project2

April 7, 2024

1 Machine Learning in Python - Project 2

Due Friday, April 12th by 4 pm.

Include contributors names in notebook metadata or here

1.1 Setup

Install any packages here and load data

[182]:
```
# Add any additional libraries or submodules below

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80

# sklearn modules
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import FunctionTransformer, StandardScaler,␣
↪OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score, roc_auc_score, balanced_accuracy_score, log_loss

from sklearn.dummy import DummyClassifier
```

[2]:
```python
# Load data in easyshare.csv
d = pd.read_csv("freddiemac.csv")
d.head()
```

[2]:
```
    fico  dt_first_pi flag_fthb  dt_matr   cd_msa  mi_pct  cnt_units  \
0  709.0       201703         9   204702      NaN      12          1
1  649.0       201703         9   203202  33124.0       0          1
2  747.0       201703         9   203702  41180.0       0          1
3  711.0       201703         9   204702  20260.0       0          1
4  751.0       201703         N   204702      NaN      35          1

  occpy_sts  cltv  dti  …  zipcode      id_loan loan_purpose  \
0         P    84   26  …    51300  F117Q1000376            N
1         P    52   22  …    33100  F117Q1000418            C
2         I    43   20  …    63100  F117Q1000479            N
3         I    80   21  …    55800  F117Q1000523            P
4         P    95   24  …    75900  F117Q1000719            P

  orig_loan_term cnt_borr    seller_name       servicer_name flag_sc  \
0            360        2  Other sellers     Other servicers     NaN
1            180        2  Other sellers     Other servicers     NaN
2            240        2  Other sellers     Other servicers     NaN
3            360        2  Other sellers     Other servicers     NaN
4            360        1  Other sellers  ARVESTCENTRALMTGECO     NaN

   prepaid default
0        0       1
1        1       0
2        1       0
3        1       0
4        1       0

[5 rows x 28 columns]
```

2 Introduction

This section should include a brief introduction to the task and the data (assume this is a report you are delivering to a professional body (e.g. FreddiMac company or similar company). If you use any additional data sources, you should introduce them here and discuss why they were included.

Briefly outline the approaches being used and the conclusions that you are able to draw.

3 Exploratory Data Analysis and Feature Engineering

Include a detailed discussion of the data with a particular emphasis on the features of the data that are relevant for the subsequent modeling. Including visualizations of the data is strongly encouraged - all code and plots must also be described in the write up. Think carefully about whether each plot *needs to be included in your final draft - your report should include figures but they should be as focused and impactful as possible.*

You should also split your data into training and testing sets, ideally before you look to much into the features and relationships with the target

Additionally, this section should also implement and describe any preprocessing / feature engineering of the data. Specifically, this should be any code that you use to generate new columns in the data frame `d`. Feature engineering that will be performed as part of an sklearn pipeline can be mentioned here but should be implemented in the following section.

If you decide to extract additional features from the full data (easyshare_all.csv), describe these variables here.

All code and figures should be accompanied by text that provides an overview / context to what is being done or presented.

variable summary

Numerical variable `fico` (credit score);

Categorical variable `dt_first_pi` (date of the first mortgage payment), it's a 6-digit number with format YYYYMM. From year 2017 to 2019.

Categorical variable `dt_matr` (maturity date, date of the last mortgage payment), it's a 6-digit number with format YYYYMM. From 202504 to 204812.

Binary variable `flag_fthb` (first time homebuyer), with missing value encoded with 9.

Numerical variable `orig_upb` (loan amount that has not yet been paid off);

Numerical variable `int_rt` (interest rate of the loan);

Identifier `cd_msa`, they are 5-digit codes of Metropolitan Statistical Area (MSA) regions in the US, where the complete list of encodings can be found in this document.

Categorical variable `mi_pct` (percentage of the loan amount that's required for mortgage insurance. It is often required when the borrower's down payment on a home is less than a certain percentage of the home's purchase price.) It's classified as categorical because only there's only 7 insurance levels: 0,6,12,20,25,30,35.

Categorical variable `cnt_units` (number of units in the morgaged property), 4 levels: 1,2,3,4.

Categorical variable `occpy_sts` (mortgage type), 3 levels: owner occupied (P), second home (S), or investment property (I).

Numerical variable `cltv` (rate of loan amount to total property value, e.g. 90%) SAME AS `ltv`;

Numerical variable `dti` (debt-to-income ratio, which is calculated by monthly housing expenses that incorporate the mortgage payment, divided by the monthly income used to underwrite the loan);

Numerical variable `ltv` (loan-to-value). For example, if a borrower takes out a mortgage for £150,000 to purchase a home that is appraised at $£200,000$, the original loan-to-value ratio would be $\frac{150{,}000}{200{,}000} = 0.75$, or 75%. This means that the borrower is financing 75% of the property's value with the mortgage loan, and the remaining 25% is covered by the borrower's down payment or equity.

Numerical variable `int_rt` (interest rate of the property);

Categorical variable `channel`;

Binary variable `ppmt_pnlty`, with Yes or No (penalty applied). A prepayment penalty is a fee charged by lenders if the borrower pays off the mortgage loan before the agreed-upon term. Note there's no Y instance in this dataset.

Binary variable `prod_type` only fixed-rate mortgage in this dataset.

Categorical variable `st` (US states) two-letter abbreviations;

Categorical variable `prop_type`, property type: condominium (CO), planned unit development (PU), cooperative share (CP), manufactured home (MH), or Single-Family home (SF).

Identifier `zipcode`, they are 5-digit codes in the form of ###00;

Identifier `id_loan`, unique ID for each entry;

Categorical variable `loan_purpose`, Cash-out Refinance mortgage (C), No Cash-out Refinance mortgage (N), Refinance mortgage not specified (R), or a Purchase mortgage (P);

Numerical variable `orig_loan_term`, number of monthly payments from first payment until maturity date.

Binary variable `cnt_borr`, the number of borrower(s) who're obligated to pay the mortgage. 1 = one borrower, 2 = more than one borrower.

Categorical variable `seller_name`, list of names of seller of mortgages.

Categorical variable `servicer_name`, list of names of servicer of mortgages.

Binary variable `flag_sc`, all entries either have Y or NaN.

Binary variable `default`, our response variable, 1=default, 0=no default.

Missing value analysis

There is 1 missing value for `fico` (credit score);

3468 NA values for `flag_fthb` (binary, first time homebuyer);

594 null values for `cd_msa` (metropolitan statistical area), indicating 594 mortgaged properties are either not in a Metropolitan Area or MSA status unknown;

1 NA for `cltv`;

1 NA for `dti`, indicating 1 impossible value of $> 65\%$;

1 NA for `ltv`;

38 missing values for `ppmt_pnlty`,

5751 missing values for `flag_sc`.

[3]:
```
missing_values_count = d.isnull().sum()
missing_values_table = pd.DataFrame({'Missing Values': missing_values_count})

print("Table of Null Values in Each Variable:")
```

```python
print(missing_values_table)
count_9999 = d['fico'].astype(str).str.count('9999').sum()
print("Number of NA (encoded as 9999) in 'fico':", count_9999)
count_9 = d['flag_fthb'].astype(str).str.count('9').sum()
print("Number of NA (encoded as 9) in 'flag_fthb':", count_9)
count_999 = d['mi_pct'].astype(str).str.count('999').sum()
print("Number of NA (999) in 'mi_pct':", count_999)
count_99 = d['cnt_units'].astype(str).str.count('99').sum()
print("Number of no information (99) in 'cnt_units':", count_99)
c9 = d['occpy_sts'].astype(str).str.count('9').sum()
print("Number of no information (9) in 'occpy_sts':", c9)
c999 = d['cltv'].astype(str).str.count('999').sum()
print("Number of no information (999) in 'cltv':", c999)
c_999 = d['dti'].astype(str).str.count('999').sum()
print("Number of NA (999) in 'dti':", c_999)
co_999 = d['ltv'].astype(str).str.count('999').sum()
print("Number of NA (999) in 'ltv':", co_999)
co_9 = d['channel'].astype(str).str.count('9').sum()
print("Number of NA (9) in 'channel':", co_9)
co_99 = d['prop_type'].astype(str).str.count('99').sum()
print("Number of NA (99) in 'prop_type':", co_99)
c_00 = d['zipcode'].astype(str).str.count('###00').sum()
print("Number of NA in 'zipcode':", c_00)
cou_9 = d['loan_purpose'].astype(str).str.count('9').sum()
print("Number of NA in 'loan_purpose':", cou_9)
```

```
Table of Null Values in Each Variable:
                Missing Values
fico                         1
dt_first_pi                  0
flag_fthb                    0
dt_matr                      0
cd_msa                     594
mi_pct                       0
cnt_units                    0
occpy_sts                    0
cltv                         0
dti                          0
orig_upb                     0
ltv                          0
int_rt                       0
channel                      0
ppmt_pnlty                  38
prod_type                    0
st                           0
prop_type                    0
zipcode                      0
```

```
id_loan                          0
loan_purpose                     0
orig_loan_term                   0
cnt_borr                         0
seller_name                      0
servicer_name                    0
flag_sc                       5751
prepaid                          0
default                          0
Number of NA (encoded as 9999) in 'fico': 0
Number of NA (encoded as 9) in 'flag_fthb': 3468
Number of NA (999) in 'mi_pct': 0
Number of no information (99) in 'cnt_units': 0
Number of no information (9) in 'occpy_sts': 0
Number of no information (999) in 'cltv': 1
Number of NA (999) in 'dti': 1
Number of NA (999) in 'ltv': 1
Number of NA (9) in 'channel': 0
Number of NA (99) in 'prop_type': 0
Number of NA in 'zipcode': 0
Number of NA in 'loan_purpose': 0
```

Training and testing data split: 90% and 10% of the data are allocated to training and testing dataset, respectively.

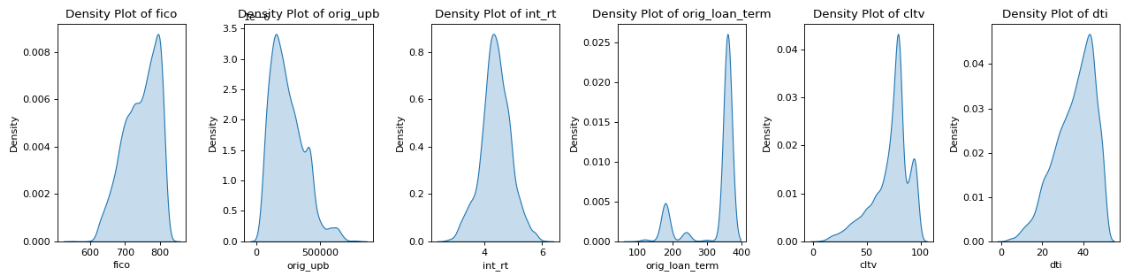
[191]:
```
X = d.drop(columns=['default'])
y = d['default']  # Response variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪stratify=y, random_state=42)
```

Numerical variables: the density plots, boxplots, heatmap and scatterplots of all continuous numerical variables.

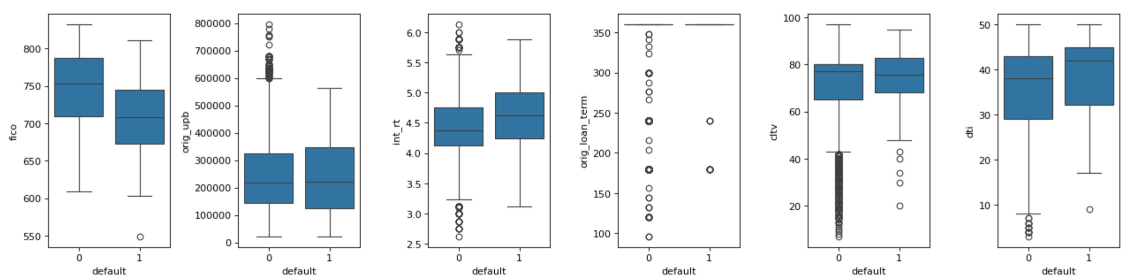
[5]:
```
# filter NA coded as 999

filtered_idx = X_train[(X_train['cltv'] != 999) & (X_train['dti'] != 999)].index
X_train_clean = X_train.loc[filtered_idx]
y_train_clean = y_train.loc[filtered_idx]
```

[6]:
```
# filter NA coded as 999
#filtered_cltv = X_train[X_train['cltv'] != 999]['cltv']
#filtered_dti = X_train[X_train['dti'] != 999]['dti']
fig, axes = plt.subplots(nrows=1, ncols=6, figsize=(16, 4))

# Numerical variables
num_var = ['fico', 'orig_upb', 'int_rt','orig_loan_term','cltv','dti']
for i, variable in enumerate(num_var):
```

```
    sns.kdeplot(data=X_train_clean[variable], ax=axes[i], fill=True)

    axes[i].set_title(f'Density Plot of {variable}')

plt.tight_layout()
plt.show()
```



[7]:
```
fig, axes = plt.subplots(nrows=1, ncols=6, figsize=(16, 4))

# Looping through the numerical variables and creating the boxplots
for i, var in enumerate(num_var):
    sns.boxplot(x=y_train_clean, y=X_train_clean[var], ax=axes[i])

# Displaying the plot
plt.tight_layout()
plt.show()
```

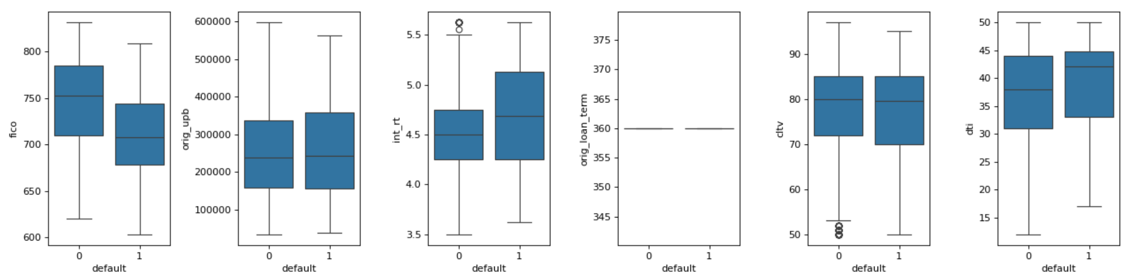


most of observations in 'orig_loan_term' are = 360. Very few observations (and defaults) for others consider deleting them.

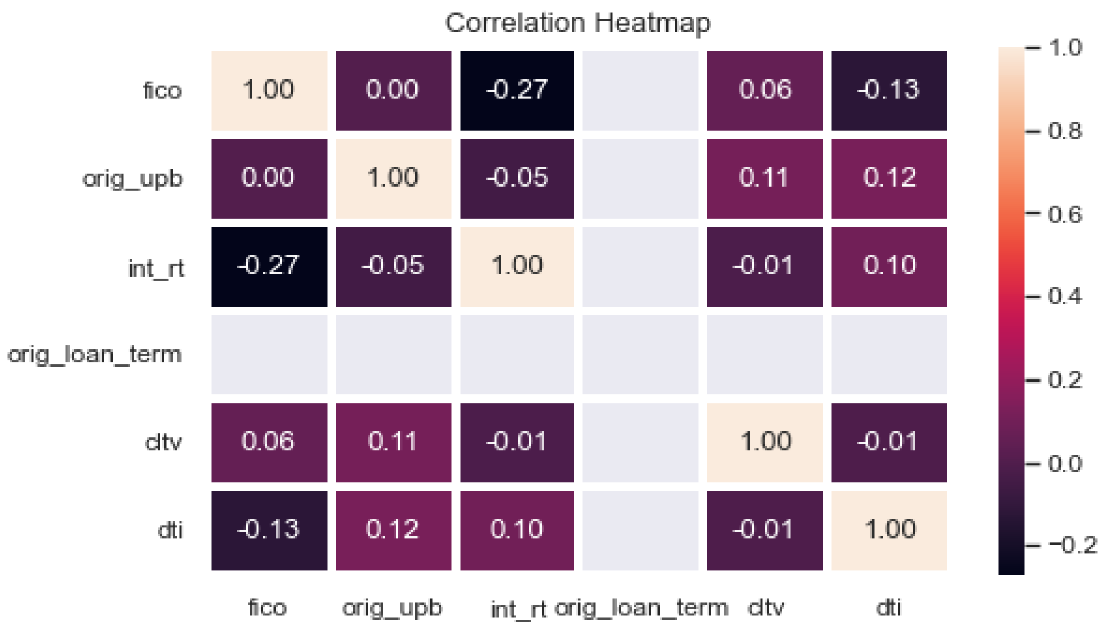
[8]:
```
# set na
for column in X_train_clean[num_var].columns:
    Q1 = X_train_clean[num_var][column].quantile(0.25)
    Q3 = X_train_clean[num_var][column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
```

```
    upper_bound = Q3 + 1.5 * IQR

    filtered_idx = X_train_clean[(X_train_clean[column] >= lower_bound) &
↪(X_train_clean[column] <= upper_bound)].index
    X_train_clean = X_train_clean.loc[filtered_idx]
    y_train_clean = y_train_clean.loc[filtered_idx]

    #d[column] = d[num_var][column].mask((d[num_var][column] < lower_bound) |
↪(d[num_var][column] > upper_bound), np.nan)
```

[9]:
```
# boxplot after filtering iqr
fig, axes = plt.subplots(nrows=1, ncols=6, figsize=(16, 4))

for i, var in enumerate(num_var):
    sns.boxplot(x=y_train_clean, y=X_train_clean[var], ax=axes[i])

plt.tight_layout()
plt.show()
```



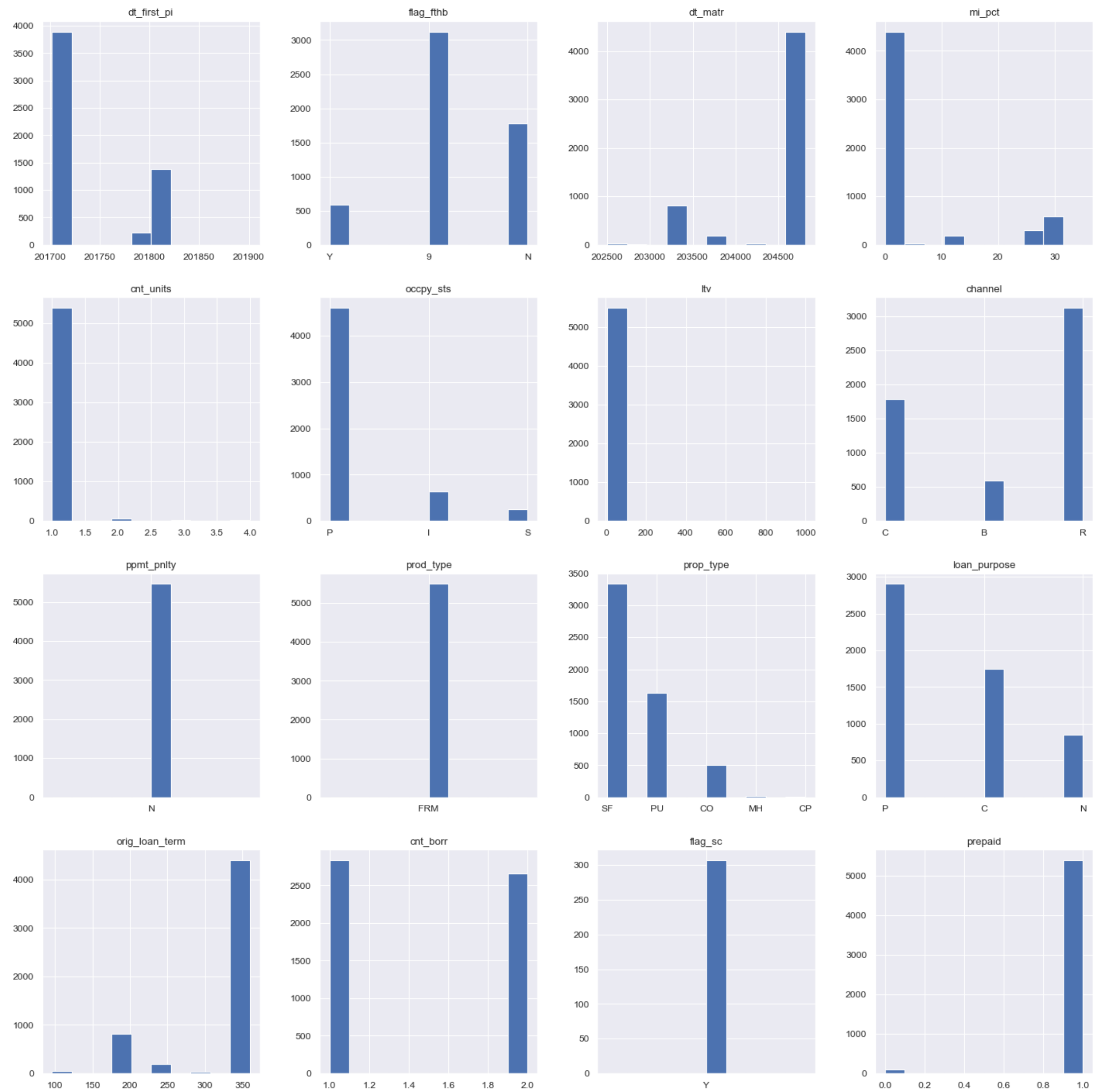
[10]:
```
sns.set(rc={'figure.figsize': (7, 4)})
sns.heatmap(X_train_clean[num_var].corr(), annot = True, fmt = '.2f',
↪linewidths = 2)
plt.title("Correlation Heatmap")
plt.show()
```



loan term is constant - all 360

Categorical variables: Below displays the bar charts of all categorical variables.

[11]:
```
# Identifiers variables are: 'id_loan','cd_msa','zipcode'
# Long catgotical variables are: 'st', 'servicer_name', 'seller_name'
# Numerical variables are: 'fico', 'orig_upb', 'int_rt','cltv','dti'
exclude_var = ['id_loan','cd_msa','zipcode','st', 'servicer_name',␣
↪'seller_name','fico', 'orig_upb', 'int_rt','cltv','dti']

columns_to_plot = [col for col in X_train_clean.columns if col not in␣
↪exclude_var]

fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(18, 18))
axes = axes.flatten()
fig.patch.set_facecolor('white')

for i, column in enumerate(columns_to_plot):
    X_train[column].hist(ax=axes[i])
    axes[i].set_title(f'{column}')
    axes[i].set_xlabel(' ')
    axes[i].set_ylabel(' ')

plt.tight_layout()
plt.show()
```

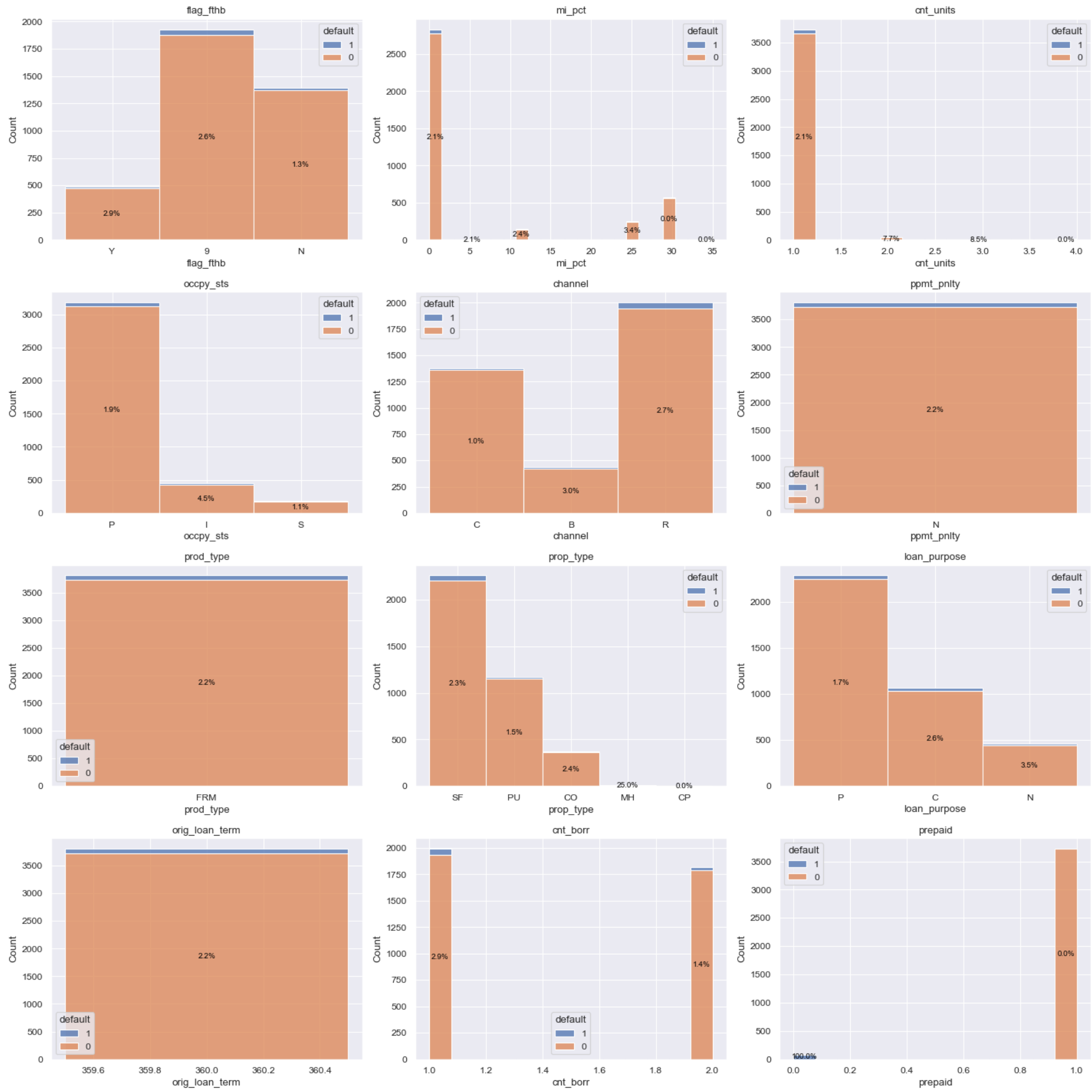


[12]:
```
exclude_var = ['id_loan','cd_msa','zipcode','st', 'servicer_name', 'seller_name','fico', 'orig_upb', 'int_rt','cltv', 'ltv', 'dti']

columns_to_plot = [col for col in X_train_clean.columns if col not in exclude_var + ['flag_sc', 'ppmt_pnlt', 'dt_matr', 'dt_first_pi']] # data first payment is sht, why do you care
```

[13]:
```
data = pd.concat([X_train_clean[columns_to_plot].astype('object'), pd.DataFrame(y_train_clean)], axis=1).dropna()
```

[14]:
```
fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(18, 18))
axes = axes.flatten()
```

```
for i, column in enumerate(columns_to_plot):
    ax = sns.histplot(data=data, x=column, hue='default', multiple='stack',␣
↪hue_order=[1, 0], ax=axes[i])
    axes[i].set_title(f'{column}')

    # Calculate the counts for each category within the column
    category_order = data[column].dropna().unique()
    counts_total = data[column].value_counts().reindex(category_order).fillna(0)
    counts_default_1 = data[data['default'] == 1][column].value_counts().
↪reindex(category_order).fillna(0)

    # Calculate the percentages
    percentages = 100 * counts_default_1 / counts_total

    # Iterate over the bars for the current axis
    bar_patches = [p for p in ax.patches if p.get_height() > 0]  # Only␣
↪consider bars with height > 0
    for j, bar in enumerate(bar_patches[:len(category_order)]):
        # The percentage for the category is at the same position as the bar
        percentage = percentages.iloc[j]
        # Annotate the percentage in the middle of the bar
        ax.text(bar.get_x() + bar.get_width() / 2, bar.get_height() / 2,␣
↪f'{percentage:.1f}%',
                ha='center', va='center', fontsize=9, color='black')

plt.tight_layout()
plt.show()
```



```
[15]: data = pd.concat([X_train_clean['zipcode'].astype('object'), pd.
      ↪DataFrame(y_train_clean)], axis=1).dropna()

      zipcode_counts = data['zipcode'].value_counts()

      top_20_zipcodes = zipcode_counts.head(20).index

      top_zipcodes_data = data[data['zipcode'].isin(top_20_zipcodes)]

      default_percentages = top_zipcodes_data.groupby('zipcode')['default'].agg(
          total_defaults='sum',
          total='count',
      )
```

```
default_percentages['default_percentage'] =␣
↪default_percentages['total_defaults'] / default_percentages['total']

top_zipcodes_default_percentage = default_percentages.
↪sort_values('default_percentage', ascending=False)
print(top_zipcodes_default_percentage)
```

```
         total_defaults  total  default_percentage
zipcode
95300                 2     31            0.064516
75000                 2     39            0.051282
60000                 1     28            0.035714
92800                 1     29            0.034483
92500                 1     31            0.032258
80000                 1     31            0.032258
89100                 1     33            0.030303
92000                 1     33            0.030303
30000                 1     37            0.027027
95600                 1     42            0.023810
94500                 1     55            0.018182
80100                 0     44            0.000000
91700                 0     37            0.000000
85300                 0     45            0.000000
84000                 0     48            0.000000
85200                 0     57            0.000000
80200                 0     33            0.000000
92600                 0     28            0.000000
98000                 0     40            0.000000
98200                 0     27            0.000000
```

zipcode doesnt seem very relevant to default rate

[16]:
```
def default_percentages(var_name):
    var_data = X_train_clean[var_name].astype('object').fillna('Missing')

    data = pd.concat([var_data, pd.DataFrame(y_train_clean)], axis=1)

    default_percentages = data.groupby(var_name)['default'].agg(
    total_defaults='sum',
    total='count',
    )

    default_percentages['default_percentage'] =␣
↪default_percentages['total_defaults'] / default_percentages['total']
    print(default_percentages.sort_values('default_percentage',␣
↪ascending=False))
```

[17]:
```
default_percentages('flag_fthb')
```

```
           total_defaults  total  default_percentage
flag_fthb
Y                      14    490            0.028571
9                      50   1939            0.025786
N                      18   1406            0.012802
```

i don tknow maybe treat 9 as a separate category. The default rate is quite high

[18]:
```
default_percentages('flag_sc')
```

```
         total_defaults  total  default_percentage
flag_sc
Missing              79   3640            0.021703
Y                     3    195            0.015385
```

identifiers `id_loan` is an unique identifiers with no duplicates in this dataset. On the contrary, `zipcode` is not an unique identifier, observations are 5-digit area codes. Similarly, `cd_msa` are 5-digit codes of Metropolitan Statistical Area (MSA) regions, where the complete list of regions can be found here.

[19]:
```
# Check for duplicates in 'id_loan' variable
duplicates = d[d.duplicated(subset=['id_loan'], keep=False)]

if not duplicates.empty:
    print("Duplicates found in 'id_loan' variable:")
    print(duplicates)
else:
    print("No duplicates found in 'id_loan' variable.")

d['zipcode_str'] = d['zipcode'].astype(str)
```

```
No duplicates found in 'id_loan' variable.
```

[20]:
```
X_train_clean['flag_fthb']
```

[20]:
```
1886    Y
2440    9
2545    9
3112    9
3043    9
       ..
161     9
4909    9
381     N
1422    N
5727    9
Name: flag_fthb, Length: 3835, dtype: object
```

Feature Engineering

`flag_fthb` replace all 9s with NaN and map Y as 1, N as 0.

`cnt_units` One-hot encoding 4 levels: 1,2,3,4.

`occpy_sts` One-hot encoding 3 levels: P,S,I.

`cltv`, `dti`,`ltv` discard 1 NA.

`channel` One-hot encoding 4 levels: R,B,C,T.

`ppmy_pnlty` disccard all NaN, map Y as 1, N as 0. Note there's no Y in this dataset.

`prod_type` discard this feature, all observations are "FRM" (fixed-rate mortgage). Have no predictive power to adjustable-rate mortgage.

`prop_type` One-hot encoding to 5 levels: 'SF' 'PU' 'MH' 'CO' 'CP'.

`loan_purpose` One-hot encoding to 4 levels: C,N,R,P.

`cnt_bnrr` Map 1(1 borrower) to 0 and 2(>1 borrower) to 1.

`flag_sc` discard this feature, all observations are either Y or NaN. Have no predictive power.

[ ]:
```
# Replace '9' values with NaN
X_train['flag_fthb'] = X_train['flag_fthb'].replace('9', np.nan)
X_test['flag_fthb'] = X_test['flag_fthb'].replace('9', np.nan)
# Map 'Y' to 1 and 'N' to 0
X_train['flag_fthb'] = X_train['flag_fthb'].map({'Y': 1, 'N': 0})
X_test['flag_fthb'] = X_test['flag_fthb'].map({'Y': 1, 'N': 0})
```

[ ]:
```
# There's no NA in 'cnt_units', apply one-hot encoding to values 1, 2, 3, 4
X_train = pd.get_dummies(X_train, columns=['cnt_units'], prefix='cnt_units')
X_test = pd.get_dummies(X_test, columns=['cnt_units'], prefix='cnt_units')
```

[ ]:
```
# There's no NA in 'occpy_sts', apply one-hot encoding to P,S,I
X_train = pd.get_dummies(X_train, columns=['occpy_sts'], prefix='occpy_sts')
X_test = pd.get_dummies(X_test, columns=['occpy_sts'], prefix='occpy_sts')
```

[ ]:
```
# Filter out instances of 999 from 'cltv','dti','ltv'
X_train = X_train[X_train['cltv'] != 999]
X_test = X_test[X_test['cltv'] != 999]
X_train = X_train[X_train['dti'] != 999]
X_test = X_test[X_test['dti'] != 999]
X_train = X_train[X_train['ltv'] != 999]
X_test = X_test[X_test['ltv'] != 999]
# Discard missing values in the 'fico'
X_train = X_train.dropna(subset=['fico'])
X_test = X_test.dropna(subset=['fico'])
```

[ ]:
```
# There's no NA in 'channel', apply one-hot encoding to R,B,C,T.
X_train = pd.get_dummies(X_train, columns=['channel'], prefix='channel')
```

```
X_test = pd.get_dummies(X_test, columns=['channel'], prefix='channel')
```

[ ]:
```
# Discard all NaN observations from the 'ppmy_pnlty' column
X_train = X_train.dropna(subset=['ppmt_pnlty'])
X_test = X_test.dropna(subset=['ppmt_pnlty'])
# Encode 'N' as 0 and 'Y' as 1, note there's no Y in d dataframe
X_train['ppmt_pnlty'] = X_train['ppmt_pnlty'].map({'N': 0, 'Y': 1})
X_test['ppmt_pnlty'] = X_test['ppmt_pnlty'].map({'N': 0, 'Y': 1})
```

[ ]:
```
# Discard the 'prod_type'
X_train = X_train.drop(columns=['prod_type'])
X_test = X_test.drop(columns=['prod_type'])
```

[ ]:
```
# There's no NA in 'prop_type', apply one-hot encoding to 'SF' 'PU' 'MH' 'CO'␣
↪'CP'
X_train = pd.get_dummies(X_train, columns=['prop_type'], prefix='prop_type')
X_test = pd.get_dummies(X_test, columns=['prop_type'], prefix='prop_type')
```

[ ]:
```
# There's no NA in 'loan_purpose', apply one-hot encoding to 4 levels: C,N,R,P.
X_train = pd.get_dummies(X_train, columns=['loan_purpose'],␣
↪prefix='loan_purpose')
X_test = pd.get_dummies(X_test, columns=['loan_purpose'], prefix='loan_purpose')
```

[ ]:
```
# Map 1 to 0 and 2 to 1
X_train['cnt_borr'] = X_train['cnt_borr'].map({1: 0, 2: 1})
X_test['cnt_borr'] = X_test['cnt_borr'].map({1: 0, 2: 1})
```

[ ]:
```
# Discard the 'flag_sc'
X_train = X_train.drop(columns=['flag_sc'])
X_test = X_test.drop(columns=['flag_sc'])
```

[ ]:
```
unique_occurrences = d['flag_sc'].unique()
print(unique_occurrences)
```

[ ]:
```
smallest_value = d['orig_loan_term'].min()
largest_value = d['orig_loan_term'].max()

print("Smallest value in column 'dt_matr':", smallest_value)
print("Largest value in column 'dt_matr':", largest_value)
```

[ ]:
```
def clean_data(X, y):
    X_clean = X.copy()
    y_clean = y.copy()

    X_clean['cltv'] = X_clean['cltv'].replace(999, np.nan)
    X_clean['dti'] = X_clean['dti'].replace(999, np.nan)
```

```
    return X_clean, y_clean
```

4 Model Fitting and Tuning

In this section you should detail your choice of model and describe the process used to refine and fit that model. You are strongly encouraged to explore many different modeling methods (e.g. linear regression, interaction terms, lasso, etc.) but you should not include a detailed narrative of all of these attempts. At most this section should mention the methods explored and why they were rejected - most of your effort should go into describing the model you are using and your process for tuning and validating it.

For example if you considered a linear regression model, a polynomial regression, and a lasso model and ultimately settled on the linear regression approach then you should mention that other two approaches were tried but do not include any of the code or any in depth discussion of these models beyond why they were rejected. This section should then detail is the development of the linear regression model in terms of features used, interactions considered, and any additional tuning and validation which ultimately led to your final model.

This section should also include the full implementation of your final model, including all necessary validation. As with figures, any included code must also be addressed in the text of the document.

Finally, you should also provide comparison of your model with baseline model(s) on the test data but only briefly describe the baseline model(s) considered

[197]:
```
num_features = ['fico', 'orig_upb', 'int_rt','orig_loan_term','cltv','dti']
#cat_features = X_train_clean.columns.drop(num_features)
cat_features = ['flag_fthb', 'flag_sc',␣
↪'prepaid','cnt_borr','loan_purpose','prop_type','ppmt_pnlty','prod_type']
```

[178]:
```
class CleanDataTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, value_to_replace):
        self.value_to_replace = value_to_replace

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

    def transform(self, X):
        return X.replace(self.value_to_replace, np.nan)

class IQRBasedOutlierRemoverEnhanced(BaseEstimator, TransformerMixin):
    def __init__(self, factor=1.5, remove_outliers=False):
        self.factor = factor
        self.remove_outliers = remove_outliers

    def fit(self, X, y=None):
        # Compute the IQR bounds
        Q1 = np.percentile(X, 25, axis=0)
```

```
        Q3 = np.percentile(X, 75, axis=0)
        IQR = Q3 - Q1
        self.lower_bounds_ = Q1 - self.factor * IQR
        self.upper_bounds_ = Q3 + self.factor * IQR
        return self

    def transform(self, X):
        if self.remove_outliers:
            # Apply the mask for the bounds to the data
            mask = (X >= self.lower_bounds_) & (X <= self.upper_bounds_)
            return X[mask]
        else:
            # Mark outliers as NaN
            mask_lower = (X < self.lower_bounds_)
            mask_upper = (X > self.upper_bounds_)
            X_copy = X.copy()
            X_copy[mask_lower | mask_upper] = np.nan
            return X_copy


class AutoBinaryEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, val1='N', val2='Y'):
        self.val1 = val1
        self.val2 = val2

    def fit(self, X, y=None):
        # Dictionary to store mappings for each column
        self.mappings_ = {}
        for col in X.columns:
            self.mappings_[col] = {self.val1: "0", self.val2: "1"}
        return self

    def transform(self, X):
        X_copy = X.copy()
        for col, mapping in self.mappings_.items():
            X_copy[col] = X_copy[col].map(mapping)
        return X_copy
```

[180]:
```
preprocessed_data = cat_pre1.fit_transform(X_train[cat])
preprocessed_data
```

[180]:
```
array([['1', 'missing', 'missing', …, 'missing', '0', 'missing'],
       ['missing', 'missing', 'missing', …, 'missing', '0', 'missing'],
       ['0', 'missing', 'missing', …, 'missing', '0', 'missing'],
       …,
       ['0', 'missing', 'missing', …, 'missing', '0', 'missing'],
       ['missing', 'missing', 'missing', …, 'missing', '0', 'missing'],
```

```
       ['missing', 'missing', 'missing', …, 'missing', '0', 'missing']],
      dtype=object)
```

[164]:
```
X_train[cat].astype('object')
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5493 entries, 1886 to 2790
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   flag_fthb     5493 non-null   object
 1   flag_sc       307 non-null    object
 2   prepaid       5493 non-null   object
 3   cnt_borr      5493 non-null   object
 4   loan_purpose  5493 non-null   object
 5   prop_type     5493 non-null   object
 6   ppmt_pnlty    5459 non-null   object
 7   prod_type     5493 non-null   object
dtypes: object(8)
memory usage: 515.3+ KB
```

[208]:
```
cat_pre1 = Pipeline(steps=[
    ('cat_clean', CleanDataTransformer(value_to_replace='9')),
    ('cat_binary_encode', AutoBinaryEncoder()),
    ("cat_impute", SimpleImputer(strategy="constant", fill_value="missing")),
    ("cat_encode", OneHotEncoder(drop='first'))])

cat_pre2 = Pipeline(steps=[
    ('cat_binary_encode', AutoBinaryEncoder(val1='1', val2='2')),
    ("cat_impute", SimpleImputer(strategy="constant", fill_value="missing")),
    ("cat_encode", OneHotEncoder(drop='first'))])

cat_pre3 = Pipeline(steps=[
    ("cat_impute", SimpleImputer(strategy="constant", fill_value="missing")),
    ("cat_encode", OneHotEncoder(drop='first'))])

cat_preprocessing = ColumnTransformer([
    ("cat_pre1", cat_pre1, ['flag_fthb']),
    ("cat_pre2", cat_pre2, ['cnt_borr']),
    ("cat_pre3", cat_pre3, [x for x in cat if x not in {"flag_fthb",␣
 ↪"cnt_borr"}])],
     remainder='passthrough')


pipe = Pipeline([("pre_processing", cat_preprocessing),
                 ("model", LogisticRegression())])
```

```
pipe.fit(X_train[cat].astype('object'), y_train)
```

[208]:
```
Pipeline(steps=[('pre_processing',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_pre1',
                                                  Pipeline(steps=[('cat_clean',
CleanDataTransformer(value_to_replace='9')),
('cat_binary_encode',
AutoBinaryEncoder()),
                                                                  ('cat_impute',
SimpleImputer(fill_value='missing',
 strategy='constant')),
                                                                  ('cat_encode',
OneHotEncoder(drop='first'))]),
                                                  ['flag_fthb']),
                                                 ('…
SimpleImputer(fill_value='missing',
 strategy='constant')),
                                                                  ('cat_encode',
OneHotEncoder(drop='first'))]),
                                                  ['cnt_borr']),
                                                 ('cat_pre3',
                                                  Pipeline(steps=[('cat_impute',
SimpleImputer(fill_value='missing',
 strategy='constant')),
                                                                  ('cat_encode',
OneHotEncoder(drop='first'))]),
                                                  ['flag_sc', 'prepaid',
                                                   'loan_purpose', 'prop_type',
                                                   'ppmt_pnlty',
                                                   'prod_type'])])),
                ('model', LogisticRegression())])
```

[193]:
```
accuracy = accuracy_score(y_test, pipe.predict(X_test[cat].astype('object')))

print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 1.00
```

[201]:
```
preprocessed_data = num_pre1.fit_transform(X_train[cat])
preprocessed_data
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [201], line 1
----> 1 preprocessed_data = num_pre1.fit_transform(X_train[cat])
      2 preprocessed_data
```

```
File /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/
↪site-packages/sklearn/pipeline.py:413, in Pipeline.fit_transform(self, X, y,␣
↪**fit_params)
    386 def fit_transform(self, X, y=None, **fit_params):
    387     """Fit the model and transform with the final estimator.
    388
    389     Fits all the transformers one after the other and transform the
   (…)
    411         Transformed samples.
    412     """
--> 413     fit_params_steps = self._check_fit_params(**fit_params)
    414     Xt = self._fit(X, y, **fit_params_steps)
    416     last_step = self._final_estimator

File /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/
↪site-packages/sklearn/pipeline.py:297, in Pipeline._check_fit_params(self,␣
↪**fit_params)
    296 def _check_fit_params(self, **fit_params):
--> 297     fit_params_steps = {name: {} for name, step in self.steps if step is␣
↪not None}
    298     for pname, pval in fit_params.items():
    299         if "__" not in pname:

File /Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/
↪site-packages/sklearn/pipeline.py:297, in <dictcomp>(.0)
    296 def _check_fit_params(self, **fit_params):
--> 297     fit_params_steps = {name: {} for name, step in self.steps if step is␣
↪not None}
    298     for pname, pval in fit_params.items():
    299         if "__" not in pname:

ValueError: too many values to unpack (expected 2)
```

[210]:
```
num_pre1 = Pipeline(steps=[
    ('num_clean', CleanDataTransformer(value_to_replace='999')),
    ("num_outliers", IQRBasedOutlierRemoverEnhanced(remove_outliers=False)),
    ("num_impute", SimpleImputer(strategy="median")),
    ("num_scale", StandardScaler())])

num_pre2 = Pipeline(steps=[
    ("num_outliers", IQRBasedOutlierRemoverEnhanced(remove_outliers=False)),
    ("num_impute", SimpleImputer(strategy="median")),
    ("num_scale", StandardScaler())])

num_preprocessing = ColumnTransformer([
    ("num_pre1", num_pre1, ['cltv', 'dti']),
```

```
    ("num_pre2", num_pre2, [x for x in num_features if x not in {'cltv',␣
 ↪'dti'}])],
     remainder='passthrough')


#preprocessor = Pipeline([('num_preprocessing', num_pre),('cat_preprocessing',␣
 ↪cat_pre)])

pipe2 = Pipeline([("pre_processing", num_preprocessing),
                 ("model", LogisticRegression())])

pipe2.fit(X_train[num_features], y_train)

accuracy = accuracy_score(y_test, pipe2.predict(X_test[num_features]))
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.98
```

[216]:
```
preprocessing = ColumnTransformer([
    ("num_pre1", num_pre1, ['cltv', 'dti']),
    ("num_pre2", num_pre2, [x for x in num_features if x not in {'cltv',␣
 ↪'dti'}]),
    ("cat_pre1", cat_pre1, ['flag_fthb']),
    ("cat_pre2", cat_pre2, ['cnt_borr']),
    ("cat_pre3", cat_pre3, [x for x in cat if x not in {"flag_fthb",␣
 ↪"cnt_borr"}])])

logistic_pipe  = Pipeline([
    ("pre_processing", preprocessing),
    ("model", LogisticRegression())])

logistic_pipe.fit(pd.concat([X_train[num_features], X_train[cat].
 ↪astype('object')], axis=1), y_train)
```

[216]:
```
Pipeline(steps=[('pre_processing',
                 ColumnTransformer(transformers=[('num_pre1',
                                                  Pipeline(steps=[('num_clean',
CleanDataTransformer(value_to_replace='999')),
('num_outliers',
IQRBasedOutlierRemoverEnhanced()),
                                                                  ('num_impute',
SimpleImputer(strategy='median')),
                                                                  ('num_scale',
StandardScaler())]),
                                                  ['cltv', 'dti']),
                                                 ('num_pre2',
Pipeline(steps=[('num_outliers',
```

```
                                                                    IQRBas…
SimpleImputer(fill_value='missing',
 strategy='constant')),
                                                                   ('cat_encode',
OneHotEncoder(drop='first'))]),
                                                  ['cnt_borr']),
                                                 ('cat_pre3',
                                                  Pipeline(steps=[('cat_impute',
SimpleImputer(fill_value='missing',
 strategy='constant')),
                                                                   ('cat_encode',
OneHotEncoder(drop='first'))]),
                                                  ['flag_sc', 'prepaid',
                                                   'loan_purpose', 'prop_type',
                                                   'ppmt_pnlty',
                                                   'prod_type'])])),
                ('model', LogisticRegression())])
```

[218]:
```python
accuracy = accuracy_score(y_test, logistic_pipe.predict(pd.
 ↪concat([X_test[num_features], X_test[cat].astype('object')], axis=1)))
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 1.00
```

5 Discussion & Conclusions

In this section you should provide a general overview of your final model, its performance, and reliability. You should discuss what the implications of your model are in terms of the included features, predictive performance, and anything else you think is relevant.

This should be written with a target audience of a government official or charity directy, who is understands the pressing challenges associated with ageining and dementia but may only have university level mathematics (not necessarily postgraduate statistics or machine learning). Your goal should be to highlight to this audience how your model can useful. You should also mention potential limitations of your model.

Finally, you should include recommendations on potential lifestyle changes or governmental/societal interventions to reduce dementia risk.

Keep in mind that a negative result, i.e. a model that does not work well predictively, that is well explained and justified in terms of why it failed will likely receive higher marks than a model with strong predictive performance but with poor or incorrect explinations / justifications.

6 References

Include references if any

[ ]:
```
# Run the following to render to PDF
!jupyter nbconvert --to pdf project2.ipynb
```

Created in Deepnote