot_cur_bal", hue=None, figsize=(15,3),
    bins=100, font_size=12, hist=True, boxplot=True, use_pct=True)
```



[ ]:
```
print(loan_data.tot_cur_bal.mean())
```

```
141079.7
```

```
[ ]: print(loan_data.tot_cur_bal.median())
```

```
80334.5
```

There is a big difference between mean and median so, we cannot impute and dropping it would be the best choice.

```
[ ]: loan_data = loan_data.drop("tot_cur_bal", axis = 1)
```

2.1.18 tot_coll_amt

Total collection amounts ever owed

```
[ ]: print(loan_data.tot_coll_amt.isna().sum())
```

```
67527
```

```
[ ]: print(loan_data.tot_coll_amt.value_counts().head()/loan_data.shape[0])
```

```
tot_coll_amt
0.0      0.803126
50.0     0.001905
100.0    0.001566
75.0     0.001168
150.0    0.000868
Name: count, dtype: float64
```

Most of the values are 0 and there are many values not availabe. we should drop this feature.

```
[ ]: loan_data = loan_data.drop("tot_coll_amt", axis = 1 )
```

2.1.19 total_rev_hi_lim

Total revolving high credit/credit limit

```
[ ]: print(loan_data.total_rev_hi_lim.isna().sum())
```

```
67527
```

```
[ ]: print(loan_data.total_rev_hi_lim.mean())
```

```
32708.164
```

```
[ ]: print(loan_data.total_rev_hi_lim.median())
```

```
24000.0
```

2.1.20 Imputing rev_hi_lim with median

```
[ ]: loan_data.total_rev_hi_lim.fillna(loan_data.total_rev_hi_lim.median(),
     inplace=True)
```

[ ]:
```
print(loan_data.columns)
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'loan_status',
       'purpose', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line',
       'inq_last_6mths', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util',
       'total_acc', 'total_pymnt', 'collections_12_mths_ex_med',
       'total_rev_hi_lim', 'acc_open_past_24mths', 'avg_cur_bal',
       'bc_open_to_buy', 'bc_util', 'chargeoff_within_12_mths', 'delinq_amnt',
       'mo_sin_old_il_acct', 'mo_sin_old_rev_tl_op', 'mo_sin_rcnt_rev_tl_op',
       'mo_sin_rcnt_tl', 'mort_acc', 'mths_since_recent_bc',
       'mths_since_recent_inq', 'num_accts_ever_120_pd', 'num_actv_bc_tl',
       'num_actv_rev_tl', 'num_bc_sats', 'num_bc_tl', 'num_il_tl',
       'num_op_rev_tl', 'num_rev_accts', 'num_rev_tl_bal_gt_0', 'num_sats',
       'num_tl_120dpd_2m', 'num_tl_30dpd', 'num_tl_90g_dpd_24m',
       'num_tl_op_past_12m', 'pct_tl_nvr_dlq', 'percent_bc_gt_75',
       'pub_rec_bankruptcies', 'tax_liens', 'tot_hi_cred_lim',
       'total_bal_ex_mort', 'total_bc_limit', 'total_il_high_credit_limit',
       'hardship_flag', 'disbursement_method', 'debt_settlement_flag',
       'default_flag', 'emp_title_cat', 'title_cat'],
      dtype='object')
```

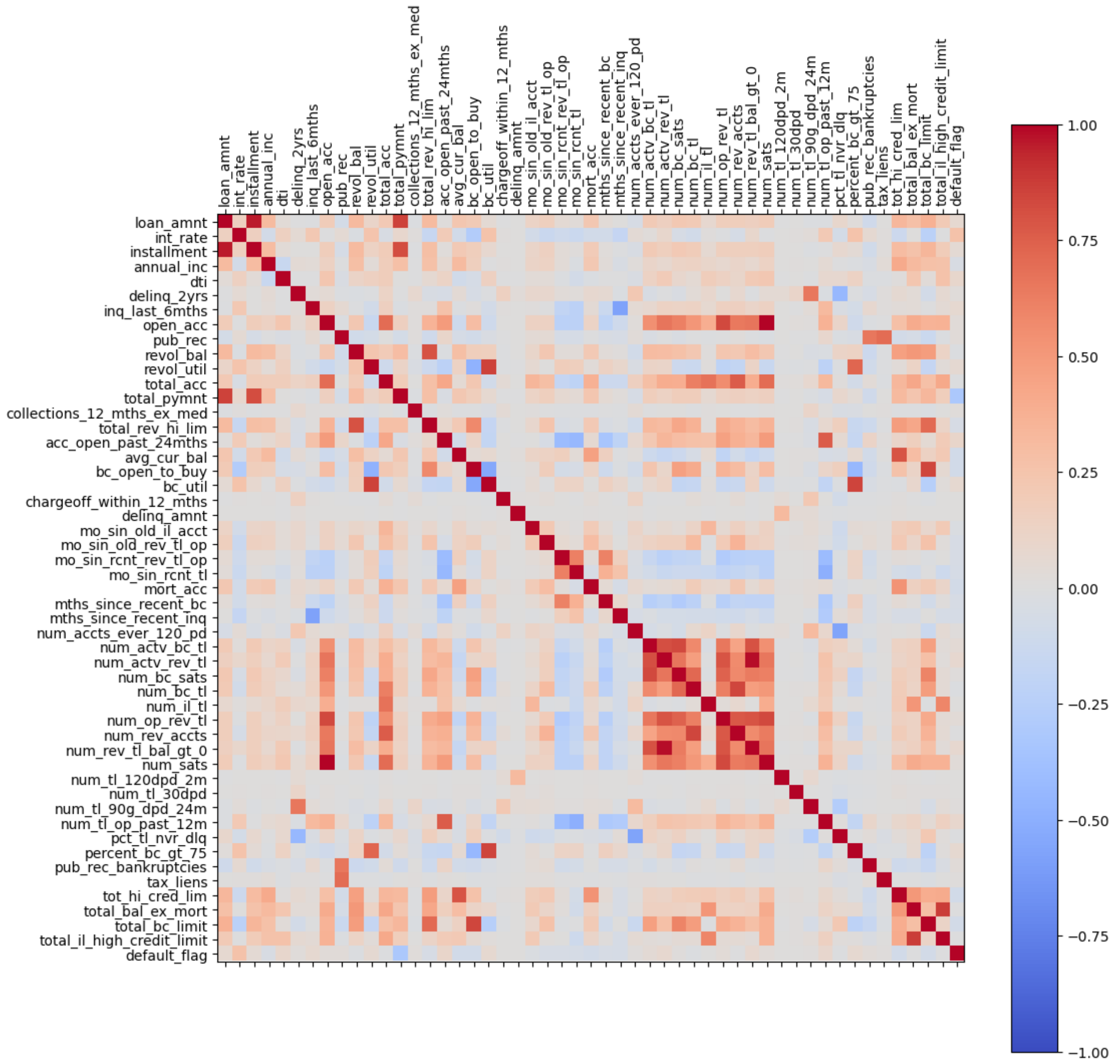
[ ]:
```
print(loan_data.shape)
```

```
(1303607, 64)
```

2.2 Numerical features

[ ]:
```
num_cols = loan_data._get_numeric_data().columns
data = loan_data.select_dtypes(np.number)
```

[ ]:
```
corr_heatmap(data=data, num_cols=num_cols)
```



Not dropping any additional features at this point.

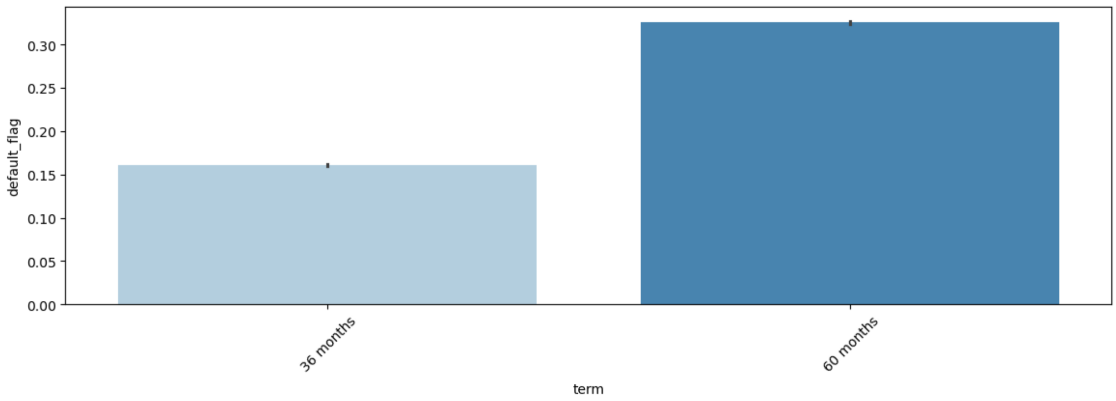
2.3 Categorical features

[ ]:
```
print(loan_data.columns[loan_data.dtypes == object])
```

```
Index(['term', 'emp_length', 'home_ownership', 'verification_status',
       'loan_status', 'purpose', 'addr_state', 'earliest_cr_line',
       'hardship_flag', 'disbursement_method', 'debt_settlement_flag',
       'emp_title_cat', 'title_cat'],
      dtype='object')
```

2.3.1 term

[ ]:
```
simple_bar(data=loan_data, x='term', y='default_flag', sort_by='term',␣
↪figsize=(14,4), n=1)
```



2.3.2 emp_length

[ ]:
```
print(loan_data.emp_length.value_counts())
```

```
emp_length
10+ years    428547
2 years      117820
< 1 year     104550
3 years      104200
1 year        85677
5 years       81623
4 years       78029
6 years       60933
8 years       59125
7 years       58145
9 years       49504
Name: count, dtype: int64
```

[ ]:
```
print(loan_data.emp_length.isna().sum())
```

```
75454
```

[ ]:
```
loan_data["emp_length"] = loan_data["emp_length"].replace({'years':'','year':
↪'',' ':'','<':'','\+':'','n/a':'0'}, regex = True)
```

[ ]:
```
loan_data.emp_length.isna().sum()
```

[ ]: 75454

```
[ ]: pp.pprint(loan_data.columns[loan_data.isnull().any()].tolist())
```

```
['emp_length', 'dti', 'inq_last_6mths',
 'revol_util',
 'collections_12_mths_ex_med',
 'acc_open_past_24mths', 'avg_cur_bal',
 'bc_open_to_buy', 'bc_util',
 'chargeoff_within_12_mths',
 'mo_sin_old_il_acct',
 'mo_sin_old_rev_tl_op',
 'mo_sin_rcnt_rev_tl_op',
 'mo_sin_rcnt_tl', 'mort_acc',
 'mths_since_recent_bc',
 'mths_since_recent_inq',
 'num_accts_ever_120_pd',
 'num_actv_bc_tl', 'num_actv_rev_tl',
 'num_bc_sats', 'num_bc_tl',
 'num_il_tl', 'num_op_rev_tl',
 'num_rev_accts', 'num_rev_tl_bal_gt_0',
 'num_sats', 'num_tl_120dpd_2m',
 'num_tl_30dpd', 'num_tl_90g_dpd_24m',
 'num_tl_op_past_12m', 'pct_tl_nvr_dlq',
 'percent_bc_gt_75',
 'pub_rec_bankruptcies', 'tax_liens',
 'tot_hi_cred_lim', 'total_bal_ex_mort',
 'total_bc_limit',
 'total_il_high_credit_limit']
```

```
[ ]: nan_cols=loan_data.columns[loan_data.isnull().any()].tolist()
num_cols = loan_data._get_numeric_data().columns
means=loan_data[num_cols].mean().to_dict()
loan_data.fillna(value=means, inplace=True)
```

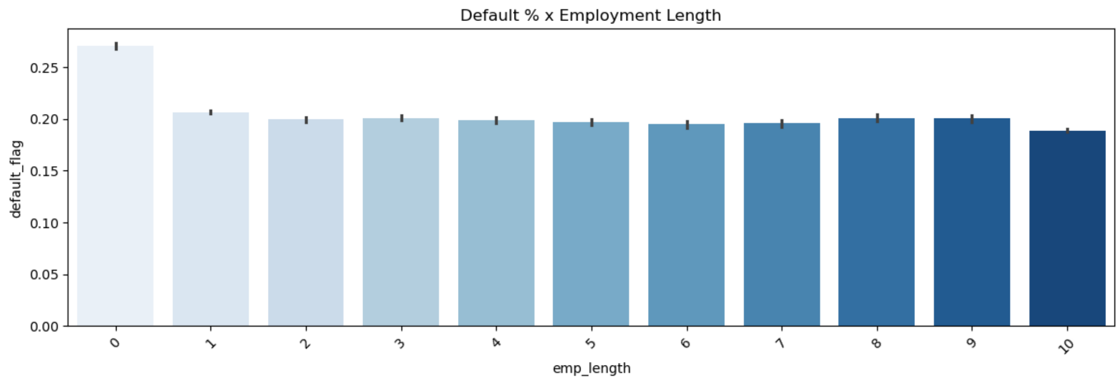
```
[ ]: loan_data.fillna(0, inplace=True)
```

```
[ ]: loan_data["emp_length"] = loan_data["emp_length"].apply(lambda x:int(x))
```

```
[ ]: bin_test = pd.qcut(loan_data['emp_length'], q=10, labels=None,␣
  ↪duplicates='drop').value_counts().to_dict()
pp.pprint(bin_test)
```

```
{Interval(-0.001, 1.0, closed='right'): 265681,
 Interval(1.0, 3.0, closed='right'): 222020,
 Interval(3.0, 4.0, closed='right'): 78029,
 Interval(4.0, 6.0, closed='right'): 142556,
 Interval(6.0, 8.0, closed='right'): 117270,
 Interval(8.0, 10.0, closed='right'): 478051}
```

```
[ ]: simple_bar(data=loan_data, x='emp_length', y='default_flag', sort_by='term',␣
     ↪figsize=(14,4), n=1, title='Default % x Employment Length')
```

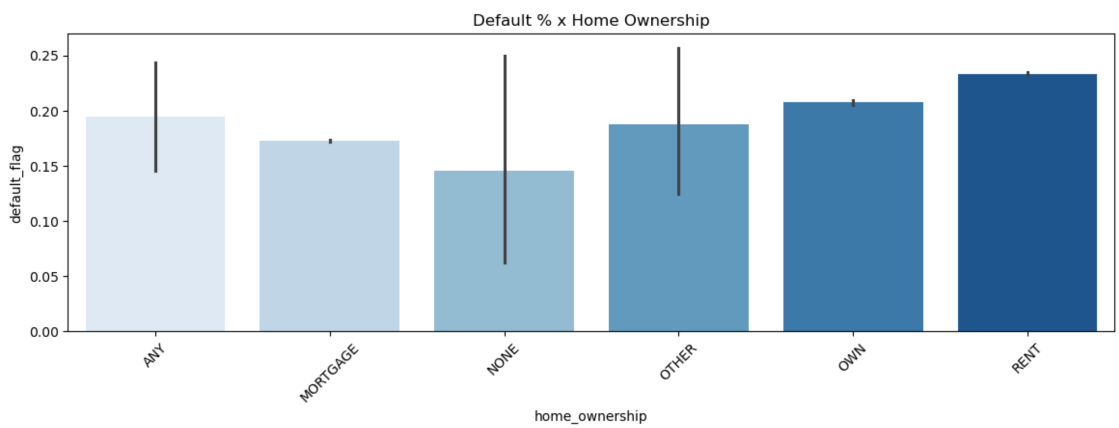


2.3.3 home_ownership

```
[ ]: print(loan_data.home_ownership.value_counts())
```

```
home_ownership
MORTGAGE    645496
RENT        517808
OWN         139844
ANY            267
OTHER          144
NONE            48
Name: count, dtype: int64
```

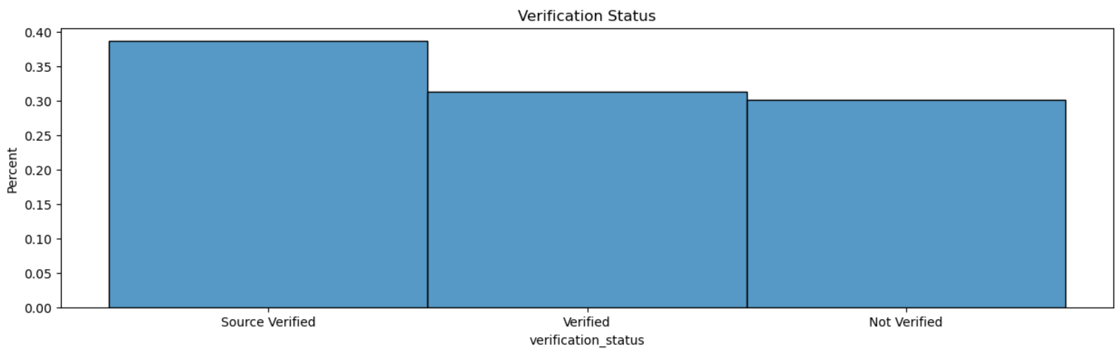
```
[ ]: simple_bar(data=loan_data, x='home_ownership', y='default_flag',␣
     ↪sort_by='home_ownership', figsize=(14,4), n=1, title='Default % x Home␣
     ↪Ownership')
```



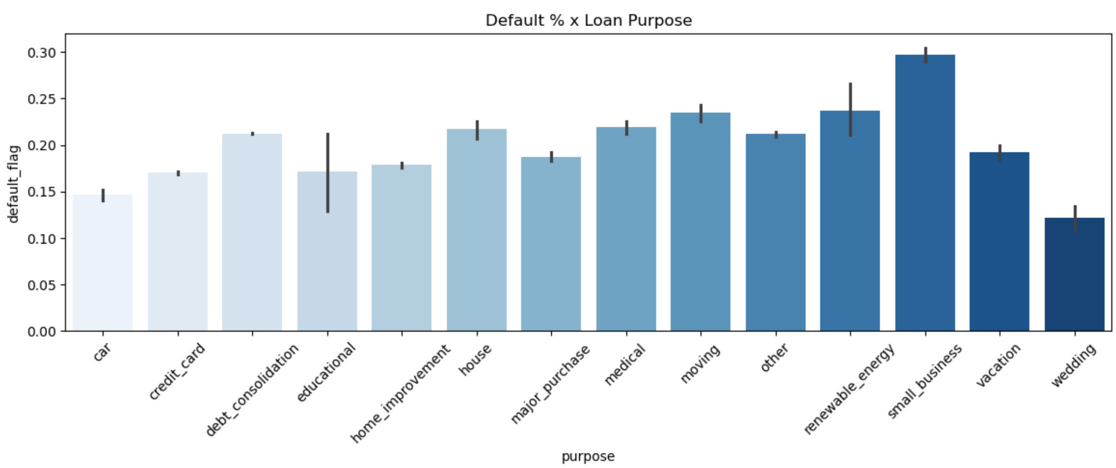
```
[ ]: print(loan_data.verification_status.value_counts())
```

```
verification_status
Source Verified    503726
Verified           407676
Not Verified       392205
Name: count, dtype: int64
```

```
[ ]: histogram_boxplot(loan_data, x="verification_status", figsize=(15,4), bins=3,
       font_size=12, hist=True, boxplot=False, use_pct=True, title='Verification
       Status')
```




```
[ ]: simple_bar(data=loan_data, x='purpose', y='default_flag',
       sort_by='purpose',figsize=(14,4), n=1, title='Default % x Loan Purpose')
```



2.3.4 earliest_cr_line

The month the borrower's earliest reported credit line was opened

Removing.

[ ]:
```python
loan_data.drop("earliest_cr_line", axis=1, inplace=True)
```

Drop some final features that can't be used on unseen data.

[ ]:
```python
loan_data.drop(['total_pymnt', 'debt_settlement_flag'], axis=1, inplace=True)
```

[ ]:
```python
pp.pprint(loan_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1303607 entries, 100 to 2260664
Data columns (total 61 columns):
 #   Column                      Non-Null Count    Dtype
---  ------                      --------------    -----
 0   loan_amnt                   1303607 non-null  int32
 1   term                        1303607 non-null  object
 2   int_rate                    1303607 non-null  float16
 3   installment                 1303607 non-null  float16
 4   emp_length                  1303607 non-null  int64
 5   home_ownership              1303607 non-null  object
 6   annual_inc                  1303607 non-null  float32
 7   verification_status         1303607 non-null  object
 8   loan_status                 1303607 non-null  object
 9   purpose                     1303607 non-null  object
 10  addr_state                  1303607 non-null  object
 11  dti                         1303607 non-null  float16
 12  delinq_2yrs                 1303607 non-null  float16
 13  inq_last_6mths              1303607 non-null  float16
 14  open_acc                    1303607 non-null  float16
 15  pub_rec                     1303607 non-null  float16
 16  revol_bal                   1303607 non-null  int32
 17  revol_util                  1303607 non-null  float16
 18  total_acc                   1303607 non-null  float16
 19  collections_12_mths_ex_med  1303607 non-null  float16
 20  total_rev_hi_lim            1303607 non-null  float32
 21  acc_open_past_24mths        1303607 non-null  float16
 22  avg_cur_bal                 1303607 non-null  float32
 23  bc_open_to_buy              1303607 non-null  float32
 24  bc_util                     1303607 non-null  float16
 25  chargeoff_within_12_mths    1303607 non-null  float16
 26  delinq_amnt                 1303607 non-null  float32
 27  mo_sin_old_il_acct          1303607 non-null  float16
 28  mo_sin_old_rev_tl_op        1303607 non-null  float16
 29  mo_sin_rcnt_rev_tl_op       1303607 non-null  float16
 30  mo_sin_rcnt_tl              1303607 non-null  float16
```

```
 31  mort_acc                    1303607 non-null  float16
 32  mths_since_recent_bc        1303607 non-null  float16
 33  mths_since_recent_inq       1303607 non-null  float16
 34  num_accts_ever_120_pd       1303607 non-null  float16
 35  num_actv_bc_tl              1303607 non-null  float16
 36  num_actv_rev_tl             1303607 non-null  float16
 37  num_bc_sats                 1303607 non-null  float16
 38  num_bc_tl                   1303607 non-null  float16
 39  num_il_tl                   1303607 non-null  float16
 40  num_op_rev_tl               1303607 non-null  float16
 41  num_rev_accts               1303607 non-null  float16
 42  num_rev_tl_bal_gt_0         1303607 non-null  float16
 43  num_sats                    1303607 non-null  float16
 44  num_tl_120dpd_2m            1303607 non-null  float16
 45  num_tl_30dpd                1303607 non-null  float16
 46  num_tl_90g_dpd_24m          1303607 non-null  float16
 47  num_tl_op_past_12m          1303607 non-null  float16
 48  pct_tl_nvr_dlq              1303607 non-null  float16
 49  percent_bc_gt_75            1303607 non-null  float16
 50  pub_rec_bankruptcies        1303607 non-null  float16
 51  tax_liens                   1303607 non-null  float16
 52  tot_hi_cred_lim             1303607 non-null  float32
 53  total_bal_ex_mort           1303607 non-null  float32
 54  total_bc_limit              1303607 non-null  float32
 55  total_il_high_credit_limit  1303607 non-null  float32
 56  hardship_flag               1303607 non-null  object
 57  disbursement_method         1303607 non-null  object
 58  default_flag                1303607 non-null  int64
 59  emp_title_cat               1303607 non-null  object
 60  title_cat                   1303607 non-null  object
dtypes: float16(38), float32(9), int32(2), int64(2), object(10)
memory usage: 278.5+ MB
None
```

2.3.5 Encoding

[ ]:
```
num_cols = loan_data.select_dtypes(include=np.number).columns
data_type_dict = loan_data.drop(["loan_status", "default_flag"], axis = 1).
↪dtypes
cat_vars = []
for key, value in data_type_dict.items():
    if data_type_dict[key] in ['bool', 'object']:
        cat_vars.append(key)

loan_data = pd.get_dummies(loan_data, columns = cat_vars, dtype=int)
loan_data_s = loan_data.copy()
```

```
scaler=StandardScaler()
loan_data_s[loan_data_s.drop(["loan_status", "default_flag"], axis = 1).
 ↪columns] = scaler.fit_transform(loan_data_s.drop(["loan_status",␣
 ↪"default_flag"], axis = 1))

y = loan_data_s.default_flag
X = loan_data_s.drop(["loan_status", "default_flag"], axis = 1)
```

2.4 Train-Test Split

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

print("Shape of X_train: ", X_train.shape)
print("Shape of y_train: ", y_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_test: ", y_test.shape)
```

```
Shape of X_train:  (912524, 152)
Shape of y_train:  (912524,)
Shape of X_test:  (391083, 152)
Shape of y_test:  (391083,)
```

[ ]:
```
## Models
```

[ ]:
```
samples_dict = {'X_train': X_train, 'X_test': X_test, 'y_train': y_train,␣
 ↪'y_test': y_test}
models_dict = {}
```

3 Logistic Regression - pytorch NN

[ ]:
```
if torch.backends.mps.is_available():
    device = "mps"
    processor = torch.device("mps")
    x = torch.ones(1, device=processor)
    print (x)
else:
    print ("GPU device not found.")
```

```
tensor([1.], device='mps:0')
```

[ ]:
```
X_train_t = torch.tensor(X_train.values, dtype=torch.float, device=processor)
X_test_t = torch.tensor(X_test.values, dtype=torch.float, device=processor)
y_train_t = torch.tensor(y_train.values, dtype=torch.float, device=processor).
 ↪view(-1,1)
y_test_t = torch.tensor(y_test.values, dtype=torch.float, device=processor).
 ↪view(-1,1)
```

[ ]:
```python
class LogReg(nn.Module):
    def __init__(self, num_features):
        super().__init__()
        self.layer0 = nn.Linear(in_features=num_features,␣
 ↪out_features=round(num_features/2))
        self.relu = nn.ReLU()
        self.layer1 = nn.Linear(round(num_features/2), 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.layer0(x)
        x = self.relu(x)
        x = self.layer1(x)
        x = self.sigmoid(x)
        return x

def calculate_accuracy(preds, actuals):

    with torch.no_grad():
        rounded_preds = torch.round(preds)
        num_correct = torch.sum(rounded_preds == actuals)
        accuracy = num_correct/len(preds)

    return accuracy
```

[ ]:
```python
train_losses = []
test_losses  = []
train_accs = []
test_accs  = []
iter = 0

n_features = X_train_t.shape[1]
print(n_features)

model=LogReg(n_features).to(device)

criterion=torch.nn.MSELoss().to(device)

optimizer=torch.optim.Adam(model.parameters(),lr=0.001)

start = time.perf_counter()
epochs=12000
for epoch in tqdm(range(int(epochs)),desc='Training Epochs'):
    # Forward propagation
    train_preds_t=model(X_train_t)
    train_loss_t=criterion(train_preds_t,y_train_t).to(device)
```

```python
    # Backward propagation
    train_loss_t.backward()

    # Gradient descent step
    optimizer.step()

    # Reset gradient
    optimizer.zero_grad()

    # Calculating the loss and accuracy for the test dataset
    # Predicting test data #b
    with torch.no_grad():
        test_preds_t = model(X_test_t)
        test_loss_t  = criterion(test_preds_t, y_test_t).to(device)

    # Calculate accuracy #c
    train_acc_t = calculate_accuracy(train_preds_t, y_train_t)
    test_acc_t  = calculate_accuracy(test_preds_t, y_test_t)

    # Store training history #f
    train_losses.append(train_loss_t.item())
    test_losses.append(test_loss_t.item())
    train_accs.append(train_acc_t.item())
    test_accs.append(test_acc_t.item())

    # Print training data #g
    if epoch%int(epochs/10)==0:
        print(f'Epoch: {epoch} \t|' \
            f' Train loss: {np.round(train_loss_t.item(),4)} \t|' \
            f' Test loss: {np.round(test_loss_t.item(),4)} \t|' \
            f' Train acc: {np.round(train_acc_t.item(),4)} \t|' \
            f' Test acc: {np.round(test_acc_t.item(),4)}')

end = time.perf_counter()
```

```
152

Training Epochs:   0%|          | 3/12000 [00:00<19:16, 10.37it/s]

Epoch: 0        | Train loss: 0.2802    | Test loss: 0.2725     | Train acc:
0.3013     | Test acc: 0.3445

Training Epochs:  10%|          | 1203/12000 [01:02<09:57, 18.08it/s]

Epoch: 1200     | Train loss: 0.1404    | Test loss: 0.1427     | Train acc:
0.8077     | Test acc: 0.8045

Training Epochs:  20%|          | 2403/12000 [02:08<08:53, 17.98it/s]

Epoch: 2400     | Train loss: 0.1389    | Test loss: 0.1434     | Train acc:
0.8101     | Test acc: 0.8034
```

```
Training Epochs:  30%|          | 3603/12000 [03:21<09:08, 15.31it/s]

Epoch: 3600     | Train loss: 0.1382    | Test loss: 0.1441     | Train acc:
0.8112     | Test acc: 0.8023

Training Epochs:  40%|          | 4803/12000 [04:41<07:11, 16.67it/s]

Epoch: 4800     | Train loss: 0.1379    | Test loss: 0.1446     | Train acc:
0.8118     | Test acc: 0.8018

Training Epochs:  50%|          | 6003/12000 [05:51<05:21, 18.65it/s]

Epoch: 6000     | Train loss: 0.1377    | Test loss: 0.1448     | Train acc:
0.812      | Test acc: 0.8013

Training Epochs:  60%|          | 7203/12000 [06:57<04:14, 18.86it/s]

Epoch: 7200     | Train loss: 0.1377    | Test loss: 0.1449     | Train acc:
0.8122     | Test acc: 0.8012

Training Epochs:  70%|          | 8403/12000 [08:00<03:07, 19.18it/s]

Epoch: 8400     | Train loss: 0.1376    | Test loss: 0.1449     | Train acc:
0.8122     | Test acc: 0.8011

Training Epochs:  80%|          | 9603/12000 [09:04<02:10, 18.33it/s]

Epoch: 9600     | Train loss: 0.1376    | Test loss: 0.145      | Train acc:
0.8122     | Test acc: 0.8011

Training Epochs:  90%|          | 10803/12000 [10:10<01:06, 17.90it/s]

Epoch: 10800    | Train loss: 0.1376    | Test loss: 0.145      | Train acc:
0.8123     | Test acc: 0.8011

Training Epochs: 100%|          | 12000/12000 [11:16<00:00, 17.73it/s]
```

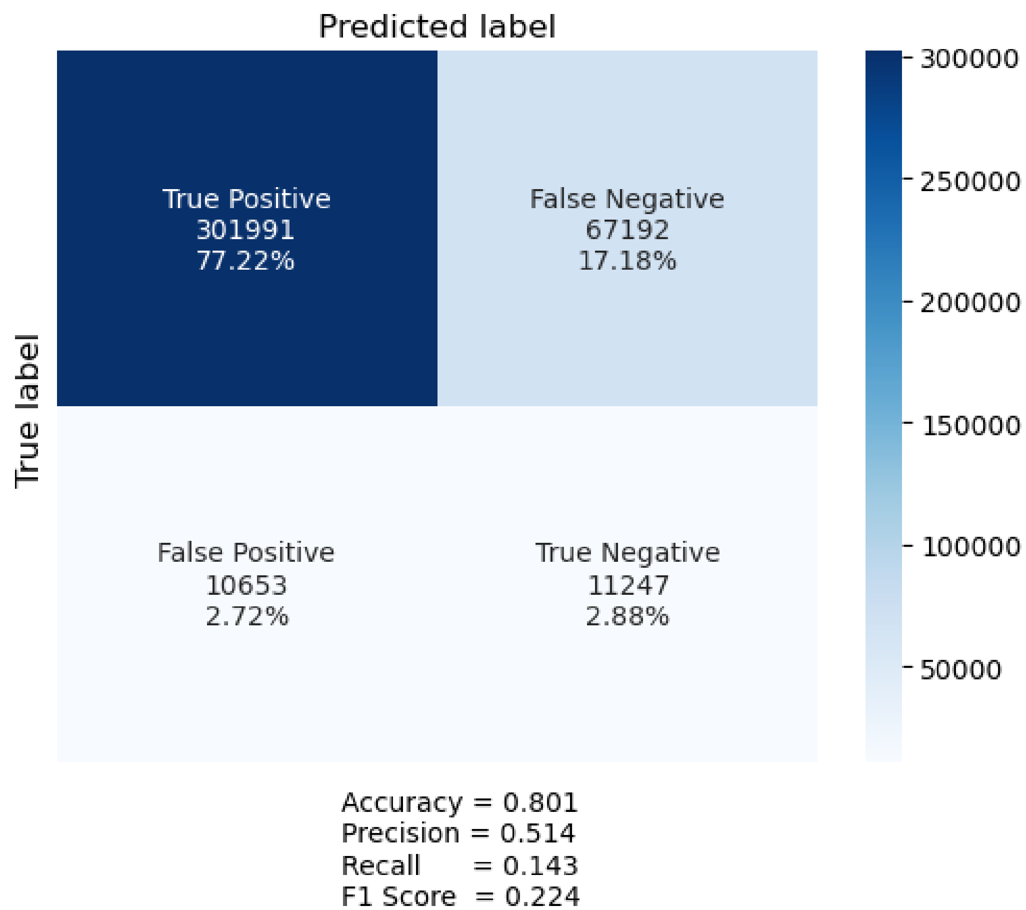
[ ]:

```
## Confusion Matrix on unseen test set
test_preds = test_preds_t.cpu().detach().numpy()

for i in range(len(y_test)):
    if test_preds[i]>0.5:
        test_preds[i]=1
    else:
        test_preds[i]=0
```

[ ]:

```
pred_train = prob_to_label(model(X_train_t).cpu().detach().numpy())
pred_test = prob_to_label(model(X_test_t).cpu().detach().numpy())
samples_dict['pred_train'] = pred_train
samples_dict['pred_test'] = pred_test

# Compute and print the confusion matrix and classification report
#Creating confusion matrix
make_cm([y_test,pred_test])
```

```
#Models and sample dicts
models_dict['Torch LogReg'] = {'pred_train':pred_train, 'pred_test':pred_test,␣
 ↪'est_time':end-start}
#Print model comparisons
model_comparisons(models_dict, samples_dict)
```



[ ]:
```
          Model  Conv. Time (sec.)  Train_Accuracy  Test_Accuracy  \
0  Torch LogReg           677.2053        0.812232        0.80095

   Train_Recall  Test_Recall  Train_Precision  Test_Precision  Train_F1-Score  \
0      0.170373     0.143385         0.617422        0.513562        0.267054

   Test_F1-Score
0        0.22418
```

[ ]:
```
## serialize model ##
torch.save(model, "_pkls/nn_model.p")
```

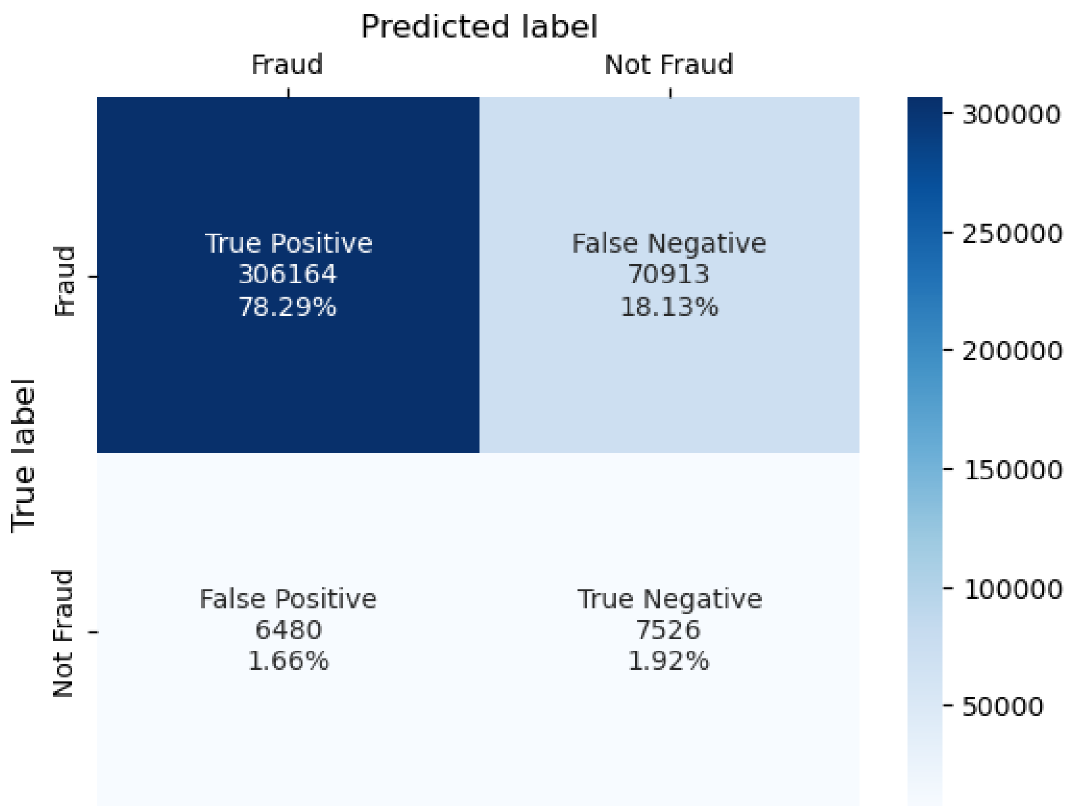
```
## load model ##
#model = torch.load("pickles/nn_model.p")
```

4 Logistic Regression - sklearn

[ ]:
```
%%time

pipe = Pipeline(steps=[
('logit', LogisticRegression(solver='sag', max_iter=1000))
])

# Fit the classifier to the training data
start = time.perf_counter()
pipe.fit(X_train, y_train)
end = time.perf_counter()
```

```
CPU times: user 7min 20s, sys: 880 ms, total: 7min 21s
Wall time: 7min 23s
```

[ ]:
```
# Generate predictions
pred_train = pipe.predict(X_train)
pred_test = pipe.predict(X_test)

#Creating confusion matrix
make_cm([y_test,pred_test], labels=["Fraud", "Not Fraud"])

#Models and sample dicts
models_dict['sklearn LogReg'] = {'pred_train':pred_train, 'pred_test':
↪pred_test, 'est_time':end-start}

#Print model comparisons
model_comparisons(models_dict, samples_dict)
```



Accuracy = 0.802
Precision = 0.537
Recall = 0.096
F1 Score = 0.163

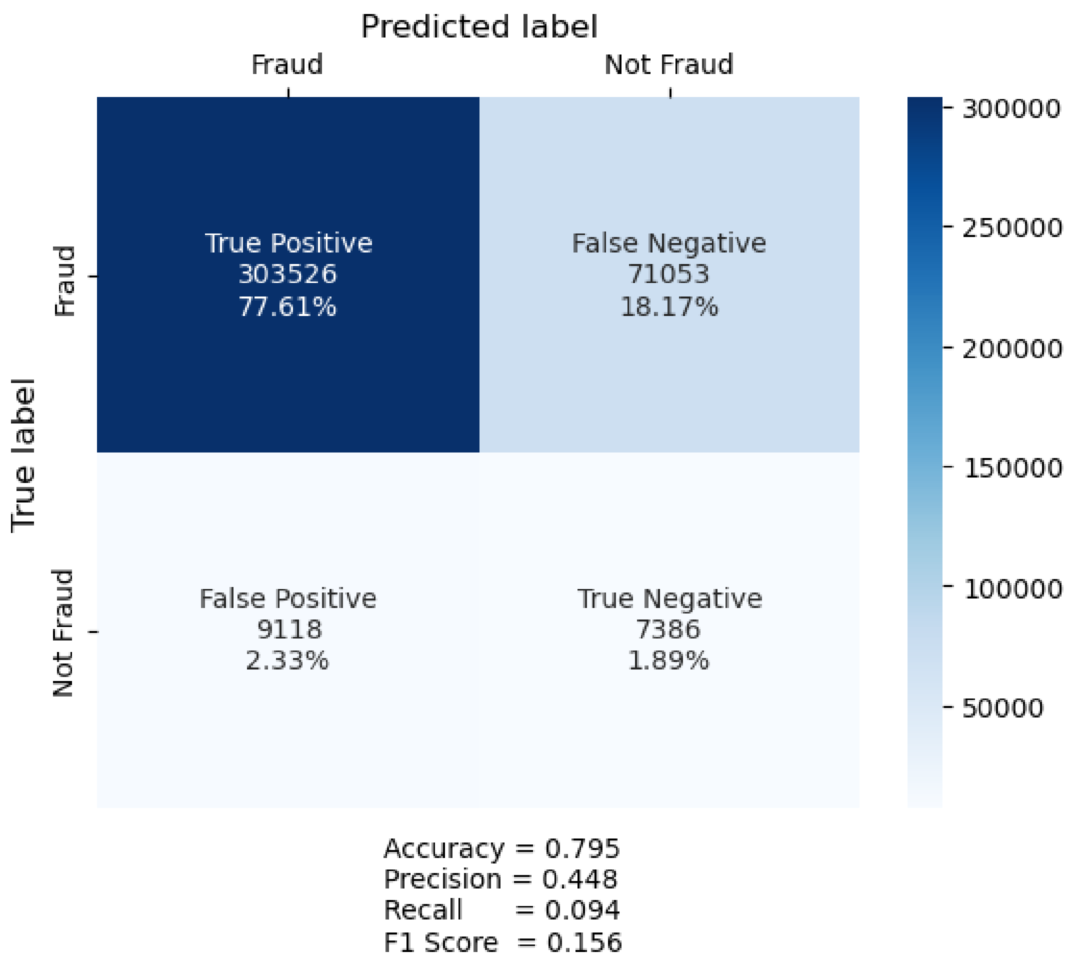
```
[ ]:             Model  Conv. Time (sec.)  Train_Accuracy  Test_Accuracy  \
     0    Torch LogReg           677.2053        0.812232       0.800950
     1  sklearn LogReg           443.0372        0.802343       0.802106

        Train_Recall  Test_Recall  Train_Precision  Test_Precision  Train_F1-Score  \
     0      0.170373     0.143385         0.617422        0.513562        0.267054
     1      0.097300     0.095947         0.543423        0.537341        0.165049

        Test_F1-Score
     0       0.224180
     1       0.162821
```

```
[ ]: pickle.dump(pipe, open("_pkls/sklearn_logreg.p", "wb"))
```

5 Random Forest

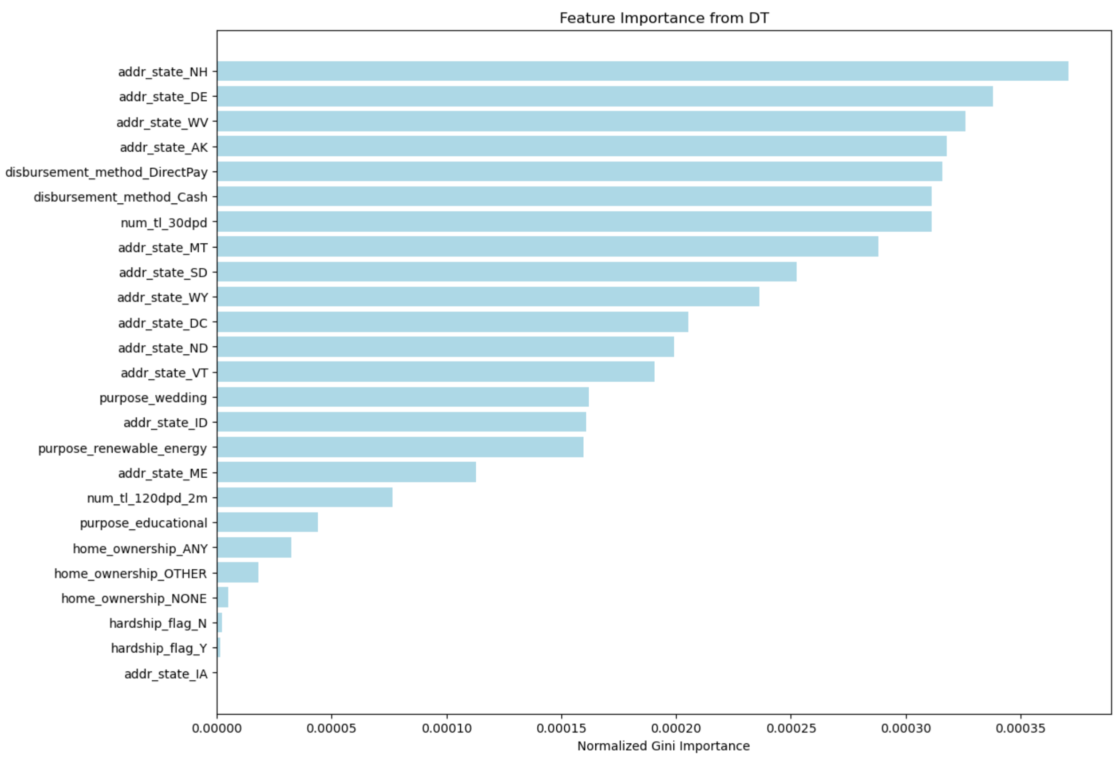
[ ]:
```
clf_rf = RandomForestClassifier(n_estimators=10, random_state=21)
start = time.perf_counter()
clf_rf.fit(X_train, y_train)
end = time.perf_counter()
```

[ ]:
```
# Generate predictions
pred_train = clf_rf.predict(X_train)
pred_test = clf_rf.predict(X_test)

#Creating confusion matrix
make_cm([y_test,pred_test], labels=["Fraud", "Not Fraud"])

#Models and sample dicts
models_dict['Random Forest'] = {'pred_train':pred_train, 'pred_test':pred_test,␣
↪'est_time':end-start}

#Print model comparisons
model_comparisons(models_dict, samples_dict)
```




```
[ ]:             Model  Conv. Time (sec.)  Train_Accuracy  Test_Accuracy  \
     0    Torch LogReg           677.2053        0.812232       0.800950
     1  sklearn LogReg           443.0372        0.802343       0.802106
     2   Random Forest            25.4343        0.978734       0.795003

        Train_Recall  Test_Recall  Train_Precision  Test_Precision  Train_F1-Score  \
     0      0.170373     0.143385         0.617422        0.513562        0.267054
     1      0.097300     0.095947         0.543423        0.537341        0.165049
     2      0.895096     0.094162         0.998867        0.447528        0.944139

        Test_F1-Score
     0       0.224180
     1       0.162821
     2       0.155588
```

[ ]:
```
keep_ft_num = 25

# Get feature importances
importances = pd.Series(clf_rf.feature_importances_, index=X_train.columns)

# Sort importances
sorted_importances = importances.sort_values(ascending=True)

# Get the most important features
top_features = sorted_importances.head(keep_ft_num)

fig, ax = plt.subplots(figsize=(13,10))
ax.barh(top_features.index, top_features.values, color='lightblue')
ax.set_xlabel('Normalized Gini Importance')
ax.set_title('Feature Importance from DT')
plt.show()
```



[ ]:
```
# save the model to disk
pickle.dump(clf_rf, open('_pkls/rf_model.p', 'wb'))
```

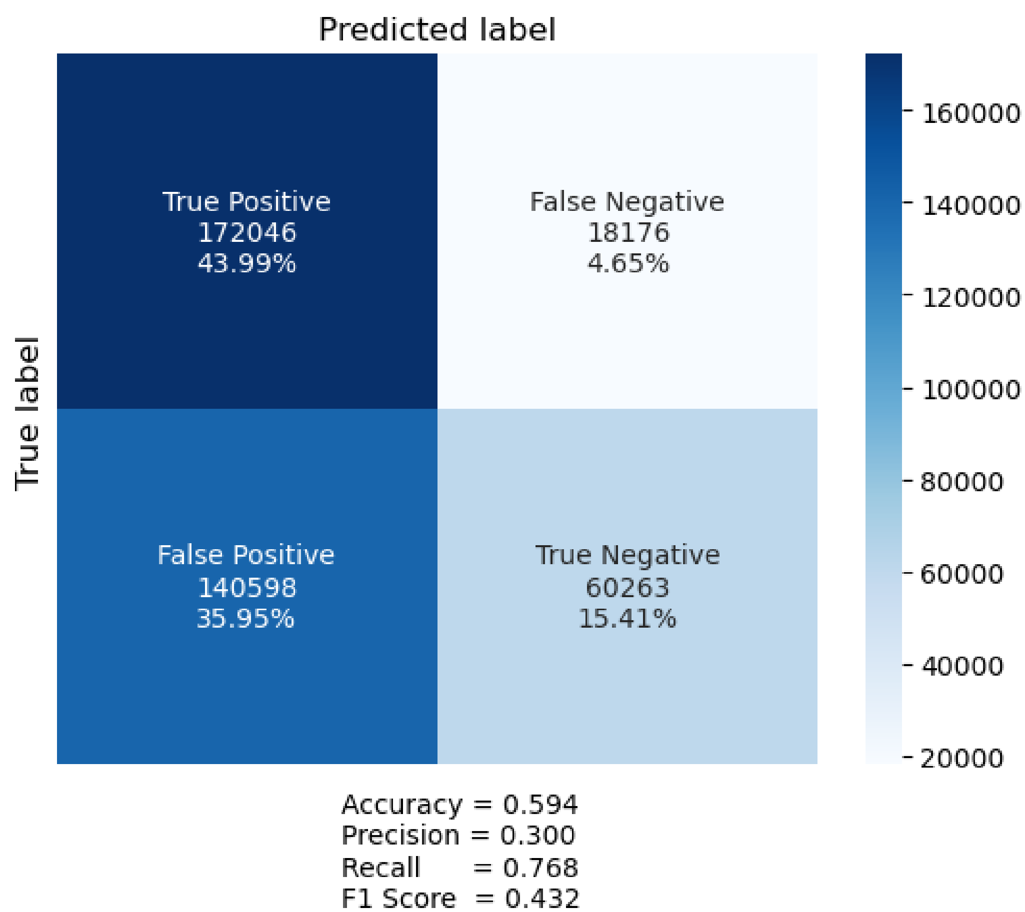
6 XGBoost

[ ]:
```
# Choose the type of classifier.
xgb_tuned = XGBClassifier(random_state=1, eval_metric='logloss')

# Grid of parameters to choose from
parameters = {
    "n_estimators": [10,30,50],
    "scale_pos_weight":[1,2,5],
    "subsample":[0.7,0.9,1],
    "learning_rate":[0.05, 0.1,0.2],
    "colsample_bytree":[0.7,0.9,1],
    "colsample_bylevel":[0.5,0.7,1]
}

# Type of scoring used to compare parameter combinations
scorer = make_scorer(f1_score)

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters, scoring=scorer, cv=5)

start = time.perf_counter()
grid_obj = grid_obj.fit(X_train, y_train)
end = time.perf_counter()

# Set the clf to the best combination of parameters
xgb_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
xgb_tuned.fit(X_train, y_train)
```

[ ]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=1, colsample_bynode=None, colsample_bytree=0.9,
              device=None, early_stopping_rounds=None, enable_categorical=False,
              eval_metric='logloss', feature_types=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.2, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=50, n_jobs=None,
              num_parallel_tree=None, random_state=1, …)
```

[ ]:
```
# Generate predictions
pred_train = xgb_tuned.predict(X_train)
pred_test = xgb_tuned.predict(X_test)
```

```
#Creating confusion matrix
make_cm([y_test,pred_test])

#Models and sample dicts
models_dict['XGBoost Tuned'] = {'pred_train':pred_train, 'pred_test':pred_test,
 ↪'est_time':end-start}

#Print model comparisons
model_comparisons(models_dict, samples_dict)
```

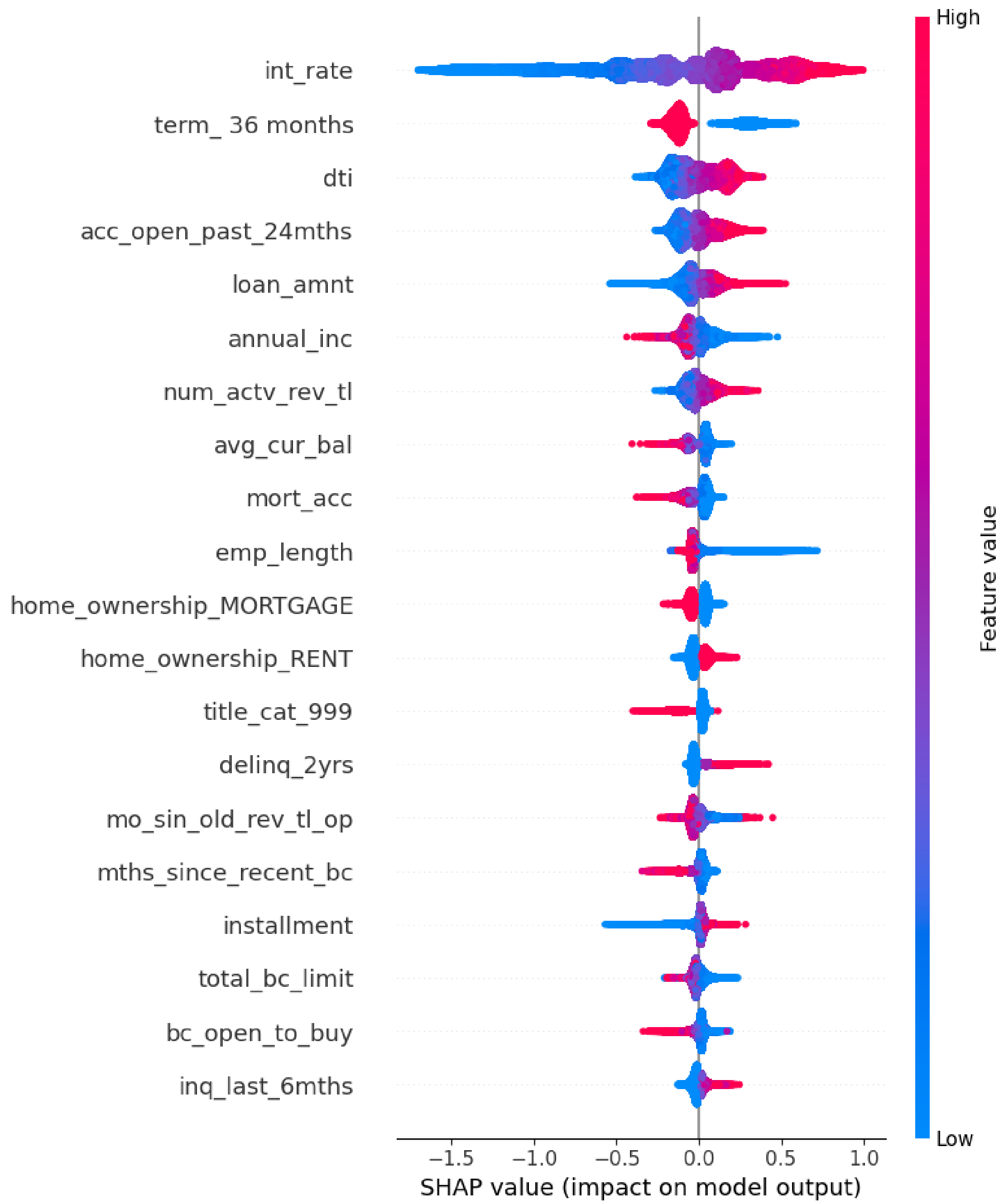


[ ]:
```
              Model  Conv. Time (sec.)  Train_Accuracy  Test_Accuracy  \
0     Torch LogReg           677.2053        0.812232       0.800950
1   sklearn LogReg           443.0372        0.802343       0.802106
2    Random Forest            25.4343        0.978734       0.795003
3    XGBoost Tuned          6340.4054        0.599231       0.594015

   Train_Recall  Test_Recall  Train_Precision  Test_Precision  Train_F1-Score  \
0      0.170373     0.143385         0.617422        0.513562        0.267054
```

```
1      0.097300     0.095947          0.543423          0.537341          0.165049
2      0.895096     0.094162          0.998867          0.447528          0.944139
3      0.782508     0.768279          0.305538          0.300023          0.439478

   Test_F1-Score
0       0.224180
1       0.162821
2       0.155588
3       0.431529
```

[ ]:

```
explainer = shap.TreeExplainer(xgb_tuned)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



[ ]:
```
# save the model to disk
pickle.dump(xgb_tuned, open('_pkls/xgb_model.p', 'wb'))
```