
Task Description

This is your first open task. You will create a machine learning model in a semi-guided environment. There is some information that we deliberately do not provide. So pleease do you own research.

Case:

- You are working as a data scientist in a risk analyst team in a finance industry.
- Your company generates profit by giving loans to customers.
- However, your company might suffer loss if the customer did not pay the loan back (we called it as default customer).
- To minimize the loss, the simple thing to do is to
  prevent bad application (who later become a default customer) get the loan.
- As a dta scientist, you want to create a classifier model to classify good or bad applicants from the given customer data **to minimize the potential loss**.

Rules:

- Please **create & collect** you analysis in Jupyter Notebook / Google Collaboratory format
- **State & explain** your answer on the Jupyter Notebook / Google Collaboratory
- Makesure your code is clean & explainable

Dataset:

- We will use a public dataset: https://www.kaggle.com/datasets/laotse/credit\-risk\-dataset
- Your target variable is **loan status (0 is non default, 1 is default)**

Task 1: Data Preparation [score: 30]

Credit Risk Dataset

This dataset contains columns simulating credit bureau data Detailed data description of Credit Risk dataset:

Feature Name	Description
person_age	Age
person_income	Annual Income
person_home_ownership	Home ownership
person_emp_length	Employment length (in years)
loan_intent	Loan intent
loan_grade	Loan grade
loan_amnt	Loan amount
loan_int_rate	Interest rate
loan_status	Loan status (0 is non default 1 is default)
loan_percent_income	Percent income
cb_person_default_on_file	Historical default
cb_preson_cred_hist_length	Credit history length

i. **Load your data** correctly. [score: 2]

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#Load data
data = pd.read_csv("credit_risk_dataset.csv")

print("Data shape raw =", data.shape)
data.head()
```

```
Data shape raw = (32581, 12)
```

Out[2]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	lo
0	22	59000	RENT	123.0	PERSONAL	D	35000	
1	21	9600	OWN	5.0	EDUCATION	B	1000	
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	
3	23	65500	RENT	4.0	MEDICAL	C	35000	
4	24	54400	RENT	8.0	MEDICAL	C	35000	

In [3]:
```
#Check any data duplicate
data.duplicated().sum()
```

Out[3]: 165

In [4]:
```
#Drop any duplicates
data_dropped = data.drop_duplicates(keep = 'first')
print("Data shape after dropping :",data_dropped.shape)
```

```
Data shape after dropping : (32416, 12)
```

ii. **Split your data** so that you can tune the model & predict correctly. [score: 3]

- SPLIT INPUT-OUTPUT DATA

In [5]:
```
#Create split input-output data function
def split_input_output(data, target_column):
    """
    A function to split input and output data
    :param data: <pandas dataframe> all data sample
    :param target_column: <string> output column name
    :return input_data: <pandas dataframe> input data
    :return output_data: <pandas series> data output
    """
    input_data = data.drop(columns = target_column)
    output_data = data[target_column]

    return input_data, output_data
```

In [6]:
```
X, y = split_input_output(data = data_dropped,
                          target_column = "loan_status")


print("X shape:", X.shape)
print("Y shape:", y.shape)
```

```
X shape: (32416, 11)
Y shape: (32416,)
```

In [7]:
```
X.head()
```

Out[7]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	lo
0	22	59000	RENT	123.0	PERSONAL	D	35000	
1	21	9600	OWN	5.0	EDUCATION	B	1000	
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	
3	23	65500	RENT	4.0	MEDICAL	C	35000	
4	24	54400	RENT	8.0	MEDICAL	C	35000	

In [8]:
```
y.head()
```

Out[8]:
```
0    1
1    0
2    1
3    1
4    1
Name: loan_status, dtype: int64
```

- SPLIT TRAIN-VALID-TEST DATA

In [9]:
```
#Import train-test splitting library from sklearn (scikit learn)
from sklearn.model_selection import train_test_split

#Create split train-valid-test function
def split_train_test (X, y, test_size, seed):

    """
    A function to split train-test data
    :param X: <pandas dataframe> input data
    :param y: <pandas series> output data
    :param test_size: <float> test size between 0 - 1
    :param seed: <int> random state
    :return X_train: <pandas dataframe> input train data
    :return X_test: <pandas dataframe> input test data
    :return y_train: <pandas series> output train data
    :return y_test: <pandas series> output test data
    """

    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                        test_size = test_size,
                                                        random_state = seed)


    return X_train, X_test, y_train, y_test
```

In [10]:
```
#Split the train & not train
X_train, X_not_train, y_train, y_not_train = split_train_test(X, y,
                                                              test_size = 0.2,
                                                              seed= 123)


print("X train shape     :", X_train.shape)
print("y train shape     :", y_train.shape)
print("X not train shape :", X_not_train.shape)
print("y not train shape :", y_not_train.shape, "\n")


#Split the valid & test
```

```
X_valid, X_test, y_valid, y_test = split_train_test(X = X_not_train, y = y_not_train,
                                                    test_size = 0.5,
                                                    seed= 123)
print("X valid shape :", X_valid.shape)
print("y valid shape :", y_valid.shape)
print("X test shape  :", X_test.shape)
print("y test shape  :", y_test.shape, "\n")

#Validate the split
print(len(X_train)/len(X))  # should be 0.8
print(len(X_valid)/len(X))  # should be 0.1
print(len(X_test)/len(X))   # should be 0.1
```

```
X train shape     : (25932, 11)
y train shape     : (25932,)
X not train shape : (6484, 11)
y not train shape : (6484,)

X valid shape : (3242, 11)
y valid shape : (3242,)
X test shape  : (3242, 11)
y test shape  : (3242,)

0.7999753208292202
0.10001233958538994
0.10001233958538994
```

- SPLIT NUMERICAL AND CATEGORICAL VALUE

In [11]:

```
#Separate Numerical & Categorical Value
def split_num_cat (data, num_cols, cat_cols):

    """
    A function to split numerical & categorical input
    :param data: <pandas dataframe> input train data
    :param num_cols: <list> list of numerical columns
    :param cat_cols: <list> list of categorical columns
    :return num_column: <pandas dataframe> numerical train data input
    :return cat_column: <pandas dataframe> categorical train input
    """

    num_column = data[num_cols]
    cat_column = data[cat_cols]

    return num_column, cat_column
```

In [12]:

```
#Check data type for each column
print(X_train.dtypes)
X_train.columns
```

```
person_age                      int64
person_income                   int64
person_home_ownership          object
person_emp_length             float64
loan_intent                    object
loan_grade                     object
loan_amnt                       int64
loan_int_rate                 float64
loan_percent_income           float64
cb_person_default_on_file      object
cb_person_cred_hist_length      int64
dtype: object
```

Out[12]:
```
Index(['person_age', 'person_income', 'person_home_ownership',
       'person_emp_length', 'loan_intent', 'loan_grade', 'loan_amnt',
       'loan_int_rate', 'loan_percent_income', 'cb_person_default_on_file',
       'cb_person_cred_hist_length'],
      dtype='object')
```

In [13]:
```
X_train.head()
```

Out[13]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade
5828	25	47000	OWN	9.0	PERSONAL	A
27467	27	140004	MORTGAGE	11.0	VENTURE	A
3240	25	66660	RENT	0.0	EDUCATION	B
9470	25	87000	RENT	3.0	MEDICAL	B
29011	28	42000	MORTGAGE	4.0	HOMEIMPROVEMENT	B

In [14]:
```
# Split the data

numerical_columns = ['person_age', 'person_income',  'person_emp_length', 'loan_amnt', 'loan_int_rate', '

categorical_columns = ['person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file']

X_train_num, X_train_cat = split_num_cat(data = X_train,
                                         num_cols = numerical_columns,
                                         cat_cols = categorical_columns)

print("Data num shape:", X_train_num.shape)
print("Data cat shape:", X_train_cat.shape)
```

```
Data num shape: (25932, 7)
Data cat shape: (25932, 4)
```

In [15]:
```
X_train_num.head()
```

Out[15]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_percent_income	cb_person_c
5828	25	47000	9.0	7000	7.51	0.15	
27467	27	140004	11.0	9800	5.42	0.07	
3240	25	66660	0.0	3500	NaN	0.05	
9470	25	87000	3.0	8000	10.37	0.09	
29011	28	42000	4.0	10000	10.75	0.24	

In [16]:
```
X_train_cat.head()
```

Out[16]:

	person_home_ownership	loan_intent	loan_grade	cb_person_default_on_file
5828	OWN	PERSONAL	A	N
27467	MORTGAGE	VENTURE	A	N
3240	RENT	EDUCATION	B	N
9470	RENT	MEDICAL	B	N
29011	MORTGAGE	HOMEIMPROVEMENT	B	N

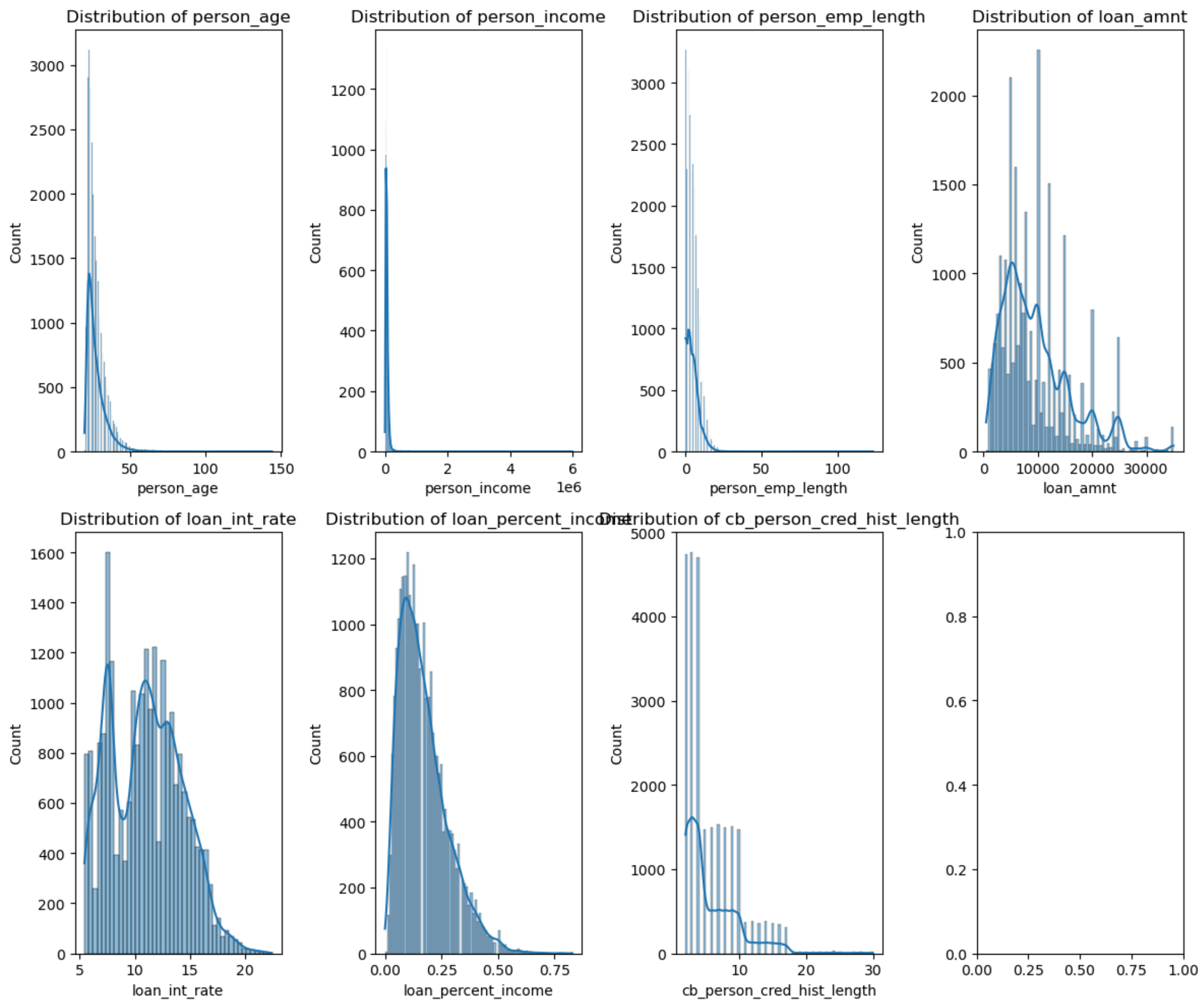
iii. **Perform EDA** and **conclude your data preprocessing plan** from that. [score: 10]

In [17]:
```python
#Find missing value
100 * (X_train.isna().sum(0) / len(X_train))
```

Out[17]:
```
person_age                    0.000000
person_income                 0.000000
person_home_ownership         0.000000
person_emp_length             2.834336
loan_intent                   0.000000
loan_grade                    0.000000
loan_amnt                     0.000000
loan_int_rate                 9.501774
loan_percent_income           0.000000
cb_person_default_on_file     0.000000
cb_person_cred_hist_length    0.000000
dtype: float64
```

In [18]:
```python
#Check numerical value distribution

#Import library
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Plot histogram
fig, ax = plt.subplots(nrows=2, ncols=4, figsize=(12, 10))
axes = ax.flatten()

for i, col in enumerate(X_train_num.columns):
    sns.histplot(X_train_num[col], ax=axes[i], kde=True)
    axes[i].set_title(f'Distribution of {col}')

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

```
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [19]:
```
X_train_num.describe()
```

Out[19]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_percent_income	cb_pers
count	25932.000000	2.593200e+04	25197.000000	25932.000000	23468.000000	25932.000000	
mean	27.731490	6.610317e+04	4.779140	9569.822035	11.014987	0.170029	
std	6.337296	6.445098e+04	4.149961	6292.989974	3.244845	0.106446	
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	0.000000	
25%	23.000000	3.840000e+04	2.000000	5000.000000	7.900000	0.090000	
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	0.150000	
75%	30.000000	7.963125e+04	7.000000	12000.000000	13.470000	0.230000	
max	144.000000	6.000000e+06	123.000000	35000.000000	22.480000	0.830000	

In [20]:
```
#Check categorical value
categorical_columns = ['person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file']

for col in categorical_columns:
    print(f"Value counts for {col} (normalized):")
    print(X_train[col].value_counts(normalize=True))
    print("-" * 40)
```

```
Value counts for person_home_ownership (normalized):
person_home_ownership
RENT        0.507365
MORTGAGE    0.409417
OWN         0.079901
OTHER       0.003316
Name: proportion, dtype: float64
----------------------------------------
Value counts for loan_intent (normalized):
loan_intent
EDUCATION            0.197941
MEDICAL              0.185562
VENTURE              0.176886
PERSONAL             0.170176
DEBTCONSOLIDATION    0.160034
HOMEIMPROVEMENT      0.109402
Name: proportion, dtype: float64
----------------------------------------
Value counts for loan_grade (normalized):
loan_grade
A    0.330325
B    0.321880
C    0.196205
D    0.112448
E    0.029577
F    0.007250
G    0.002314
Name: proportion, dtype: float64
----------------------------------------
Value counts for cb_person_default_on_file (normalized):
cb_person_default_on_file
N    0.824233
Y    0.175767
Name: proportion, dtype: float64
----------------------------------------
```

In [21]:

```
#Check class proportion
y_train.value_counts(normalize = True)
```

Out[21]:

```
loan_status
0    0.781775
1    0.218225
Name: proportion, dtype: float64
```

Preprocessing Plan:

- Drop data anomaly in `data in person_age`, `person_emp_length`, and `cb_person_cred_hist_length`
- Impute missing value in `person_emp_length` and `loan_int_rate` with median
- Encode `loan_grade` with `Label Encoding`
- Encode `person_home_ownership`, `loan_intent`, `cb_person_default_on_file` with `OHE`
- Applying oversampling to even out the class with SMOTE
- Standardize using `StandardScaler`

iv. **Perform data preprocessing** according to plan on poin iii. [score: 15]

- Drop data anomaly in `person_age`, `person_emp_length`, and `cb_person_cred_hist_length`

In [22]:

```
condition = (
    (X_train['person_age']>80) |
    (X_train['person_emp_length']>45) |
    (X_train['person_age'] - X_train['cb_person_cred_hist_length'] > 55)
)
```

```python
# Get the indexes where the condition is True
idx_to_drop = X_train.index[condition].tolist()

print(f'Number of index to drop:', len(idx_to_drop),'\n')
```

```
Number of index to drop: 8
```

In [23]:
```python
X_train_num.loc[idx_to_drop]
```

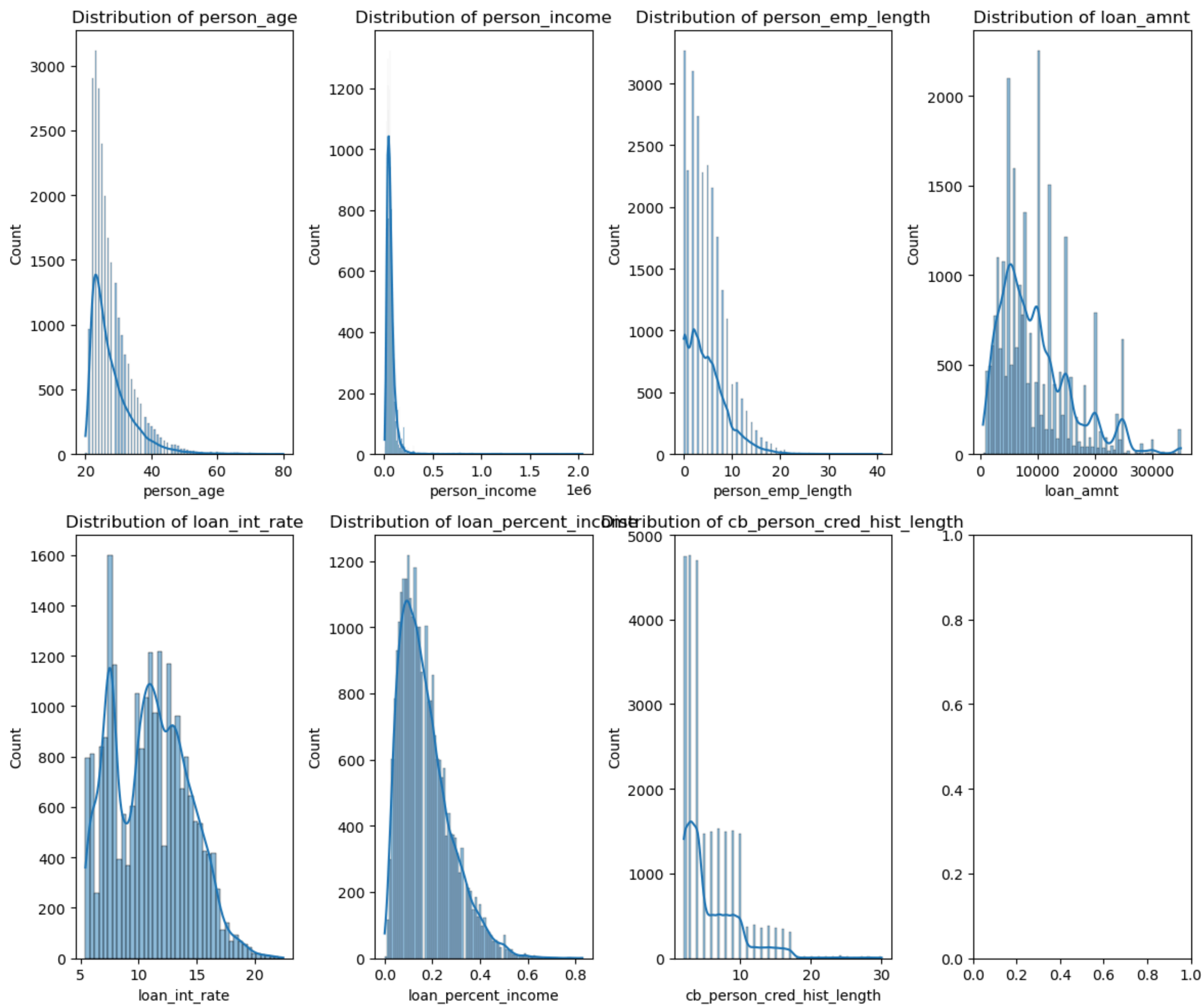
Out[23]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_percent_income	cb_person_c
32416	94	24000	1.0	6500	NaN	0.27	
575	123	80004	2.0	20400	10.25	0.25	
32297	144	6000000	12.0	5000	12.73	0.00	
210	21	192000	123.0	20000	6.54	0.10	
0	22	59000	123.0	35000	16.02	0.59	
183	144	200000	4.0	6000	11.86	0.03	
747	123	78000	7.0	20000	NaN	0.26	
32506	84	94800	2.0	10000	7.51	0.11	

In [24]:
```python
X_train_num_dropped = X_train_num.drop(idx_to_drop)
X_train_cat_dropped = X_train_cat.drop(idx_to_drop)
y_train_dropped = y_train.drop(idx_to_drop)
```

In [25]:
```python
# Plot histogram
fig, ax = plt.subplots(nrows=2, ncols=4, figsize=(12, 10))
axes = ax.flatten()

for i, col in enumerate(X_train_num.columns):
    sns.histplot(X_train_num_dropped[col], ax=axes[i], kde=True)
    axes[i].set_title(f'Distribution of {col}')

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

```
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\ASUS\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is
deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



- Impute missing value in `person_emp_length` and `loan_int_rate` with median

In [26]:
```
#Check missing value
X_train_num_dropped.isna().sum()
```

Out[26]:
```
person_age                       0
person_income                    0
person_emp_length              735
loan_amnt                        0
loan_int_rate                 2462
loan_percent_income              0
cb_person_cred_hist_length       0
dtype: int64
```

In [27]:
```
#Import sklearn imputer
from sklearn.impute import SimpleImputer

#Create fitting numerical imputer function
def num_imputer_fit (data):
    """
    A function to fit numerical imputers
    :param data: <dataframe> numerical train data input
    :return num_imputer: numerical imputer method
    """
    num_imputer = SimpleImputer(missing_values = np.nan,
                                strategy = "median")
    num_imputer.fit(data)
```

```python
    return num_imputer

#Create transforming numerical imputer function
def num_imputer_transform (data, imputer):
    """
    A function to transform numerical imputers
    :param data: <dataframe> numerical train data input
    :param imputer: imputer method
    :return num_imputed_data: <dataframe> imputed numerical train data input
    """
    imputed_data = imputer.transform(data)
    num_imputed_data = pd.DataFrame(imputed_data)

    num_imputed_data.columns = data.columns
    num_imputed_data.index = data.index

    return num_imputed_data
```

In [28]:
```python
# Get the numerical imputer
num_imputer = num_imputer_fit (data = X_train_num_dropped)

# Transform the data
X_train_num_imputed = num_imputer_transform(data = X_train_num_dropped, imputer = num_imputer)
```

In [29]:
```python
#Check missing value after imputed
X_train_num_imputed.isna().sum()
```

Out[29]:
```
person_age                    0
person_income                 0
person_emp_length             0
loan_amnt                     0
loan_int_rate                 0
loan_percent_income           0
cb_person_cred_hist_length    0
dtype: int64
```

In [30]:
```python
X_train_num_imputed.shape
```

Out[30]:
```
(25924, 7)
```

- Encode `loan_grade` with `Label Encoding`

In [31]:
```python
def label_encoder(data, label_column_name, sorted_grades):
    """
    Function to encoded ordinal classification value data

    Parameters:
    - data (pd.DataFrame): The input DataFrame.
    - column_name (str): The name of the categorical column to be sorted and encoded.
    - sorted_grades (list): The desired order of categories from lowest to highest.

    Returns:
    - pd.DataFrame: A DataFrame with the original column replaced by the encoded version.
    """
    # Convert the column to categorical with the specified order
    data[label_column_name] = pd.Categorical(data[label_column_name], categories=sorted_grades, ordered=T

    # Create a new column for the encoded values
    encoded_column_name = f"{label_column_name}_LABELED"
    data[encoded_column_name] = data[label_column_name].cat.codes

    # Drop the original column
    data_labenc = data.drop(columns=[label_column_name])
```

```
    return data_labenc
```

In [32]:
```
# Define the sorted grades
sorted_grades = ['G', 'F', 'E', 'D', 'C', 'B', 'A']

# Apply the function to your DataFrame
X_train_cat_le = label_encoder(X_train_cat_dropped, 'loan_grade', sorted_grades)

# Display the result
X_train_cat_le.head()
```

Out[32]:

	person_home_ownership	loan_intent	cb_person_default_on_file	loan_grade_LABELED
5828	OWN	PERSONAL	N	6
27467	MORTGAGE	VENTURE	N	6
3240	RENT	EDUCATION	N	5
9470	RENT	MEDICAL	N	5
29011	MORTGAGE	HOMEIMPROVEMENT	N	5

- Encode `person_home_ownership`, `loan_intent`, `cb_person_default_on_file` with `OHE`

In [33]:
```
from sklearn.preprocessing import OneHotEncoder

def one_hot_encodes(data, ohe_column_name):
    """
    Function to apply One Hot Encoding to non ordinal categorical value

    Parameters:
    - data (pd.DataFrame): The input DataFrame.
    - columns_to_encode (list): List of column names to one-hot encode.

    Returns:
    - pd.DataFrame: A DataFrame with one-hot encoded columns merged and original columns dropped.
    """
    # Initialize the OneHotEncoder
    ohe = OneHotEncoder(sparse=False, drop=None)

    # Fit and transform the specified columns
    ohe_encoded = ohe.fit_transform(data[ohe_column_name])

    # Get the column names for the encoded DataFrame
    ohe_columns = ohe.get_feature_names_out(ohe_column_name)

    # Convert the array to a DataFrame with proper column names and original indices
    ohe_df = pd.DataFrame(ohe_encoded, columns=ohe_columns, index=data.index)

    # Drop the original columns and concatenate the encoded DataFrame
    data_encoded = pd.concat([data.drop(columns=ohe_column_name), ohe_df], axis=1)

    return data_encoded
```

In [34]:
```
# Specify the nominal columns to encode
ohe_column_name = ['person_home_ownership', 'loan_intent', 'cb_person_default_on_file']

# Apply the function to your DataFrame
X_train_cat_encoded = one_hot_encodes(X_train_cat_le, ohe_column_name)

# Display the resulting DataFrame
X_train_cat_encoded.head()
```

```
C:\Users\ASUS\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

Out[34]:

	loan_grade_LABELED	person_home_ownership_MORTGAGE	person_home_ownership_OTHER	person_home_ow
5828	6	0.0	0.0	
27467	6	1.0	0.0	
3240	5	0.0	0.0	
9470	5	0.0	0.0	
29011	5	1.0	0.0	

In [35]:

```
X_train_cat_encoded.shape
```

Out[35]: (25924, 13)

- Join the data

In [36]:

```
def concat_data(num_data, cat_data):
    """
    A function to join preprocessed numerical & categorical data
    :param num_data: <dataframe> preprocessed numerical data
    :param cat_data: <dataframe> preprocessed categorical data
    :return concated_data: <dataframe> preprocessed input train data
    """
    num_data = num_data.reset_index(drop=True)
    cat_data = cat_data.reset_index(drop=True)

    concated_data = pd.concat([num_data, cat_data] , axis = 1)

    return concated_data
```

In [37]:

```
# Concat the data
X_train_concat = concat_data (num_data = X_train_num_imputed,
                              cat_data = X_train_cat_encoded)

print("Numerical data shape  :", X_train_num_imputed.shape)
print("Categorical data shape:", X_train_cat_encoded.shape)
print("Concat data shape     :", X_train_concat.shape)
```

```
Numerical data shape  : (25924, 7)
Categorical data shape: (25924, 13)
Concat data shape     : (25924, 20)
```

- Standardize using `StandardScaler`

In [161…

```
from sklearn.preprocessing import StandardScaler

def fit_scaler (data):
    """
    A function to fit the scaler
    :param data: <dataframe> input train data
    :return scaler: fitted scaler object
    """
    scaler = StandardScaler()
    scaler = scaler.fit(data)
```

```
    return scaler

def transform_scaler(data, scaler):
    """
    A function to transform scaled data
    :param data: <dataframe> input train data
    :param scaler: scaler method
    :return scaled data: <dataframe> scaled input train data
    """
    standardized_data_raw = scaler.transform(data)
    standardized_data = pd.DataFrame(standardized_data_raw)
    standardized_data.columns = data.columns
    standardized_data.index = data.index

    return standardized_data
```

In [131…

```
# Fit the scaler
fit_scaler = fit_scaler (data = X_train_concat)

# Transform the scaler
X_train_clean = transform_scaler (data = X_train_concat,
                                  scaler = fit_scaler)
```

In [132…

```
X_train_clean.describe().round(4)
```

Out[132…

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_percent_income	cb_person_c
count	25924.0000	25924.0000	25924.0000	25924.0000	25924.0000	25924.0000	
mean	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	
std	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	
min	-1.2487	-1.1701	-1.1993	-1.4415	-1.8118	-1.5977	
25%	-0.7629	-0.5195	-0.6941	-0.7262	-0.8172	-0.7520	
50%	-0.2771	-0.2055	-0.1889	-0.2493	-0.0073	-0.1881	
75%	0.3707	0.2592	0.5689	0.3866	0.6794	0.5637	
max	8.4675	37.3347	9.1570	4.0428	3.7149	6.2020	

In [ ]:

- preprocess All data:

In [133…

```
X_train_dropped = X_train.drop(idx_to_drop)
y_train_dropped = y_train.drop(idx_to_drop)
```

In [159…

```
def preprocess_data (data, num_cols, cat_cols, label_column_name, sorted_grades, ohe_column_name, scaler

    # Split categorical and numerical data
    num_data, cat_data = split_num_cat (data = data,
                                        num_cols = num_cols,
                                        cat_cols = cat_cols)

    # Get numerical imputer
    num_imputer = num_imputer_fit (data = num_data)

    num_imputed = num_imputer_transform(data = num_data,
                                        imputer = num_imputer)

    # Get categorical imputer
    cat_labenc = label_encoder(data = cat_data,
```

```python
                                label_column_name = label_column_name,
                                sorted_grades = sorted_grades)

    cat_encoded = one_hot_encodes(data = cat_labenc,
                                   ohe_column_name = ohe_column_name)

    # Concatenate categorical and numerical data
    data_concated = concat_data (num_data = num_imputed,
                                 cat_data = cat_encoded)

    # Fit the scaler
    scaler = fit_scaler(data_concated)

    # Transform the scaler
    data_cleaned = transform_scaler (data = data_concated,
                                      scaler = scaler)

    print("Numerical data shape  :", num_imputed.shape)
    print("Categorical data shape:", cat_encoded.shape)
    print("Concat data shape     :", data_concated.shape, "\n")
    print("Original data shape:", data.shape)
    print("Preprocessed data shape  :", data_cleaned.shape, "\n")


    return data_cleaned
```

In [150…

```python
numerical_columns = ['person_age', 'person_income',  'person_emp_length', 'loan_amnt', 'loan_int_rate', '
categorical_columns = ['person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file']

label_column_names = 'loan_grade'
sorted_grades = ['G', 'F', 'E', 'D', 'C', 'B', 'A']

ohe_column_name = ['person_home_ownership', 'loan_intent', 'cb_person_default_on_file']
```

In [162…

```python
X_train_cleaned = preprocess_data (data = X_train_dropped,
                                   num_cols = numerical_columns,
                                   cat_cols = categorical_columns,
                                   label_column_name = label_column_names,
                                   sorted_grades = sorted_grades,
                                   ohe_column_name = ohe_column_name,
                                   scaler = scaler)
```

```
Numerical data shape  : (25924, 7)
Categorical data shape: (25924, 13)
Concat data shape     : (25924, 20)

Original data shape: (25924, 11)
Preprocessed data shape  : (25924, 20)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_11596\2085028562.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy
  data[label_column_name] = pd.Categorical(data[label_column_name], categories=sorted_grades, ordered=Tru
e)
C:\Users\ASUS\AppData\Local\Temp\ipykernel_11596\2085028562.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy
  data[encoded_column_name] = data[label_column_name].cat.codes
C:\Users\ASUS\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:868: FutureWarning: `sparse`
was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unles
s you leave `sparse` to its default value.
  warnings.warn(
```

In [61]:

```python
X_train_cleaned.shape
```

Out[61]:

```
(25924, 20)
```

- Applying oversampling to even out the class with `SMOTE`

In [51]:

```python
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=123)
X_train_smote, y_train_smote = smote.fit_resample(X_train_cleaned, y_train_dropped)

# Check the new shapes and class distribution
print("Shape of X_train after SMOTE:", X_train_smote.shape)
print("Shape of y_train after SMOTE:", y_train_smote.shape)
print("Class distribution after SMOTE:", y_train_smote.value_counts(normalize=True))
```

```
Shape of X_train after SMOTE: (40532, 20)
Shape of y_train after SMOTE: (40532,)
Class distribution after SMOTE: loan_status
0    0.5
1    0.5
Name: proportion, dtype: float64
```

Task 2: Modeling [score: 40]

i. **Define your metrics** (you can use more than 1 metrics, just explain why) **for optimizing the model.** [score: 6]

- `Recall (Sensitivity)` : The ratio of true positives to actual positives.

To ensures that most of the actual defaulting customers (positives) are identified. Missing a default (false negative) is a high-risk scenario for banks, as it means giving a loan to a customer who is likely to default.

- `Precision` : The ratio of true positives to the total predicted positives.

To ensure that customers flagged as likely defaulters actually default. This avoids rejecting loans for customers who are non-defaulters.

- `F1 Score` : The harmonic mean of precision and recall.

The F1 Score balances precision and recall, especially useful when there's a trade-off between the two.

ii. **Define your baseline model** (explain why) and **print out** the score that you want to beat. [score: 7]

In [52]:
```
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
```

- Baseline model of the NON SMOTE data

In [53]:
```
# Buat objek
dummy_clf = DummyClassifier(strategy = "most_frequent")

# Lakukan fit, untuk data y_train saja
dummy_clf.fit(X = X_train_cleaned,
              y = y_train_dropped)
```

Out[53]:
```
▾ DummyClassifier
DummyClassifier(strategy='most_frequent')
```

In [54]:
```
# Predict
y_pred_dummy = dummy_clf.predict(X_train_cleaned)
```

In [55]:
```
# Tampilkan confusion matrix
from sklearn.metrics import confusion_matrix

cf_base = confusion_matrix(y_true = y_train_dropped,
                 y_pred = y_pred_dummy)

cf_base

# [[TP, FP]]
# [[FN, TN]]
```

Out[55]:
```
array([[20266,     0],
       [ 5658,     0]], dtype=int64)
```

In [56]:
```
accuracy_score(y_true = y_train_dropped,
               y_pred = y_pred_dummy)
```

Out[56]:
```
0.7817466440364141
```

Due the imbalance of the data, where Non-Defaulters is much higher than the Defaulters, `DummyClassifier` with `most_frequent` model is used for baseline and Accuracy metrics is used to assess overall performance. Pragmatically, this will result in the highest baseline score due to its most frequent nature. The model should outperform this.

- Baseline with SMOTE Applied data

In [57]:
```
# Buat objek
dummy_clf = DummyClassifier(strategy = "uniform")

# Lakukan fit, untuk data y_train saja
dummy_clf.fit(X = X_train_smote,
              y = y_train_smote)
```

Out[57]:
```
▾ DummyClassifier
DummyClassifier(strategy='uniform')
```

In [63]:
```
# Predict
y_pred_dummy_2 = dummy_clf.predict(X_train_smote)
```

In [64]:
```
# Tampilkan confusion matrix
from sklearn.metrics import confusion_matrix

confusion_matrix(y_true = y_train_smote,
                 y_pred = y_pred_dummy_2)

# [[TP, FP]]
# [[FN, TN]]
```

Out[64]:
```
array([[10147, 10119],
       [10262, 10004]], dtype=int64)
```

In [65]:
```
accuracy_2 = accuracy_score(y_train_smote, y_pred_dummy_2)
print("Accuracy Score  :", accuracy_2)

recall_2 = recall_score(y_train_smote, y_pred_dummy_2)
print("Recall Score    :", recall_2)

precision_2 = precision_score(y_train_smote, y_pred_dummy_2)
print("Precision Score :", precision_2)

f1_2 = f1_score(y_train_smote, y_pred_dummy_2)
print("F1 Score        :", f1_2)
```

```
Accuracy Score  : 0.49716273561630314
Recall Score    : 0.49363465903483666
Precision Score : 0.4971425731749739
F1 Score        : 0.495382406100671
```

Due the the balanced data, where Non-Defaulters is as much as the Defaulters, `DummyClassifier` with `uniform` method is used for baseline and `Recall metrics` is used to score True Positive Rate. The result is similar to other metrics (`Accuracy` , `Precision` and `F1`) due to data is being balanced out.

iii. Do a proper best model search & hyperparameter tuning (explain why you choose those models and why you choose those hyperparameter). [score: 17]

In [67]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import cross_val_score
```

- CV score for `NON-SMOTE` data with `Logistic Regression`

In [68]:
```
model = LogisticRegression(class_weight={0: 1/0.781775,
                                         1: 1/0.218225},
                           max_iter = 200,
                           random_state=123)

scores_1a = cross_val_score(model,
                         X = X_train_cleaned,
                         y = y_train_dropped,
                         cv=5,
                         scoring='recall')

print(f'IMB LR - Recall CV scores     : {scores_1a}')
print(f'IMB LR - Recall mean CV score : {scores_1a.mean()}')

scores_1b = cross_val_score(model,
                         X = X_train_cleaned,
                         y = y_train_dropped,
                         cv=5,
                         scoring='precision')
```

```
print(f'IMB LR & Precision CV scores     : {scores_1b}')
print(f'IMB LR & Precision mean CV score : {scores_1b.mean()}')
```

```
IMB LR - Recall CV scores     : [0.78868258 0.77473498 0.79063604 0.76943463 0.80194518]
IMB LR - Recall mean CV score : 0.7850866833503607
IMB LR & Precision CV scores     : [0.50710631 0.52140309 0.52034884 0.51085044 0.52948044]
IMB LR & Precision mean CV score : 0.5178378245438783
```

- CV score for `NON-SMOTE` data with `Random Forest Classifier`

In [69]:
```
model = RandomForestClassifier(class_weight={0: 1/0.781775,
                                             1: 1/0.218225},
                           random_state=123)

scores_2a = cross_val_score(model,
                         X =X_train_cleaned,
                         y = y_train_dropped,
                         cv=5,
                         scoring='recall')

print(f'IMB RFC & Recall CV scores    : {scores_2a}')
print(f'IMB RFC & Recall mean CV score: {scores_2a.mean()}')

scores_2b = cross_val_score(model,
                         X = X_train_cleaned,
                         y = y_train_dropped,
                         cv=5,
                         scoring='precision')

print(f'IMB RFC & Precision CV scores    : {scores_2b}')
print(f'IMB RFC & Precision mean CV score: {scores_2b.mean()}')
```

```
IMB RFC & Recall CV scores    : [0.71706454 0.70229682 0.71466431 0.69611307 0.73032714]
IMB RFC & Recall mean CV score: 0.7120931787435991
IMB RFC & Precision CV scores    : [0.9794686  0.97906404 0.98179612 0.97044335 0.96158324]
IMB RFC & Precision mean CV score: 0.9744710682045072
```

- CV score for `SMOTE` data with `Logistic Regression`

In [70]:
```
model = LogisticRegression( max_iter = 200,
                           random_state=123)

scores_3a = cross_val_score(model,
                         X = X_train_smote,
                         y = y_train_smote,
                         cv=5,
                         scoring='recall')

print(f'SMOTE LR & Recall CV scores     : {scores_3a}')
print(f'SMOTE LR & Recall mean CV score : {scores_3a.mean()}')

scores_3b = cross_val_score(model,
                         X = X_train_smote,
                         y = y_train_smote,
                         cv=5,
                         scoring='precision')

print(f'SMOTE LR & Precision CV scores     : {scores_3b}')
print(f'SMOTE LR & Precision mean CV score : {scores_3b.mean()}')
```

```
SMOTE LR & Recall CV scores     : [0.77152726 0.7947706  0.79792746 0.79447323 0.80162842]
SMOTE LR & Recall mean CV score : 0.7920653949865807
SMOTE LR & Precision CV scores     : [0.78884965 0.80309073 0.79891304 0.7960445  0.80301532]
SMOTE LR & Precision mean CV score : 0.7979826482549452
```

- CV score for `SMOTE` data with `Random Forest Classifier`

In [71]:
```
model = RandomForestClassifier(random_state=123)

scores_4a = cross_val_score(model,
                            X = X_train_smote,
                            y = y_train_smote,
                            cv=5,
                            scoring='recall')

print(f'SMOTE RFC & Recall CV scores    : {scores_4a}')
print(f'SMOTE RFC & Recall mean CV score: {scores_4a.mean()}')

scores_4b = cross_val_score(model,
                            X = X_train_smote,
                            y = y_train_smote,
                            cv=5,
                            scoring='precision')

print(f'SMOTE RFC & Precision CV scores    : {scores_4b}')
print(f'SMOTE RFC & Precision mean CV score: {scores_4b.mean()}')
```

```
SMOTE RFC & Recall CV scores    : [0.78657784 0.89713863 0.96002961 0.9607698  0.95978288]
SMOTE RFC & Recall mean CV score: 0.9128597513630143
SMOTE RFC & Precision CV scores    : [0.9869969  0.98297297 0.97960725 0.97716437 0.97837022]
SMOTE RFC & Precision mean CV score: 0.9810223430909796
```

- Hyperparameter Tuning with `RandomForestClassifier` :
    - `Precision` & `Recall` metric for `NON-SMOTE` data
    - `Precision` & `Recall` metric for `SMOTE` data

- Preprocess Validation Data

In [164…
```
X_valid_cleaned = preprocess_data (data = X_valid,
                                   num_cols = numerical_columns,
                                   cat_cols = categorical_columns,
                                   label_column_name = label_column_names,
                                   sorted_grades = sorted_grades,
                                   ohe_column_name = ohe_column_name,
                                   scaler = scaler)
```

```
Numerical data shape  : (3242, 7)
Categorical data shape: (3242, 13)
Concat data shape     : (3242, 20)

Original data shape: (3242, 11)
Preprocessed data shape  : (3242, 20)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_11596\2085028562.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy
  data[label_column_name] = pd.Categorical(data[label_column_name], categories=sorted_grades, ordered=Tru
e)
C:\Users\ASUS\AppData\Local\Temp\ipykernel_11596\2085028562.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy
  data[encoded_column_name] = data[label_column_name].cat.codes
C:\Users\ASUS\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:868: FutureWarning: `sparse`
was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unles
s you leave `sparse` to its default value.
  warnings.warn(
```

In [166…

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

rf = RandomForestClassifier(random_state=123)

param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [10, 20, None],
    'class_weight': ['balanced']
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           scoring='recall',
                           cv=5)

grid_search.fit(X_train_cleaned, y_train_dropped)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print("Best Parameters:", best_params)

y_pred = best_model.predict(X_valid_cleaned)
print("Classification Report:\n", classification_report(y_valid, y_pred))
```

```
Best Parameters: {'class_weight': 'balanced', 'max_depth': 10, 'n_estimators': 150}
Classification Report:
               precision    recall  f1-score   support

           0       0.92      0.96      0.94      2510
           1       0.84      0.73      0.78       732

    accuracy                           0.91      3242
   macro avg       0.88      0.84      0.86      3242
weighted avg       0.90      0.91      0.90      3242
```

In [168…

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

rf = RandomForestClassifier(random_state=123)

param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [10, 20, None]
}

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           scoring='recall',
```

```
                            cv=5)

grid_search.fit(X_train_smote, y_train_smote)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print("Best Parameters:", best_params)

y_pred = best_model.predict(X_valid_cleaned)
print("Classification Report:\n", classification_report(y_valid, y_pred))
```

```
Best Parameters: {'max_depth': None, 'n_estimators': 150}
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.63      0.76      2510
           1       0.41      0.89      0.57       732

    accuracy                           0.69      3242
   macro avg       0.68      0.76      0.66      3242
weighted avg       0.83      0.69      0.72      3242
```

iv. **Define your best model** (explain why). [score: 10]

> best model is `RandomForestClassifier` with these parameters: `{'class_weight': 'balanced', 'max_depth': 10, 'n_estimators': 150}`
>
> that is applied to `NON-SMOTE` data

Task 3: Model Evaluation [score: 30]

- Preprocess the test data

In [170…

```
X_test_cleaned = preprocess_data (data = X_test,
                                  num_cols = numerical_columns,
                                  cat_cols = categorical_columns,
                                  label_column_name = label_column_names,
                                  sorted_grades = sorted_grades,
                                  ohe_column_name = ohe_column_name,
                                  scaler = scaler)
```

```
Numerical data shape  : (3242, 7)
Categorical data shape: (3242, 13)
Concat data shape     : (3242, 20)

Original data shape: (3242, 11)
Preprocessed data shape  : (3242, 20)
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_11596\2085028562.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy
  data[label_column_name] = pd.Categorical(data[label_column_name], categories=sorted_grades, ordered=Tru
e)
C:\Users\ASUS\AppData\Local\Temp\ipykernel_11596\2085028562.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy
  data[encoded_column_name] = data[label_column_name].cat.codes
C:\Users\ASUS\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:868: FutureWarning: `sparse`
was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unles
s you leave `sparse` to its default value.
  warnings.warn(
```

In [171…

```
best_model = RandomForestClassifier(class_weight = 'balanced',
                                    max_depth = 10,
                                    n_estimators = 150
                                   )

best_model.fit(X_train_cleaned, y_train_dropped)

best_model
```

Out[171…

```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=10, n_estimators=150)
```

In [172…

```
y_test_pred = best_model.predict(X_test_cleaned)
```

In [173…

```
# Tampilkan confusion matrix
cf_res = confusion_matrix(y_true = y_test,
                 y_pred = y_test_pred)

cf_res

# [[tp, fp]]
# [[fn, tn]]
```

Out[173…

```
array([[2420,  124],
       [ 171,  527]], dtype=int64)
```

In [175…

```
print(classification_report(y_true = y_test,
                            y_pred = y_test_pred,
                            target_names = ["non-defaulters", "defaulter"]))
```

```
                precision    recall  f1-score   support

non-defaulters       0.93      0.95      0.94      2544
     defaulter       0.81      0.76      0.78       698

      accuracy                           0.91      3242
     macro avg       0.87      0.85      0.86      3242
  weighted avg       0.91      0.91      0.91      3242
```

i. **How does your best model perform** in the test data? Is your best model good? [score: 10]

> The score is pretty high, with `91% overall acurracy` with the chosen best model.

ii. **Compare the financial impact** between your best model & baseline model. Is your best model better than the baseline model? Assumptions: [score: 20]

- if you falsely predict good applicants as bad, you would lose potential revenue of Rp 10.000.000/applicant on average.
- If you falsely predict bad applicants as good, you would lose Rp 30.000.000/applicant on average.

> Based on below comparison between the baseline model and the best model applied, the best model resulted in potentional lost of `Rp6.370.000.000` while the baseline has `Rp169.740.000.000`. This resulting in `96.25% of potential lost saving`

In [176…

```
cost_false_positive = 10_000_000  # Rp 10.000.000
cost_false_negative = 30_000_000  # Rp 30.000.000

# Calculate total cost
cfp_model = (cf_res[0, 1] * cost_false_positive)
cfn_model = (cf_res[1, 0] * cost_false_negative)
total_cost_model =  cfp_model + cfn_model

print("Confusion Matrix:")
print(cf_res)
print(f"FP Cost    : Rp {cfp_model}")
print(f"FN Cost    : Rp {cfn_model}")
print(f"Total Cost: Rp {total_cost_model}")
```

```
Confusion Matrix:
[[2420  124]
 [ 171  527]]
FP Cost    : Rp 1240000000
FN Cost    : Rp 5130000000
Total Cost: Rp 6370000000
```

In [177…

```
cost_false_positive = 10_000_000  # Rp 10.000.000
cost_false_negative = 30_000_000  # Rp 30.000.000

# Calculate total cost
cfp_base = (cf_base[0, 1] * cost_false_positive)
cfn_base = (cf_base[1, 0] * cost_false_negative)
total_cost_base = cfp_base + cfn_base

print("Confusion Matrix:")
print(cf_base)
print(f"FP Cost    : Rp {cfp_base}")
print(f"FN Cost    : Rp {cfn_base}")
print(f"Total Cost: Rp {total_cost_base}")
```

```
Confusion Matrix:
[[20266     0]
 [ 5658     0]]
FP Cost    : Rp 0
FN Cost    : Rp 169740000000
Total Cost: Rp 169740000000
```

In [178…

```
total_diff = (total_cost_base - total_cost_model) / total_cost_base

print(f"Saving percentage: {total_diff * 100:.2f} %")
```

```
Saving percentage: 96.25 %
```