Group 15 Assignment1

Nayeon Kim 7432471

Wong Kia Yi Edric 7895677

Patel Mihir Mukeshbhai 7350314

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import *
from sklearn.preprocessing import *
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

Import data

```
#change to your local file path
df = pd.read_csv("data.csv", low_memory=False) #set to low memeory
because of mixed data type

df
```

```
               id  member_id  loan_amnt  funded_amnt  
funded_amnt_inv  \
0         1077501    1296599       5000         5000           4975.0
1         1077430    1314167       2500         2500           2500.0
2         1077175    1313524       2400         2400           2400.0
3         1076863    1277178      10000        10000          10000.0
4         1075358    1311748       3000         3000           3000.0
...           ...        ...        ...          ...              ...
855964   36371250   39102635      10000        10000          10000.0
855965   36441262   39152692      24000        24000          24000.0
855966   36271333   38982739      13000        13000          13000.0
855967   36490806   39222577      12000        12000          12000.0
855968   36271262   38982659      20000        20000          20000.0
```

```
              term  int_rate  installment grade sub_grade  ... il_util  \
0        36 months     10.65       162.87     B        B2  ...     NaN
1        60 months     15.27        59.83     C        C4  ...     NaN
2        36 months     15.96        84.33     C        C5  ...     NaN
3        36 months     13.49       339.31     C        C1  ...     NaN
4        60 months     12.69        67.79     B        B5  ...     NaN
...            ...       ...          ...   ...       ...  ...     ...
855964   36 months     11.99       332.10     B        B5  ...     NaN
855965   36 months     11.99       797.03     B        B5  ...     NaN
855966   60 months     15.99       316.07     D        D2  ...     NaN
855967   60 months     19.99       317.86     E        E3  ...     NaN
855968   36 months     11.99       664.20     B        B5  ...     NaN

       open_rv_12m open_rv_24m  max_bal_bc all_util total_rev_hi_lim
inq_fi  \
0              NaN         NaN         NaN      NaN              NaN
NaN
1              NaN         NaN         NaN      NaN              NaN
NaN
2              NaN         NaN         NaN      NaN              NaN
NaN
3              NaN         NaN         NaN      NaN              NaN
NaN
4              NaN         NaN         NaN      NaN              NaN
NaN
...            ...         ...         ...      ...              ...
...
855964         NaN         NaN         NaN      NaN          17100.0
NaN
855965         NaN         NaN         NaN      NaN          10200.0
NaN
855966         NaN         NaN         NaN      NaN          18000.0
NaN
855967         NaN         NaN         NaN      NaN          27000.0
NaN
855968         NaN         NaN         NaN      NaN          41700.0
NaN
```

```
        total_cu_tl inq_last_12m default_ind
0               NaN          NaN            0
1               NaN          NaN            1
2               NaN          NaN            0
3               NaN          NaN            0
4               NaN          NaN            0
...             ...          ...          ...
855964          NaN          NaN            0
855965          NaN          NaN            0
855966          NaN          NaN            0
855967          NaN          NaN            0
855968          NaN          NaN            0

[855969 rows x 73 columns]
```

EDA

check data types of each column

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 855969 entries, 0 to 855968
Data columns (total 73 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   id                           855969 non-null  int64
 1   member_id                    855969 non-null  int64
 2   loan_amnt                    855969 non-null  int64
 3   funded_amnt                  855969 non-null  int64
 4   funded_amnt_inv              855969 non-null  float64
 5   term                         855969 non-null  object
 6   int_rate                     855969 non-null  float64
 7   installment                  855969 non-null  float64
 8   grade                        855969 non-null  object
 9   sub_grade                    855969 non-null  object
 10  emp_title                    806526 non-null  object
 11  emp_length                   812908 non-null  object
 12  home_ownership               855969 non-null  object
 13  annual_inc                   855969 non-null  float64
 14  verification_status          855969 non-null  object
 15  issue_d                      855969 non-null  object
 16  pymnt_plan                   855969 non-null  object
 17  desc                         121812 non-null  object
 18  purpose                      855969 non-null  object
 19  title                        855936 non-null  object
 20  zip_code                     855969 non-null  object
 21  addr_state                   855969 non-null  object
 22  dti                          855969 non-null  float64
```

```
 23  delinq_2yrs                  855969 non-null  int64
 24  earliest_cr_line             855969 non-null  object
 25  inq_last_6mths               855969 non-null  int64
 26  mths_since_last_delinq       416157 non-null  float64
 27  mths_since_last_record       131184 non-null  float64
 28  open_acc                     855969 non-null  int64
 29  pub_rec                      855969 non-null  int64
 30  revol_bal                    855969 non-null  int64
 31  revol_util                   855523 non-null  float64
 32  total_acc                    855969 non-null  int64
 33  initial_list_status          855969 non-null  object
 34  out_prncp                    855969 non-null  float64
 35  out_prncp_inv                855969 non-null  float64
 36  total_pymnt                  855969 non-null  float64
 37  total_pymnt_inv              855969 non-null  float64
 38  total_rec_prncp              855969 non-null  float64
 39  total_rec_int                855969 non-null  float64
 40  total_rec_late_fee           855969 non-null  float64
 41  recoveries                   855969 non-null  float64
 42  collection_recovery_fee      855969 non-null  float64
 43  last_pymnt_d                 847107 non-null  object
 44  last_pymnt_amnt              855969 non-null  float64
 45  next_pymnt_d                 602998 non-null  object
 46  last_credit_pull_d           855919 non-null  object
 47  collections_12_mths_ex_med   855913 non-null  float64
 48  mths_since_last_major_derog  213139 non-null  float64
 49  policy_code                  855969 non-null  int64
 50  application_type             855969 non-null  object
 51  annual_inc_joint             442 non-null     float64
 52  dti_joint                    442 non-null     float64
 53  verification_status_joint    442 non-null     object
 54  acc_now_delinq               855969 non-null  int64
 55  tot_coll_amt                 788656 non-null  float64
 56  tot_cur_bal                  788656 non-null  float64
 57  open_acc_6m                  13288 non-null   float64
 58  open_il_6m                   13288 non-null   float64
 59  open_il_12m                  13288 non-null   float64
 60  open_il_24m                  13288 non-null   float64
 61  mths_since_rcnt_il           12934 non-null   float64
 62  total_bal_il                 13288 non-null   float64
 63  il_util                      11609 non-null   float64
 64  open_rv_12m                  13288 non-null   float64
 65  open_rv_24m                  13288 non-null   float64
 66  max_bal_bc                   13288 non-null   float64
 67  all_util                     13288 non-null   float64
 68  total_rev_hi_lim             788656 non-null  float64
 69  inq_fi                       13288 non-null   float64
 70  total_cu_tl                  13288 non-null   float64
 71  inq_last_12m                 13288 non-null   float64
```

```
 72  default_ind                  855969 non-null  int64
dtypes: float64(39), int64(13), object(21)
memory usage: 476.7+ MB
```

```
df.isna().sum() #check the number of null values per column
```

```
id                         0
member_id                  0
loan_amnt                  0
funded_amnt                0
funded_amnt_inv            0
                        ...
total_rev_hi_lim       67313
inq_fi                842681
total_cu_tl           842681
inq_last_12m          842681
default_ind                0
Length: 73, dtype: int64
```

There are too many rows with nan values, so the dataset becomes irrelevant after dropping all the nan values. we shall find another method to fill them in.

Data Preprocessing - Prepare data for ml

1. Drop irrelevant columns
2. Use pipeline to fill in the missing values, encode categorical values, and scale the data
3. split the data for training and testing
4. User-defined feature

```
df = df.drop(columns=['sub_grade', 'loan_amnt', 'funded_amnt_inv',
'out_prncp_inv', 'total_pymnt_inv', 'member_id', 'emp_length', 'desc',
'emp_title', 'zip_code'], axis=1) #dropped columns that are redundant
or not related to the target

df
```

```
              id  funded_amnt        term  int_rate  installment grade  \
0        1077501         5000   36 months     10.65       162.87     B

1        1077430         2500   60 months     15.27        59.83     C

2        1077175         2400   36 months     15.96        84.33     C

3        1076863        10000   36 months     13.49       339.31     C

4        1075358         3000   60 months     12.69        67.79     B

...          ...          ...         ...       ...          ...   ...

855964  36371250        10000   36 months     11.99       332.10     B
```

```
855965  36441262        24000   36 months     11.99        797.03     B
855966  36271333        13000   60 months     15.99        316.07     D
855967  36490806        12000   60 months     19.99        317.86     E
855968  36271262        20000   36 months     11.99        664.20     B

       home_ownership  annual_inc verification_status     issue_d  ...
\
0                RENT     24000.0            Verified  01-12-2011  ...
1                RENT     30000.0     Source Verified  01-12-2011  ...
2                RENT     12252.0        Not Verified  01-12-2011  ...
3                RENT     49200.0     Source Verified  01-12-2011  ...
4                RENT     80000.0     Source Verified  01-12-2011  ...
...               ...         ...                 ...         ...  ...
855964           RENT     31000.0            Verified  01-01-2015  ...
855965       MORTGAGE     79000.0            Verified  01-01-2015  ...
855966           RENT     35000.0            Verified  01-01-2015  ...
855967           RENT     64400.0     Source Verified  01-01-2015  ...
855968           RENT    100000.0            Verified  01-01-2015  ...

       il_util open_rv_12m open_rv_24m max_bal_bc  all_util
total_rev_hi_lim  \
0          NaN         NaN         NaN        NaN       NaN
NaN
1          NaN         NaN         NaN        NaN       NaN
NaN
2          NaN         NaN         NaN        NaN       NaN
NaN
3          NaN         NaN         NaN        NaN       NaN
NaN
4          NaN         NaN         NaN        NaN       NaN
NaN
...        ...         ...         ...        ...       ...
...
855964     NaN         NaN         NaN        NaN       NaN
17100.0
```

```
855965      NaN          NaN          NaN         NaN        NaN
10200.0
855966      NaN          NaN          NaN         NaN        NaN
18000.0
855967      NaN          NaN          NaN         NaN        NaN
27000.0
855968      NaN          NaN          NaN         NaN        NaN
41700.0

       inq_fi  total_cu_tl  inq_last_12m  default_ind
0         NaN          NaN           NaN            0
1         NaN          NaN           NaN            1
2         NaN          NaN           NaN            0
3         NaN          NaN           NaN            0
4         NaN          NaN           NaN            0
...       ...          ...           ...          ...
855964    NaN          NaN           NaN            0
855965    NaN          NaN           NaN            0
855966    NaN          NaN           NaN            0
855967    NaN          NaN           NaN            0
855968    NaN          NaN           NaN            0

[855969 rows x 63 columns]
```

Pipeline to preprocess numerical columns

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
df1 = df.loc[:, df.columns != "default_ind"] #exclude target
df2 = df1.loc[:, df1.columns != "id"]
df_numerics_only = df2.select_dtypes(include=np.number) #select
numerical columns

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')), #fill the nan values
with mean
    ('std_scaler', StandardScaler()) #scale the range of data
])

num_transformed = num_pipeline.fit_transform(df_numerics_only)

num_transformed

array([[-1.15593762e+00, -5.81984560e-01, -1.12161710e+00, ...,
         9.88964671e-15,  3.30330006e-15,  8.38667283e-15],
       [-1.45286854e+00,  4.75619945e-01, -1.54438566e+00, ...,
         9.88964671e-15,  3.30330006e-15,  8.38667283e-15],
       [-1.46474577e+00,  6.33573865e-01, -1.44386324e+00, ...,
         9.88964671e-15,  3.30330006e-15,  8.38667283e-15],
       ...,
```

```
       [-2.05758673e-01,  6.40441427e-01, -4.93044279e-01, ...,
         9.88964671e-15,  3.30330006e-15,  8.38667283e-15],
       [-3.24531041e-01,  1.55611632e+00, -4.85699989e-01, ...,
         9.88964671e-15,  3.30330006e-15,  8.38667283e-15],
       [ 6.25647902e-01, -2.75233470e-01,  9.35317700e-01, ...,
         9.88964671e-15,  3.30330006e-15,  8.38667283e-15]])
```

Pipleine to preprocess categorical values

```
df_non_numeric = df2.select_dtypes('object')

obj_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')), #fill in the nan values with most frequent values
    ('ordinal_encoder', OrdinalEncoder()) #Usually it is better to use one-hot encoder but ordinal encoder is used here since there are too many columns already
])

obj_transformed = obj_pipeline.fit_transform(df_non_numeric)

obj_transformed
```

```
array([[ 0.,  1.,  5., ...,  8.,  0.,  0.],
       [ 1.,  2.,  5., ..., 74.,  0.,  0.],
       [ 0.,  2.,  5., ...,  8.,  0.,  0.],
       ...,
       [ 1.,  3.,  5., ...,  8.,  0.,  0.],
       [ 1.,  4.,  5., ...,  8.,  0.,  0.],
       [ 0.,  1.,  5., ...,  8.,  0.,  0.]])
```

Now apply both pipelines together at the same time, to the dataframe

```
from sklearn.compose import ColumnTransformer

num_vars = []
cat_vars = []

for col in df2.columns:
  if df2[col].dtypes is np.number:
    num_vars.append(col)
  else:
    cat_vars.append(col)

data_pipeline = ColumnTransformer([ #apply to the whole data
    ('numerical', num_pipeline, num_vars),
    ('categorical', obj_pipeline, cat_vars)
])

df_processed = data_pipeline.fit_transform(df2)
```

Convert the data into dataframe

```
df_processed = pd.DataFrame(df_processed, columns=df2.columns)
#convert the processed data into dataframe

df_processed.sample(10)
```

```
        funded_amnt  term  int_rate  installment  grade
home_ownership  \
182704        543.0   0.0     146.0      35033.0    1.0
5.0
74997         319.0   0.0     110.0      18822.0    1.0
5.0
611680       1167.0   1.0      80.0      47171.0    1.0
1.0
74104         767.0   1.0     508.0      44269.0    6.0
5.0
320759        323.0   0.0     239.0      20432.0    2.0
1.0
348497        527.0   0.0     110.0      33297.0    1.0
4.0
620859        255.0   0.0     152.0      14771.0    2.0
1.0
677266        367.0   0.0      59.0      21509.0    1.0
4.0
779466        347.0   0.0      38.0      19634.0    0.0
4.0
497027        567.0   1.0      43.0      20356.0    1.0
5.0

        annual_inc  verification_status  issue_d  pymnt_plan  ...  \
182704     44891.0                  1.0     90.0         0.0  ...
74997       5579.0                  0.0     82.0         0.0  ...
611680     42385.0                  1.0     66.0         0.0  ...
74104      30953.0                  1.0     82.0         0.0  ...
320759      6534.0                  1.0     65.0         0.0  ...
348497     32224.0                  1.0     56.0         0.0  ...
620859      3486.0                  2.0     66.0         0.0  ...
677266     30953.0                  1.0     57.0         0.0  ...
779466     37186.0                  1.0     23.0         0.0  ...
497027     39528.0                  0.0     93.0         0.0  ...

        total_bal_il  il_util  open_rv_12m  open_rv_24m  max_bal_bc
all_util  \
182704           0.0    949.0          0.0          1.0         0.0
568.0
74997            0.0    949.0          0.0          1.0         0.0
568.0
611680           0.0    949.0          0.0          1.0         0.0
568.0
```

```
74104               0.0    949.0           0.0          1.0          0.0
568.0
320759              0.0    949.0           0.0          1.0          0.0
568.0
348497              0.0    949.0           0.0          1.0          0.0
568.0
620859              0.0    949.0           0.0          1.0          0.0
568.0
677266              0.0    949.0           0.0          1.0          0.0
568.0
779466              0.0    949.0           0.0          1.0          0.0
568.0
497027              0.0    949.0           0.0          1.0          0.0
568.0

        total_rev_hi_lim  inq_fi  total_cu_tl  inq_last_12m
182704            3980.0     0.0          0.0           2.0
74997             5885.0     0.0          0.0           2.0
611680           14755.0     0.0          0.0           2.0
74104             6326.0     0.0          0.0           2.0
320759            4795.0     0.0          0.0           2.0
348497             986.0     0.0          0.0           2.0
620859            5885.0     0.0          0.0           2.0
677266            3751.0     0.0          0.0           2.0
779466           12324.0     0.0          0.0           2.0
497027           12277.0     0.0          0.0           2.0

[10 rows x 61 columns]
```

```
df['default_ind'] #check that the target column has not been affected
by the preprocessing
```

```
0         0
1         1
2         0
3         0
4         0
         ..
855964    0
855965    0
855966    0
855967    0
855968    0
Name: default_ind, Length: 855969, dtype: int64
```

Append the preprocessed data with the id and target column

```
df_processed['default_ind'] = df['default_ind'] #append the target
column
```

```
df_processed
```

```
        funded_amnt  term  int_rate  installment  grade
home_ownership  \
0               167.0   0.0     101.0       8723.0    1.0
5.0
1                67.0   1.0     273.0       1565.0    2.0
5.0
2                63.0   0.0     296.0       3098.0    2.0
5.0
3               367.0   0.0     197.0      23285.0    2.0
5.0
4                87.0   1.0     166.0       2049.0    1.0
5.0
...               ...   ...       ...          ...    ...               ..
.
855964          367.0   0.0     142.0      22657.0    1.0
5.0
855965          927.0   0.0     142.0      56733.0    1.0
1.0
855966          487.0   1.0     297.0      21272.0    3.0
5.0
855967          447.0   1.0     410.0      21432.0    4.0
5.0
855968          767.0   0.0     142.0      48994.0    1.0
5.0

        annual_inc  verification_status  issue_d  pymnt_plan  ...
il_util  \
0           3120.0                  2.0     98.0         0.0  ...
949.0
1           5579.0                  1.0     98.0         0.0  ...
949.0
2            410.0                  0.0     98.0         0.0  ...
949.0
3          16788.0                  1.0     98.0         0.0  ...
949.0
4          33015.0                  1.0     98.0         0.0  ...
949.0
...            ...                  ...      ...         ...  ...
...
855964      6038.0                  2.0      7.0         0.0  ...
949.0
855965     32634.0                  2.0      7.0         0.0  ...
949.0
855966      8091.0                  2.0      7.0         0.0  ...
949.0
855967     25790.0                  1.0      7.0         0.0  ...
949.0
855968     39528.0                  2.0      7.0         0.0  ...
```

```
949.0

         open_rv_12m  open_rv_24m  max_bal_bc  all_util
total_rev_hi_lim  \
0                0.0          1.0         0.0     568.0
2484.0
1                0.0          1.0         0.0     568.0
2484.0
2                0.0          1.0         0.0     568.0
2484.0
3                0.0          1.0         0.0     568.0
2484.0
4                0.0          1.0         0.0     568.0
2484.0
...              ...          ...         ...       ...                .
..
855964           0.0          1.0         0.0     568.0
3078.0
855965           0.0          1.0         0.0     568.0
1235.0
855966           0.0          1.0         0.0     568.0
3333.0
855967           0.0          1.0         0.0     568.0
5967.0
855968           0.0          1.0         0.0     568.0
9747.0

         inq_fi  total_cu_tl  inq_last_12m  default_ind
0           0.0          0.0           2.0            0
1           0.0          0.0           2.0            1
2           0.0          0.0           2.0            0
3           0.0          0.0           2.0            0
4           0.0          0.0           2.0            0
...         ...          ...           ...          ...
855964      0.0          0.0           2.0            0
855965      0.0          0.0           2.0            0
855966      0.0          0.0           2.0            0
855967      0.0          0.0           2.0            0
855968      0.0          0.0           2.0            0

[855969 rows x 62 columns]
```

Add User defined functionality

```
df_processed['installment']
```

```
0         8723.0
1         1565.0
2         3098.0
```

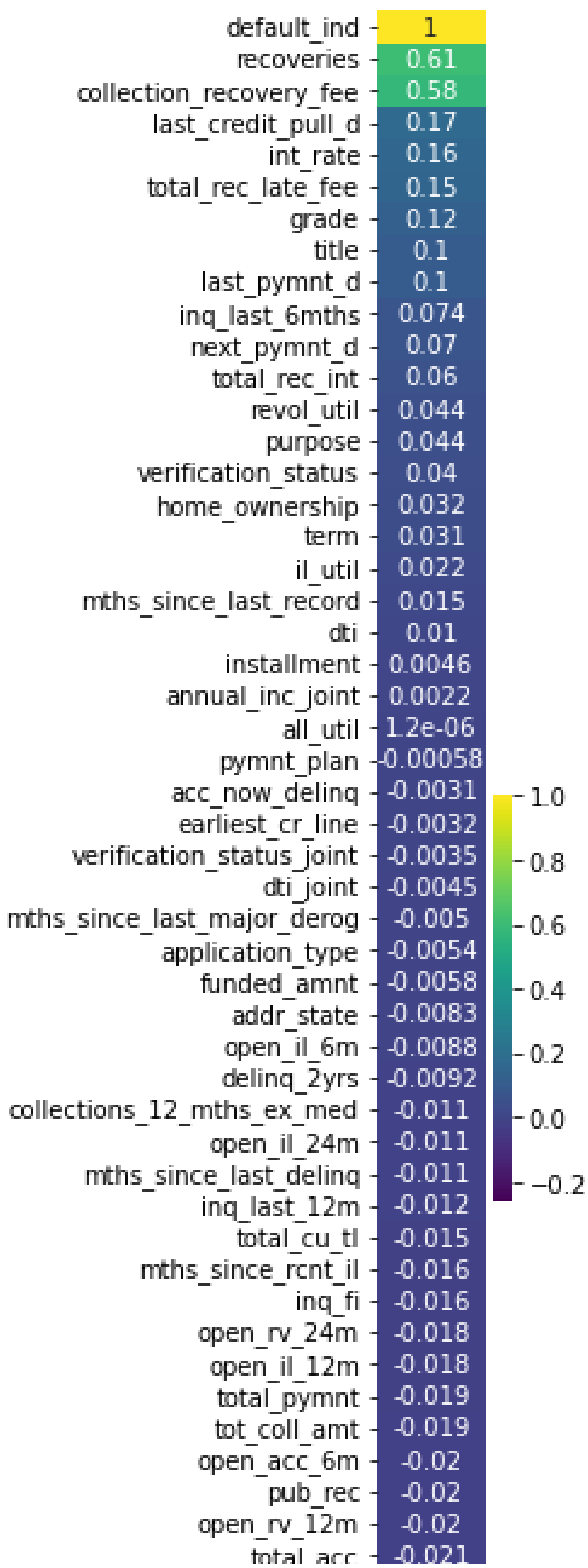
```
3         23285.0
4          2049.0
            ...
855964    22657.0
855965    56733.0
855966    21272.0
855967    21432.0
855968    48994.0
Name: installment, Length: 855969, dtype: float64
```

User defined transformer - cut the installment column into bins, label them

```
from sklearn.base import BaseEstimator, TransformerMixin

class UserDefinedTransform(BaseEstimator, TransformerMixin):
    def __init__(self, installment_cut=True):
        self.installment_cut = installment_cut

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        data = X.copy()  # Make a copy of the DataFrame to avoid 
modifying the original data
        if self.installment_cut:
            data['installment_cut'] = pd.cut(data['installment'], 5, 
precision=0, labels=[1, 2, 3, 4, 5]) #binning the data with labels
            installment_cutted = data.pop('installment_cut')
            data.insert(1, 'installment_cut', installment_cutted)
            return data.values  # Return transformed values as a numpy 
array
        else:
            raise ValueError("Invalid value for installment_cut")
```

Feature Analysis

```
coor_matrix = df_processed.corr()
["default_ind"].sort_values(ascending=False)

plt.figure(figsize=(1,15))
sns.heatmap(coor_matrix.to_frame(), annot=True, cmap='viridis')

<AxesSubplot:>
```

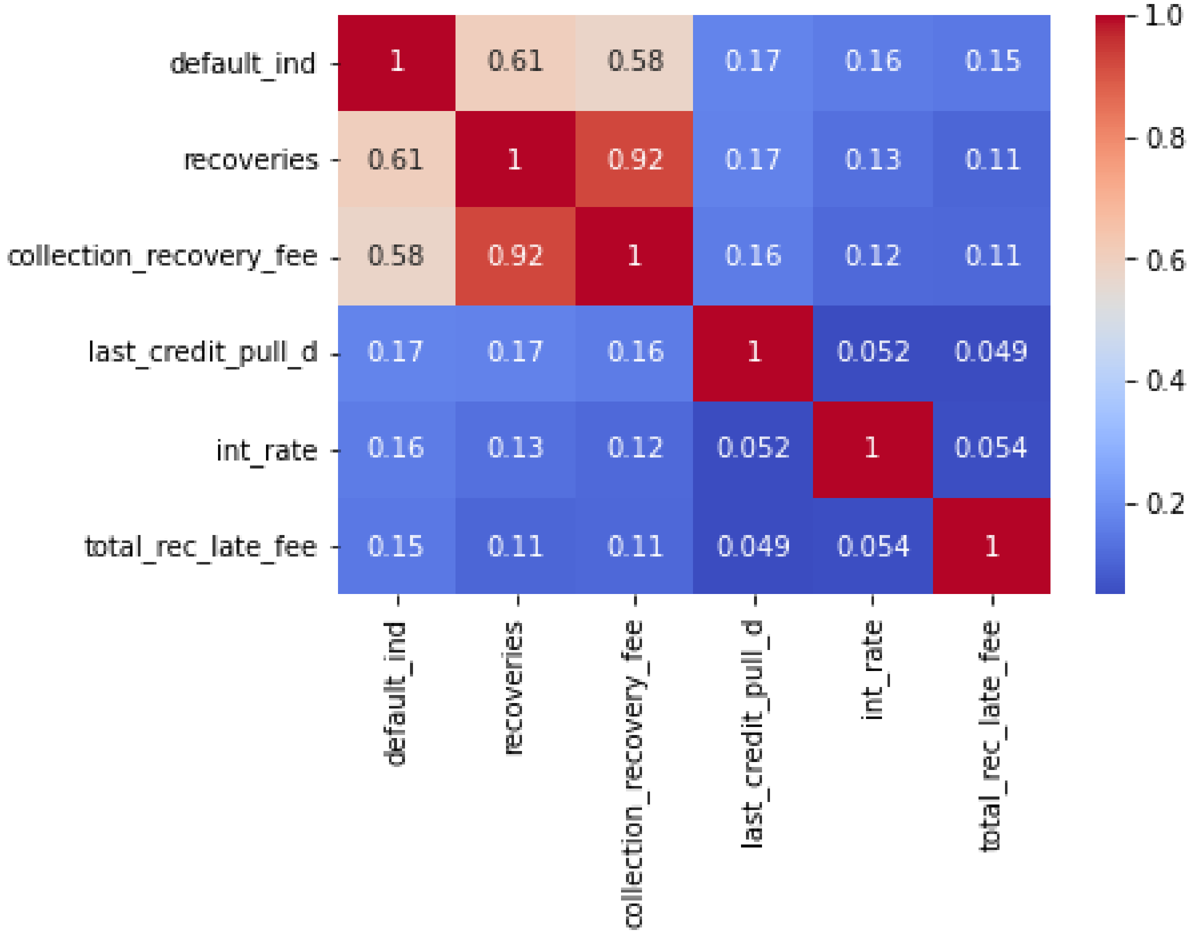

default_ind 1
recoveries 0.61
collection_recovery_fee 0.58
last_credit_pull_d 0.17
int_rate 0.16
total_rec_late_fee 0.15
grade 0.12
title 0.1
last_pymnt_d 0.1
inq_last_6mths 0.074
next_pymnt_d 0.07
total_rec_int 0.06
revol_util 0.044
purpose 0.044
verification_status 0.04
home_ownership 0.032
term 0.031
il_util 0.022
mths_since_last_record 0.015
dti 0.01
installment 0.0046
annual_inc_joint 0.0022
all_util 1.2e-06
pymnt_plan -0.00058
acc_now_delinq -0.0031
earliest_cr_line -0.0032
verification_status_joint -0.0035
dti_joint -0.0045
mths_since_last_major_derog -0.005
application_type -0.0054
funded_amnt -0.0058
addr_state -0.0083
open_il_6m -0.0088
delinq_2yrs -0.0092
collections_12_mths_ex_med -0.011
open_il_24m -0.011
mths_since_last_delinq -0.011
inq_last_12m -0.012
total_cu_tl -0.015
mths_since_rcnt_il -0.016
inq_fi -0.016
open_rv_24m -0.018
open_il_12m -0.018
total_pymnt -0.019
tot_coll_amt -0.019
open_acc_6m -0.02
pub_rec -0.02
open_rv_12m -0.02
total_acc -0.021
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

The most correlated features to the target is recovery. Now let's see how top features are correlated in a more visualised way:

```
df_processed.corr()

                  funded_amnt      term  int_rate  installment
grade  \
funded_amnt          1.000000  0.410210  0.144083     0.950618
0.150698
term                 0.410210  1.000000  0.426084     0.183292
0.444810
int_rate             0.144083  0.426084  1.000000     0.126956
0.950364
installment          0.950618  0.183292  0.126956     1.000000
0.124015
grade                0.150698  0.444810  0.950364     0.124015
1.000000
...                       ...       ...       ...          ...        .
..
total_rev_hi_lim     0.451238  0.121689 -0.202455     0.415132 -
0.189305
inq_fi               0.001772  0.006155 -0.000041    -0.000753
0.011782
total_cu_tl          0.008580  0.008354 -0.013234     0.004172 -
0.003028
inq_last_12m         0.000854  0.004794  0.015661     0.000943
0.023973
default_ind         -0.005797  0.031378  0.155715     0.004626
0.123656

                  home_ownership  annual_inc  verification_status
issue_d  \
funded_amnt            -0.196353    0.521481             0.281071
0.000670
term                   -0.111044    0.136691             0.168264
0.014674
int_rate                0.060043   -0.110275             0.249019 -
0.039858
installment            -0.168326    0.499312             0.268453 -
0.011532
grade                   0.060878   -0.101486             0.230367 -
0.012605
...                          ...         ...                  ...
...
total_rev_hi_lim       -0.208945    0.405093             0.069305
0.039454
inq_fi                 -0.005447    0.014358             0.001360
0.106372
total_cu_tl            -0.009782    0.017103            -0.005322
0.095482
```

```
inq_last_12m          -0.007560    0.014425             0.005725
0.077456
default_ind            0.032341   -0.059897             0.039585 -
0.045891

                  pymnt_plan  ...   il_util  open_rv_12m  open_rv_24m
\
funded_amnt         0.000766  ... -0.010328    -0.004448    -0.002454

term               -0.000522  ... -0.001689    -0.000706     0.000917

int_rate            0.000951  ...  0.033563    -0.006659    -0.004102

installment         0.001370  ... -0.005837    -0.005718    -0.003341

grade               0.000782  ...  0.018016     0.007642     0.008588

...                      ...  ...       ...          ...          ...

total_rev_hi_lim   -0.000243  ... -0.023207     0.025017     0.029564

inq_fi             -0.000166  ... -0.346945     0.427890     0.413296

total_cu_tl        -0.000149  ... -0.378638     0.334350     0.303851

inq_last_12m       -0.000121  ... -0.222719     0.484951     0.463706

default_ind        -0.000579  ...  0.022020    -0.020272    -0.018027


                  max_bal_bc  all_util  total_rev_hi_lim    inq_fi  \
funded_amnt         0.031233  0.000325          0.451238  0.001772
term                0.010952  0.009877          0.121689  0.006155
int_rate           -0.033945  0.032905         -0.202455 -0.000041
installment         0.024787  0.000790          0.415132 -0.000753
grade              -0.016127  0.033991         -0.189305  0.011782
...                      ...       ...               ...       ...
total_rev_hi_lim    0.056499 -0.040921          1.000000  0.003798
inq_fi              0.412964  0.086995          0.003798  1.000000
total_cu_tl         0.400182  0.032970          0.016834  0.326358
inq_last_12m        0.273350  0.044813          0.004746  0.649022
default_ind        -0.025397  0.000001         -0.065545 -0.016437

                  total_cu_tl  inq_last_12m  default_ind
funded_amnt          0.008580      0.000854    -0.005797
term                 0.008354      0.004794     0.031378
int_rate            -0.013234      0.015661     0.155715
installment          0.004172      0.000943     0.004626
grade               -0.003028      0.023973     0.123656
...                       ...           ...          ...
total_rev_hi_lim     0.016834      0.004746    -0.065545
```

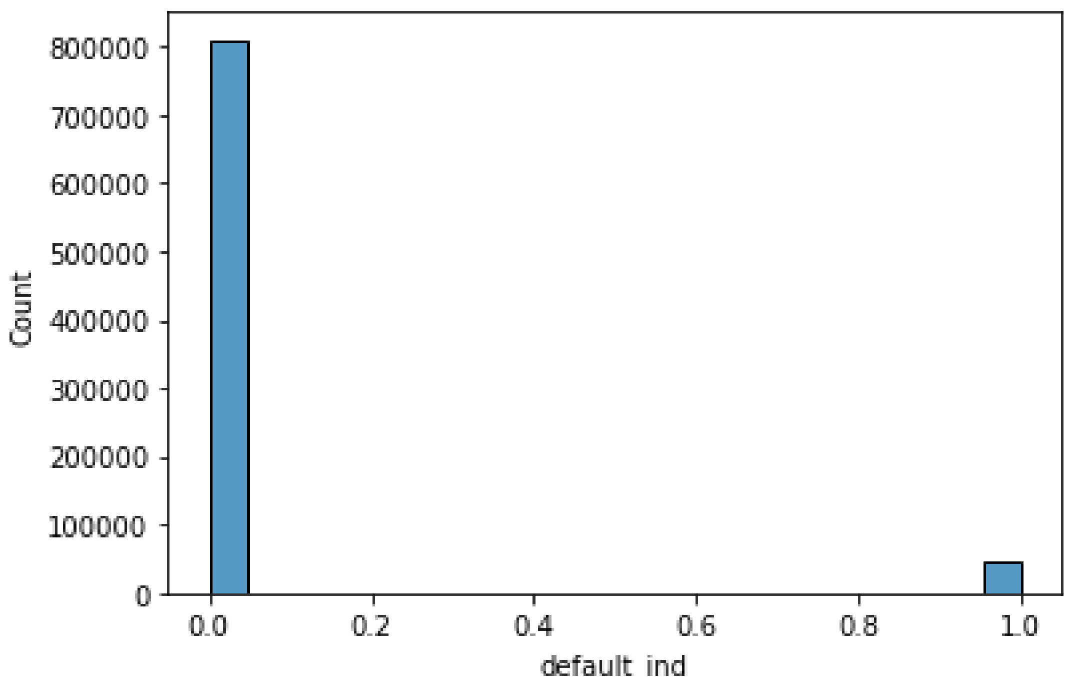
```
inq_fi                    0.326358      0.649022    -0.016437
total_cu_tl               1.000000      0.243566    -0.014755
inq_last_12m              0.243566      1.000000    -0.011969
default_ind              -0.014755     -0.011969     1.000000

[62 rows x 62 columns]

correlation_matrix = df_processed.corr()
top_corr_features = correlation_matrix.nlargest(6,
'default_ind').drop('default_ind', axis=1).index
correlation_matrix_top5 = df_processed[top_corr_features].corr()
g = sns.heatmap(correlation_matrix_top5, annot=True, cmap="coolwarm")
```



Data Visualization

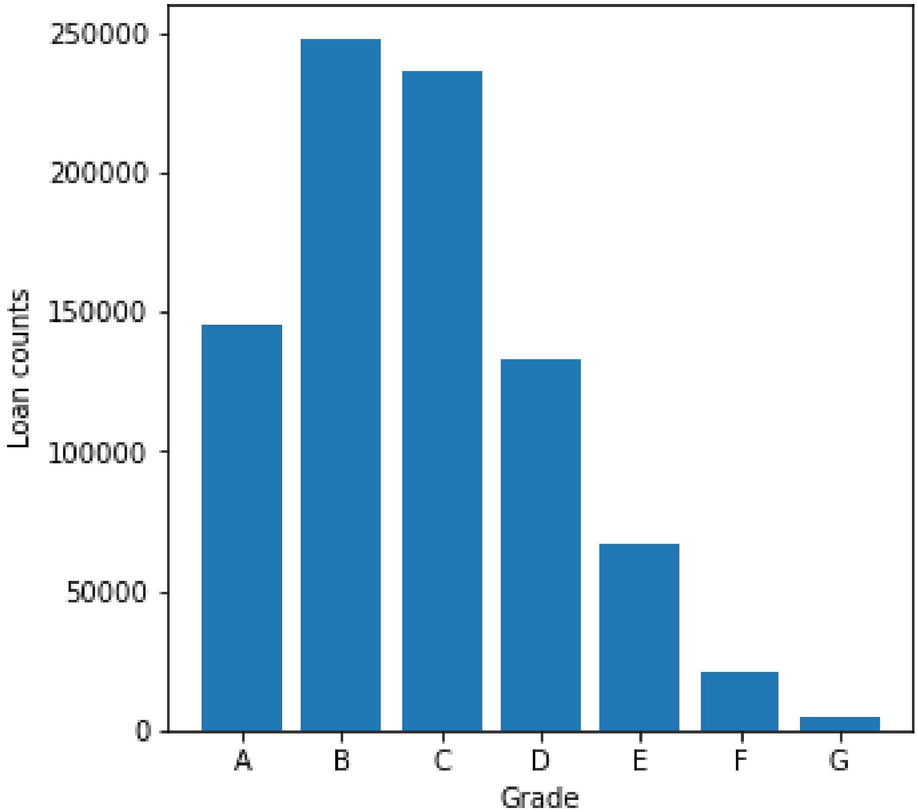
Target column distribution: we can see that most of the values are 0

```
sns.histplot(df_processed['default_ind'])

<AxesSubplot:xlabel='default_ind', ylabel='Count'>
```



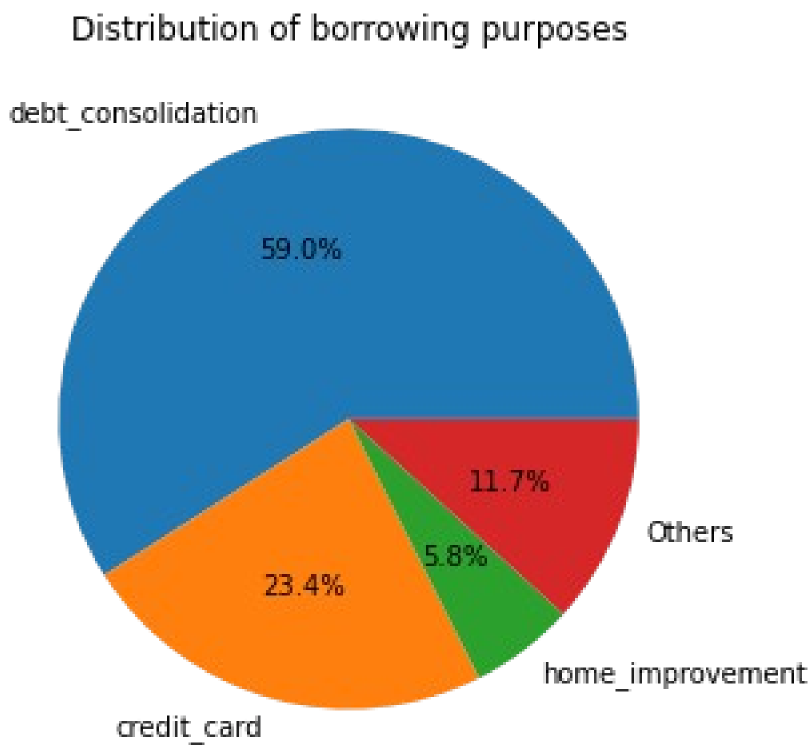
Organization of data on grade

```
grades = df['grade']
grade_counts = grades.value_counts()
sorted_grades = grade_counts.sort_index()

plt.figure(figsize=(5,5))
plt.bar(sorted_grades.index, sorted_grades)
plt.xlabel('Grade')
plt.ylabel('Loan counts')

plt.show()
```



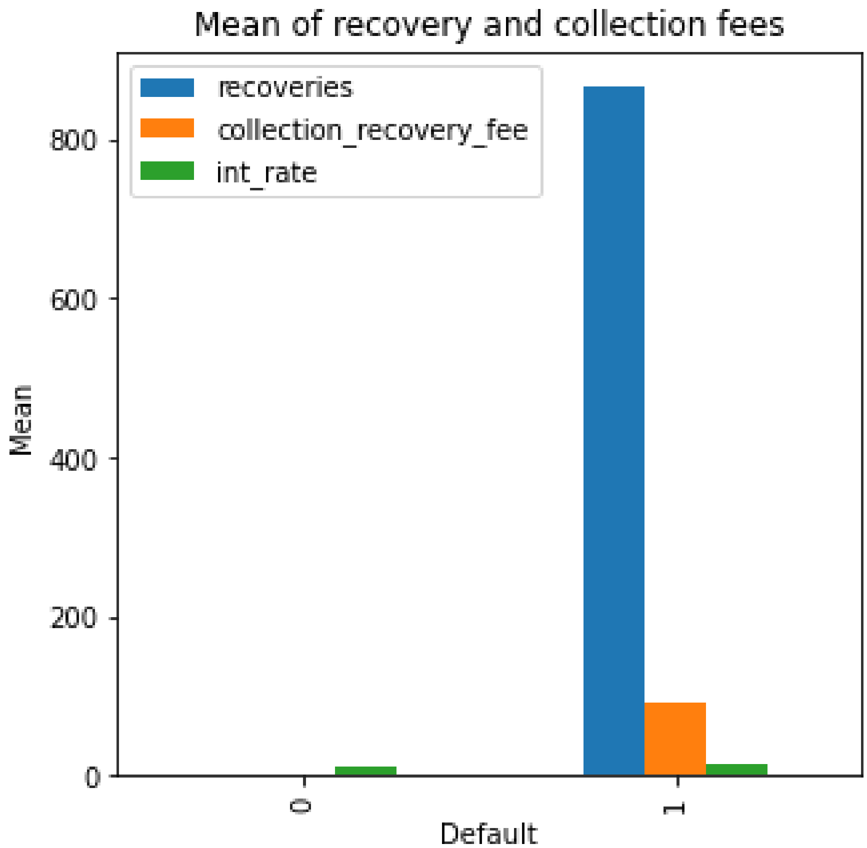
Organization of data on borrowing purposes

```
purposes = df['purpose']
purpose_count = df['purpose'].value_counts()
threshold = 0.05 # %5

others = purpose_count[purpose_count / purpose_count.sum() <
threshold]
others_count = others.sum()
purpose_count = purpose_count.drop(others.index)
purpose_count['Others'] = others_count


plt.figure(figsize=(5,5))
plt.pie(purpose_count,labels=purpose_count.index, autopct='%1.1f%%')
plt.title('Distribution of borrowing purposes')


plt.show()
```



Distribution of recoveries and collection recovery fees on default

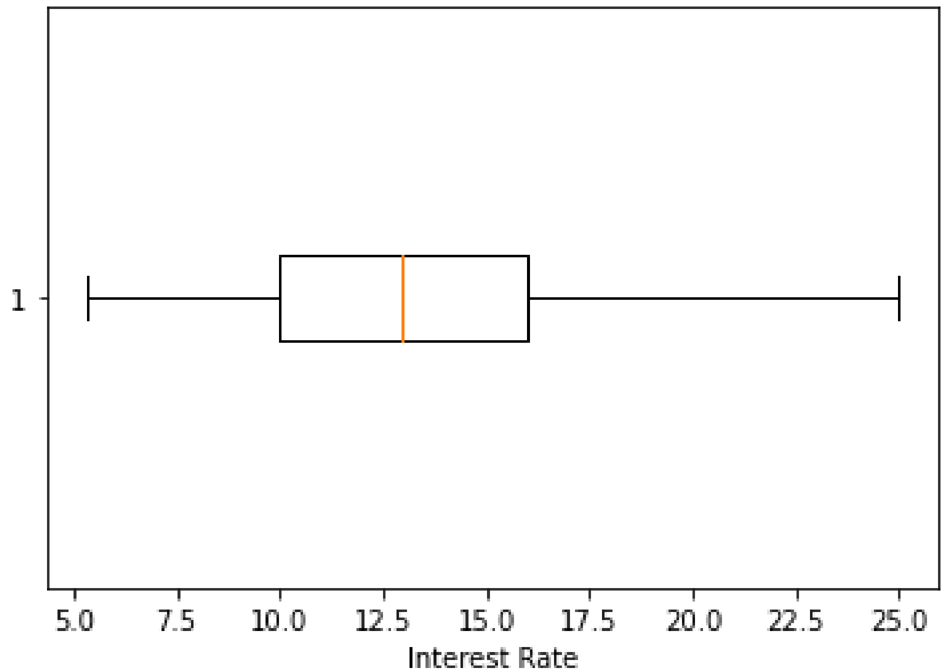
```
df3 = df[['default_ind','recoveries','collection_recovery_fee',
'int_rate']].groupby('default_ind').mean()

df3.plot(kind='bar', figsize=(5,5), xlabel='Default', ylabel='Mean',
title='Mean of recovery and collection fees')

<AxesSubplot:title={'center':'Mean of recovery and collection fees'},
xlabel='Default', ylabel='Mean'>
```



Interest Rate Info

```
print(df['int_rate'].describe())

plt.boxplot(df['int_rate'], showfliers=False, meanline=True,
vert=False)
plt.xlabel('Interest Rate')
plt.show()
```

```
count    855969.000000
mean         13.192320
std           4.368365
min           5.320000
25%           9.990000
50%          12.990000
75%          15.990000
max          28.990000
Name: int_rate, dtype: float64
```



Prepare data for ml

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

X = df_processed.loc[:, df_processed.columns!="default_ind"]
y = df_processed.loc[:, "default_ind"]
```

Split the dataset using stratified sampling by setting stratify=y

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

attr_adder = UserDefinedTransform()
```

Prepare stratified kfolds for cross validation later

```
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold()
skf.get_n_splits(X, y)

5

from sklearn.metrics import accuracy_score
```

Model 1: Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(random_state=0)
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)

from sklearn.metrics import accuracy_score

print('Model accuracy score with 10 decision-trees : {0:0.4f}'.
format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with 10 decision-trees : 0.9987
```

```
from sklearn.metrics import
roc_auc_score,roc_curve,classification_report,confusion_matrix,Confusi
onMatrixDisplay
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    161901
           1       1.00      0.98      0.99      9293

    accuracy                           1.00    171194
   macro avg       1.00      0.99      0.99    171194
weighted avg       1.00      1.00      1.00    171194
```

Both precision and recall scores are high, and the difference between them is not big as we can see from the f1 score.

The accuracy is way too high - validate with cross validation again

```
from sklearn.model_selection import cross_val_score
rf = RandomForestClassifier(random_state=0)
cvs = cross_val_score(rf, X, y, scoring=None, cv=skf)

cvs
```

```
array([0.99789128, 0.99832938, 0.94448988, 0.99905371, 0.99799641])
```

```
np.mean(cvs)
```

```
0.9875521314957456
```

The accuracy from cross validation is once again quiet high, from the fact that i am using stratified k fold method, the model is quiet accurate for sure

```
print('Training set score: {:.4f}'.format(rfc.score(X_train,
y_train)))
```

```
print('Test set score: {:.4f}'.format(rfc.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 0.9987
```

The difference between two is almost none, which shows that there is no overfitting issue.

Fine tune the model with GridSearchCV

Here I am using pipeline again to include the user-defined functionality as a hyperparameter.

```
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([
    ('user_transform', attr_adder),
    ('classifier', RandomForestClassifier(random_state=0)),
])

param_grid = {
    'classifier__max_depth' : [38, 40],
    'user_transform__installment_cut': [True, False]
}

rfc = RandomForestClassifier(random_state=0)
CV_rfc = GridSearchCV(pipeline, param_grid, scoring='accuracy')
CV_rfc.fit(X_train, y_train)
```

```
GridSearchCV(estimator=Pipeline(steps=[('user_transform',
                                        UserDefinedTransform()),
                                       ('classifier',

RandomForestClassifier(random_state=0))]),
             param_grid={'classifier__max_depth': [38, 40],
                         'user_transform__installment_cut': [True,
False]},
             scoring='accuracy')
```

computed via cross validation

```
CV_rfc.best_params_
```

```
{'classifier__max_depth': 38, 'user_transform__installment_cut': True}
```

Fine tuned accuracy

```
print('Model accuracy score : {0:0.3f}'. format(accuracy_score(y_test,
CV_rfc.predict(X_test))))
```

```
Model accuracy score : 0.999
```

Hence, we can see that using our transfomed custom feature is better for the accuracy

Model 2: Gradient Boosting model

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report,
roc_auc_score,accuracy_score
from sklearn.model_selection import learning_curve

gbm_model = GradientBoostingClassifier().fit(X_train,y_train)

y_pred = gbm_model.predict(X_test)
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))
```

```
Model accuracy score: 0.9986
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    161901
           1       1.00      0.98      0.99      9293

    accuracy                           1.00    171194
   macro avg       1.00      0.99      0.99    171194
weighted avg       1.00      1.00      1.00    171194
```

the accuracy score is similar between the train and test data, hence no overfitting

```
print('Training set score: {:.4f}'.format(gbm_model.score(X_train,
y_train)))

print('Test set score: {:.4f}'.format(gbm_model.score(X_test,
y_test)))
```

```
Training set score: 0.9987
Test set score: 0.9986
```

```
gbm_model.get_params()
```

```
{'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'deviance',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
```

```
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': None,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

Fine-tune the model with gridsearchcv

```
from sklearn.model_selection import GridSearchCV

pipeline = Pipeline([('user_transform', UserDefinedTransform()),
('GBM', GradientBoostingClassifier())])

params_gbm = {
    "GBM__max_depth": [3, 4],
    'user_transform__installment_cut': [True, False]
}

CV_gbm = GridSearchCV(pipeline, param_grid=params_gbm,
scoring='accuracy')

CV_gbm.fit(X_train, y_train)
accuracy_score(y_test, CV_gbm.predict(X_test))

0.999012815869715

CV_gbm.best_params_

{'GBM__max_depth': 4, 'user_transform__installment_cut': True}
```

We can see that using the new feature is better for model accuracy

Model 3: Logistic Regression

```
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression


# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)


# fit the model
logreg.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0, solver='liblinear')

y_pred = logreg.predict(X_test)
print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
y_pred)))

Model accuracy score: 0.9952
```

Since the accuracy is almost the same for both data, there is no overfitting

```
print('Training set score: {:.4f}'.format(logreg.score(X_train,
y_train)))

print('Test set score: {:.4f}'.format(logreg.score(X_test, y_test)))

Training set score: 0.9950
Test set score: 0.9952

print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    161901
           1       0.99      0.92      0.95      9293

    accuracy                           1.00    171194
   macro avg       0.99      0.96      0.98    171194
weighted avg       1.00      1.00      1.00    171194
```

Fine-tune the model with gridsearchcv

```
pipeline = Pipeline([('user_transform', UserDefinedTransform()),
('LR', LogisticRegression())])

parameters = {'user_transform__installment_cut': [True, False],
                'LR__penalty':['l1','l2'],
              'LR__C':[1, 10, 100]}

grid_search = GridSearchCV(pipeline, param_grid=parameters,
scoring='accuracy')

grid_search.fit(X_train, y_train)

GridSearchCV(estimator=Pipeline(steps=[('user_transform',
                                        UserDefinedTransform()),
                                       ('LR', LogisticRegression())]),
             param_grid={'LR__C': [1, 10, 100], 'LR__penalty': ['l1',
'l2'],
                         'user_transform__installment_cut': [True,
```

```
False]},
                 scoring='accuracy')

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,
grid_search.predict(X_test))))

Model accuracy score: 0.9952

grid_search.best_params_

{'LR__C': 10, 'LR__penalty': 'l2', 'user_transform__installment_cut':
True}
```

Again, it is better to use the user-defined feature for model accuracy

Final Evaluation

Both Random Forest classifier and GBM model show an accuracy over 0.99, and there exists no overfitting and precision and recall scores are stable as well. We can say that both models show outstanding performances, but considering that this dataset is quite imbalanced and the time taken to complete is much shorter for the random forest classifier model, I would say that the random forest classifier is the best option out of the three.