

Soal Mentoring Meet 2 Machine Learning Process

Job Preparation Program, Data Science, Pacmann AI

Introduction

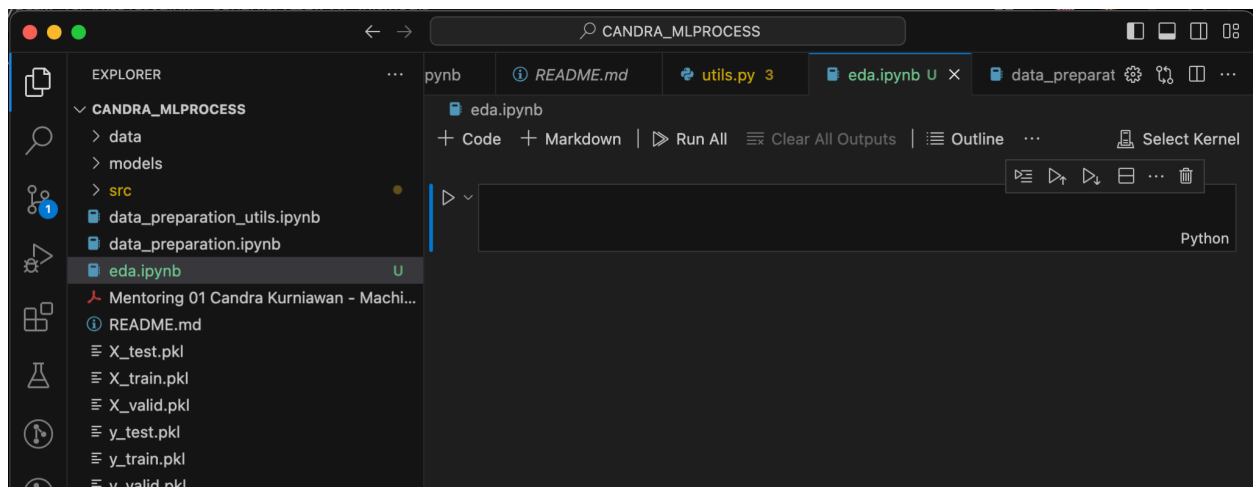
Kelanjutan dari Week 1, dengan menggunakan dataset yang sama, Anda diminta untuk melanjutkan EDA, preprocessing dan Feature Engineering.

Catatan:

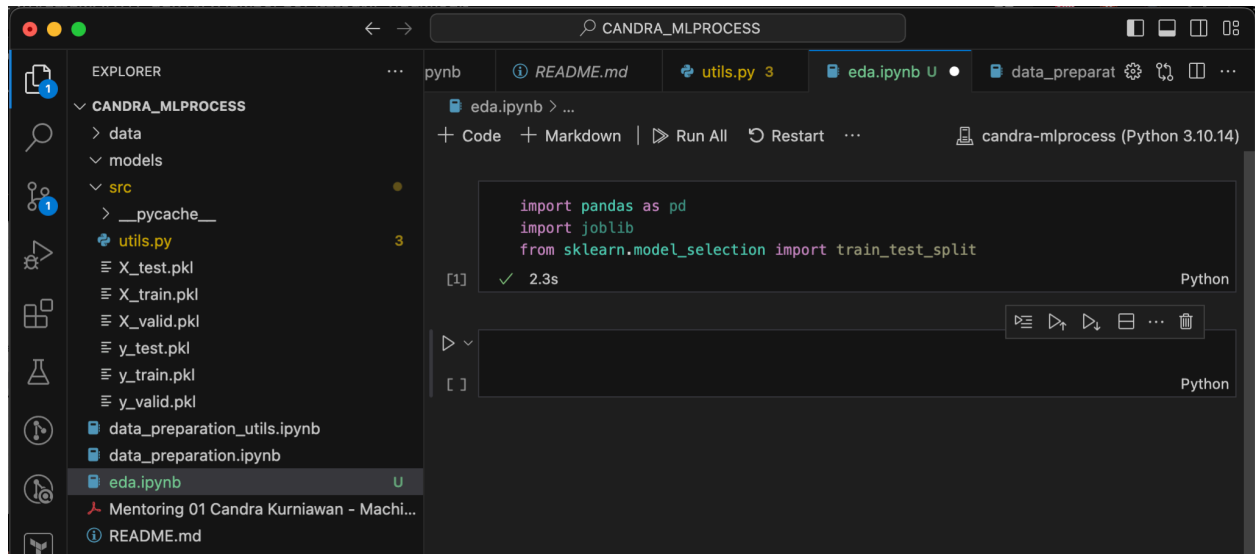
- Tulis jawaban Anda dalam jupyter notebook
- Copy file docs ini untuk menjawab dan sertakan screenshotnya
- Ekspor docs yang telah ada jawabannya ke PDF
- Archive file notebook dan file PDF Anda ke ZIP, **kecuali folder venv**
- Beri nama `[NAMA_LENGKAP]_MLPROCESS_2` file archive Anda

Soal

1. **[50 poin]** EDA
 - a. **[1 poin]** Buatlah satu file bernama `eda.ipynb` di root folder project Anda.
Lampirkan screenshot



- b. **[1 poin]** Import library yang dibutuhkan
Buka `eda.ipynb` dan import library `utils` dari folder `src`, `pandas` yang memiliki alias `pd` dan `seaborn` yang memiliki alias `sns`
Lampirkan screenshot

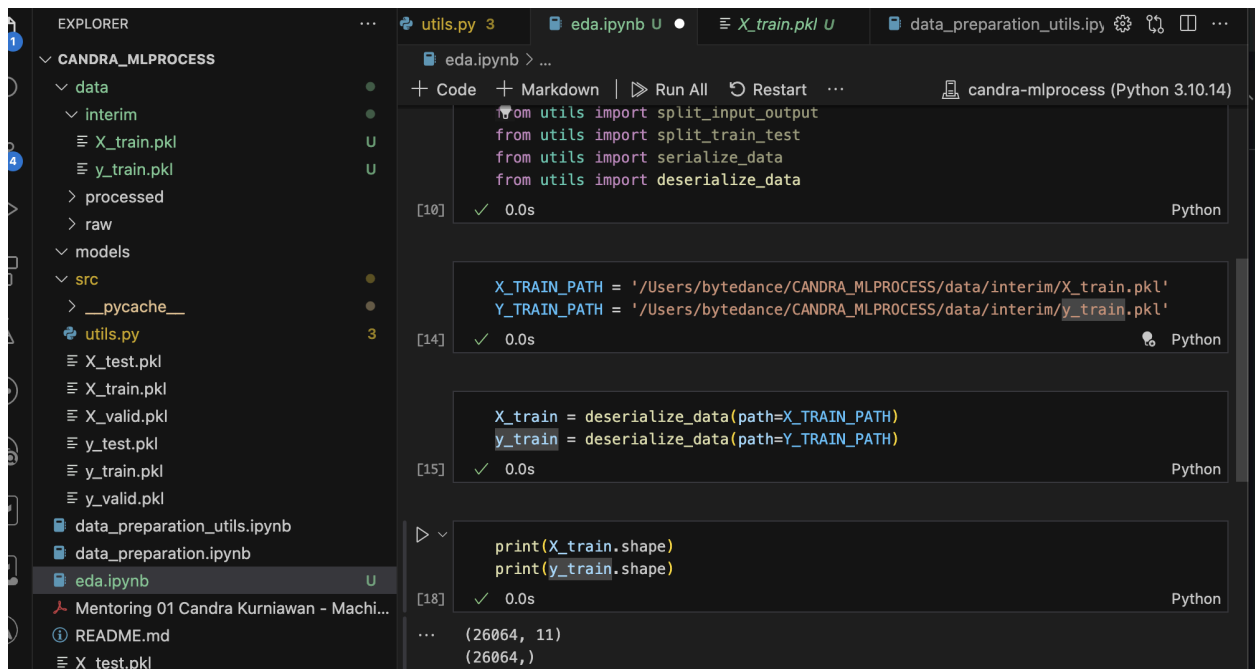


c. [1 poin] Muat train set data

i. Muat X_train data

- Siapkan variabel konstan bernama X_TRAIN_PATH
- Isi X_TRAIN_PATH dengan "data/interim/X_train.pkl", TANPA PETIK!
- Panggil fungsi deserialize_data() dan berikan X_TRAIN_PATH sebagai argumen pertama
- Simpan keluaran dari fungsi deserialize_data() ke variabel bernama X_train

Lampirkan screenshot



ii. Muat y_train data

- Siapkan variabel konstan bernama Y_TRAIN_PATH
- Isi Y_TRAIN_PATH dengan "data/interim/y_train.pkl", TANPA PETIK!
- Panggil fungsi deserialize_data() dan berikan Y_TRAIN_PATH sebagai argumen pertama
- Simpan keluaran dari fungsi deserialize_data() ke variabel bernama y_train

Lampirkan screenshot

```
utils.py 3 | eda.ipynb U | X_train.pkl U | data_preparation_utils.ipynb

CANDRA_MLPROCESS
├── data
│   ├── interim
│   │   ├── X_train.pkl
│   │   └── y_train.pkl
│   ├── processed
│   └── raw
├── models
├── src
│   ├── __pycache__
│   ├── utils.py
│   ├── X_test.pkl
│   ├── X_train.pkl
│   ├── X_valid.pkl
│   ├── y_test.pkl
│   ├── y_train.pkl
│   └── y_valid.pkl
├── data_preparation_utils.ipynb
├── data_preparation.ipynb
├── eda.ipynb U
├── Mentoring 01 Candra Kurniawan - Machi...
├── README.md
└── X_test.pkl

eda.ipynb > ...
+ Code + Markdown | ▶ Run All ↺ Restart ... | candra-mlprocess (Python 3.10.14)

[10] ✓ 0.0s Python
from utils import split_input_output
from utils import split_train_test
from utils import serialize_data
from utils import deserialize_data

[14] ✓ 0.0s Python
X_TRAIN_PATH = '/Users/bytedance/CANDRA_MLPROCESS/data/interim/X_train.pkl'
Y_TRAIN_PATH = '/Users/bytedance/CANDRA_MLPROCESS/data/interim/y_train.pkl'

[15] ✓ 0.0s Python
X_train = deserialize_data(path=X_TRAIN_PATH)
y_train = deserialize_data(path=Y_TRAIN_PATH)

[18] ✓ 0.0s Python
print(X_train.shape)
print(y_train.shape)

... (26064, 11)
(26064,)
```

d. [7 poin] Identifikasi kolom kategorik dan numerik

i. Panggil fungsi head() dari variabel X_train

Lampirkan screenshot

ii. Secara visual, catat nama kolom yang seharusnya bertipe data

- integer dan float ke variabel baru bernama num_col
person_age , person_income, person_emp_length, loan_amnt,
loan_int_rate, loan_percent_income, cb_person_cred_hist_length
- object ke variabel baru bernama cat_col
person_home_ownership, loan_intent, loan_grade,
cb_person_default_on_file

Lampirkan screenshot

X_train.head()

[19] ✓ 0.0s Python

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_percent_income
15884	25	241875	MORTGAGE	4.0	EDUCATION	A	16000	7.05	
15138	21	18000	RENT	5.0	PERSONAL	B	1500	12.18	
7474	25	53000	MORTGAGE	10.0	MEDICAL	B	16000	12.53	
18212	28	16800	OWN	NaN	MEDICAL	C	5000	13.98	
6493	25	50000	MORTGAGE	2.0	VENTURE	A	10000	7.90	

iii. Panggil fungsi info() dari variabel X_train

Lampirkan screenshot

X_train.info()

[20] ✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
Index: 26064 entries, 15884 to 17068
Data columns (total 11 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                -
0   person_age                           26064 non-null  int64
1   person_income                        26064 non-null  int64
2   person_home_ownership                26064 non-null  object
3   person_emp_length                    25326 non-null  float64
4   loan_intent                          26064 non-null  object
5   loan_grade                          26064 non-null  object
6   loan_amnt                           26064 non-null  int64
7   loan_int_rate                        23563 non-null  float64
8   loan_percent_income                  26064 non-null  float64
9   cb_person_default_on_file            26064 non-null  object
10  cb_person_cred_hist_length           26064 non-null  int64
dtypes: float64(3), int64(4), object(4)
memory usage: 2.4+ MB
```

iv. Cocokkan catatan yang telah dibuat dengan keluaran fungsi info()

Sudah dicocokkan dan betul

v. Buatlah kesimpulan dari yang telah dilakukan serta tindakan kedepannya pada sel jupyter notebook selanjutnya

Lampirkan screenshot

```
**Kesimpulan dari dataset X_train:**
1. Terdapat 7 Numerical column/features
-> person_age , person_income, person_emp_length, loan_amnt, loan_int_rate, loan_percent_income, cb_person_cred_hist_length
2. Terdapat 4 Categorical column/features
-> person_home_ownership, loan_intent, loan_grade, cb_person_default_on_file
3. Terdapat beberapa data yang berbentuk Numerical dan Categorical data, dilakukan pemisahan yang berguna untuk dilakukannya data preprocessing lebih lanjut sehingga perlunya dilakukan pemisahan.
```

e. [10 poin] Lakukan pengecekan terhadap data yang duplikat

i. Lakukan pengecekan data duplikat HANYA pada X_train

ii. Lakukan pengecekan dengan fungsi duplicated

iii. Berikan nilai False pada parameter keep di fungsi duplicated

- iv. Urutkan data berdasarkan data pada kolom person income
Lampirkan screenshot

```
# Dataframe X_train sekarang tersorted dengan kolom 'person_income' terlihat dari perbedaan dataframe X_train diatas.  
# Ketika setelah dilakukan pengecekan dengan fungsi diatas  
# terbuat 1 kolom baru 'is_duplicated' yang menyatakan apakah row tsb duplikat  
  
X_train_sorted.head()
```

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_percent_income	cb_pe
31930	41	4000	RENT	0.0	MEDICAL	C	2000	13.22	0.50	
15964	21	4080	RENT	0.0	EDUCATION	B	1400	11.86	0.34	
15963	21	4200	RENT	3.0	PERSONAL	E	2750	16.95	0.65	
27896	32	4200	RENT	NaN	PERSONAL	C	1200	12.73	0.29	
15960	22	4800	RENT	1.0	PERSONAL	D	1800	14.84	0.38	

- v. Buatlah kesimpulan dari yang telah dilakukan serta tindakan kedepannya pada sel jupyter notebook selanjutnya
Lampirkan screenshot

```
# Dataframe X_train sekarang tersorted dengan kolom 'person_income' terlihat dari perbedaan dataframe X_train diatas.  
# Ketika setelah dilakukan pengecekan dengan fungsi diatas  
# terbuat 1 kolom baru 'is_duplicated' yang menyatakan apakah row tsb duplikat  
  
X_train_sorted.head()
```

```
# Setelah dilakukan pengecekan duplicate, karena parameter di set 'False'  
# Maka data duplicate masih tetap ada  
# terbuat 1 kolom (series). yang menyatakan bahwa masing-masing row apakah duplikat atau tidak  
# Data duplikat ini kedepannya dapat dilakukan pemisahan ataupun bisa dihapus sesuai dengan permintaan Business Owner/Business Analyst nya.
```

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	is_duplicate
41	4000	RENT	0.0	MEDICAL	C	2000	13.22	0.50	Y	15	False
21	4080	RENT	0.0	EDUCATION	B	1400	11.86	0.34	N	4	False
21	4200	RENT	3.0	PERSONAL	E	2750	16.95	0.65	N	3	False
32	4200	RENT	NaN	PERSONAL	C	1200	12.73	0.29	N	9	False
22	4800	RENT	1.0	PERSONAL	D	1800	14.84	0.38	Y	2	False

- f. [10 poin] Lakukan pengecekan terhadap null value
- Lakukan pengecekan pada kolom yang memiliki baris dengan nilai null pada X_train
 - Gunakan fungsi isnull() dan chain dengan fungsi sum()

Lampirkan screenshot

Check for Null Values

```
def check_null_values(data):  
    """  
    Fungsi untuk melakukan pengecekan terhadap nilai null pada kolom di X_train.  
  
    Parameters:  
    X_train (pd.DataFrame): DataFrame yang akan dicek nilai nullnya.  
  
    Returns:  
    pd.Series: Series yang menunjukkan jumlah nilai null per kolom.  
    """  
    # Mengecek nilai null dan menjumlahkan per kolom  
    null_counts = data.isnull().sum()  
  
    # Menyaring hanya kolom yang memiliki nilai null  
    null_counts = null_counts[null_counts > 0]  
  
    return null_counts
```

✓ 0.0s

- iii. Buatlah kesimpulan dari yang telah dilakukan serta tindakan kedepannya pada sel jupyter notebook selanjutnya

Lampirkan screenshot

```
# Melakukan proses data kedalam Function 'check_null_values' untuk check Null Values  
# dan Print counts Null values nya  
X_train_null_counts = check_null_values(data = X_train_sorted)  
print(X_train_null_counts)  
  
# Null values yang terdeteksi ini kedepannya dapat dilakukan imputing atau delete data  
# Tergantung dari kebutuhan bisnis, atau request dari Business Owner/Analyst.
```

[43] ✓ 0.0s

```
... person_emp_length    738  
loan_int_rate         2501  
dtype: int64
```

- g. [10 poin] Lakukan pengecekan distribusi pada tiap input
- Buat temporary variabel bernama X_train_
 -
 - Seleksi hanya kolom numerik pada variabel X_train dan simpan pada variabel X_train_ menggunakan daftar nama kolom yang telah dibuat sebelumnya (variabel num_col)
Lampirkan screenshot
 - Buatlah perulangan dari tiap nama kolom pada variabel X_train_
 - Simpan nama kolom pada tiap iterasi perulangan ke variabel bernama col
 - Pada tiap iterasi perulangan, panggil fungsi displot() dari library seaborn
 - Data yang digunakan adalah X_train_
 - X axis menampilkan data kolom saat ini
 - Gunakan bins 20

Lampirkan screenshot

Pengecekan Distribusi pada tiap Input

```
import seaborn as sns
Click to add a breakpoint pyplot as plt

def check_distribution(data):
    """
    Fungsi untuk melakukan pengecekan distribusi pada tiap input numerik di data.

    Parameters:
    data (pd.DataFrame): DataFrame yang akan dicek distribusi kolom-kolom numeriknya.

    Returns:
    None: Fungsi ini akan menampilkan plot distribusi untuk setiap kolom numerik.

    Candra | Pacmann AI 2024.
    """
    # Seleksi hanya kolom numerik
    X_train_ = data.select_dtypes(include=['number']).columns.tolist()
    num_col = data[X_train_]

    # Perulangan untuk setiap kolom numerik
    for col in num_col:
        # Memanggil fungsi displot dari seaborn
        sns.displot(num_col[col], bins=20, kde=True)
        plt.xlabel(col)
        plt.title(f'Distribution of {col}')
        plt.show()
```

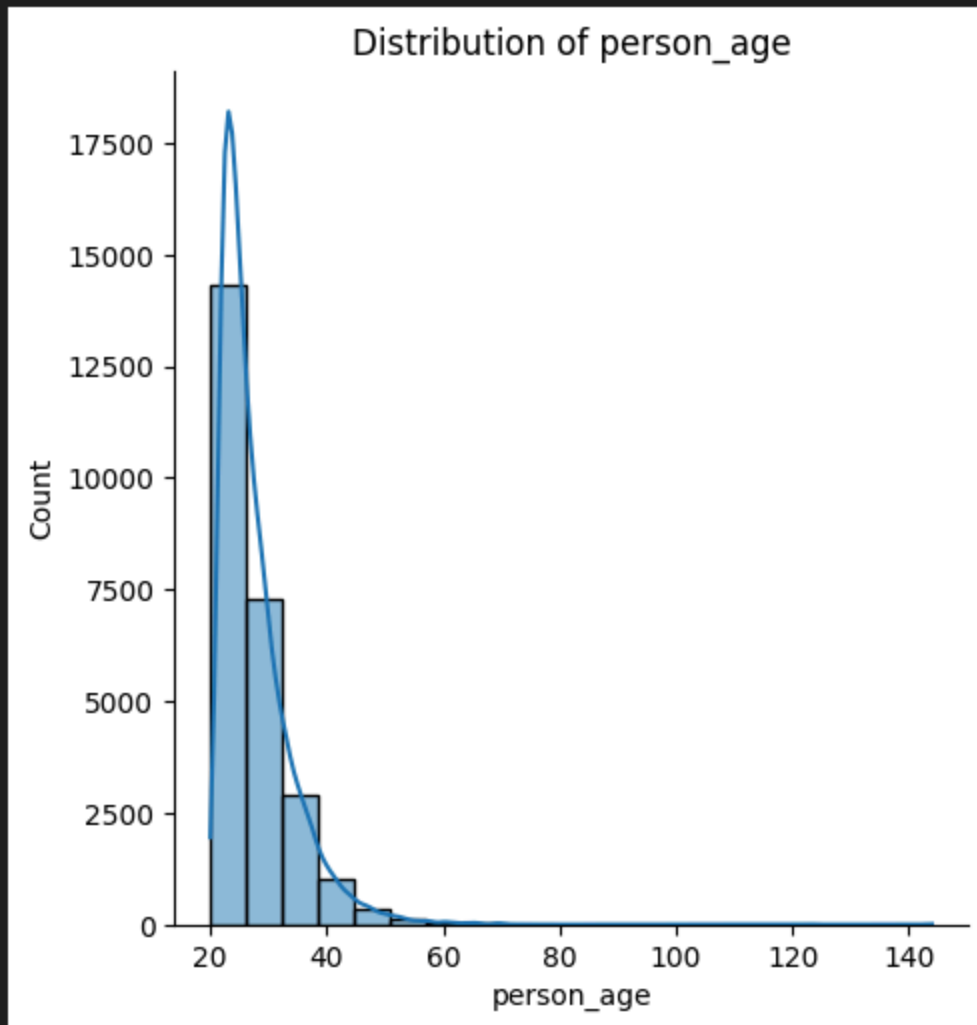
[82] ✓ 0.0s Python

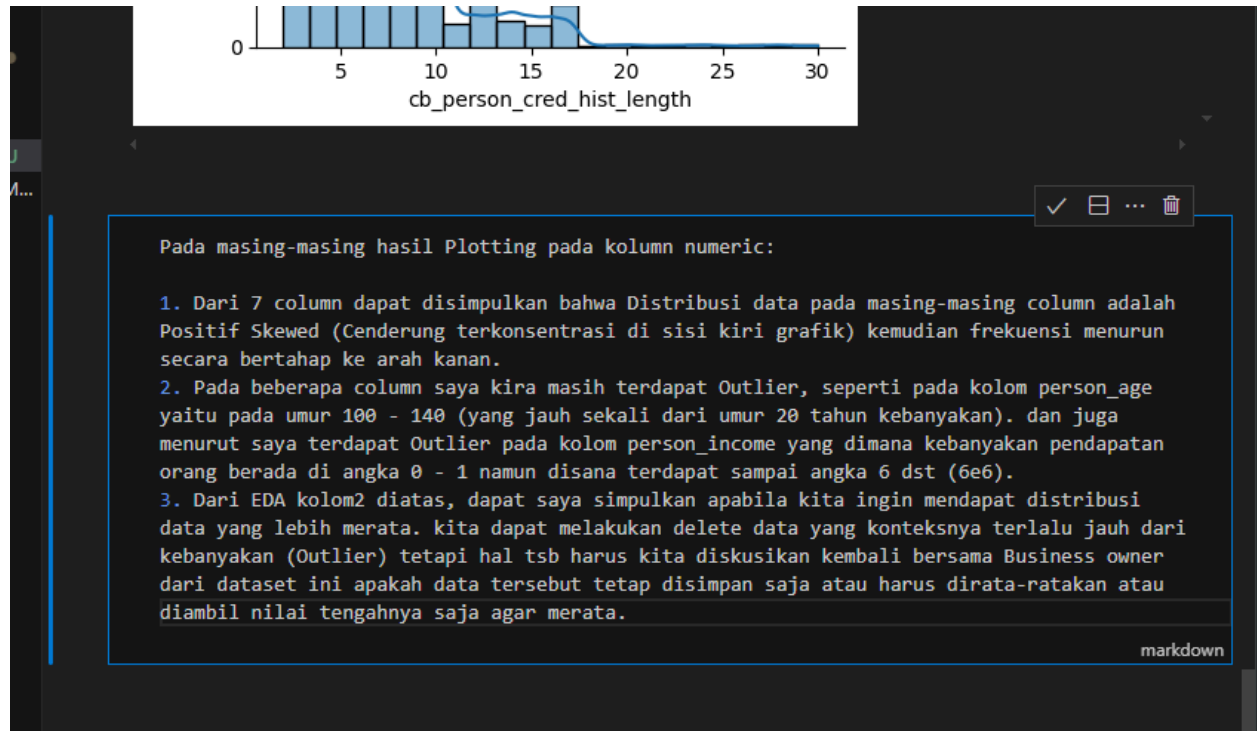
- vii. Buatlah kesimpulan dari yang telah dilakukan serta tindakan kedepannya pada sel jupyter notebook selanjutnya
Semisal: “dari distribusi ini, saya bisa melakukan xxx untuk tahap preprocessing”

Lampirkan screenshot

```
# Melakukan Check Distribusi data pada column Numeric  
check_distribution(data=X_train)
```

[83] ✓ 1.8s





- h. [4 poin] Lakukan pengecekan terhadap target balance
- i. Panggil fungsi `countplot()` dari library `seaborn`
- Data yang digunakan adalah `y_train` yang telah diubah ke `dataframe`
 - X axis menampilkan kolom `target`
 - Set `hue` ke kolom `target`
- Lampirkan screenshot**

```
def check_target_balance(y_train, target_column_name='target'):
    """
    Function to check the balance of the target variable in y_train using a count plot.

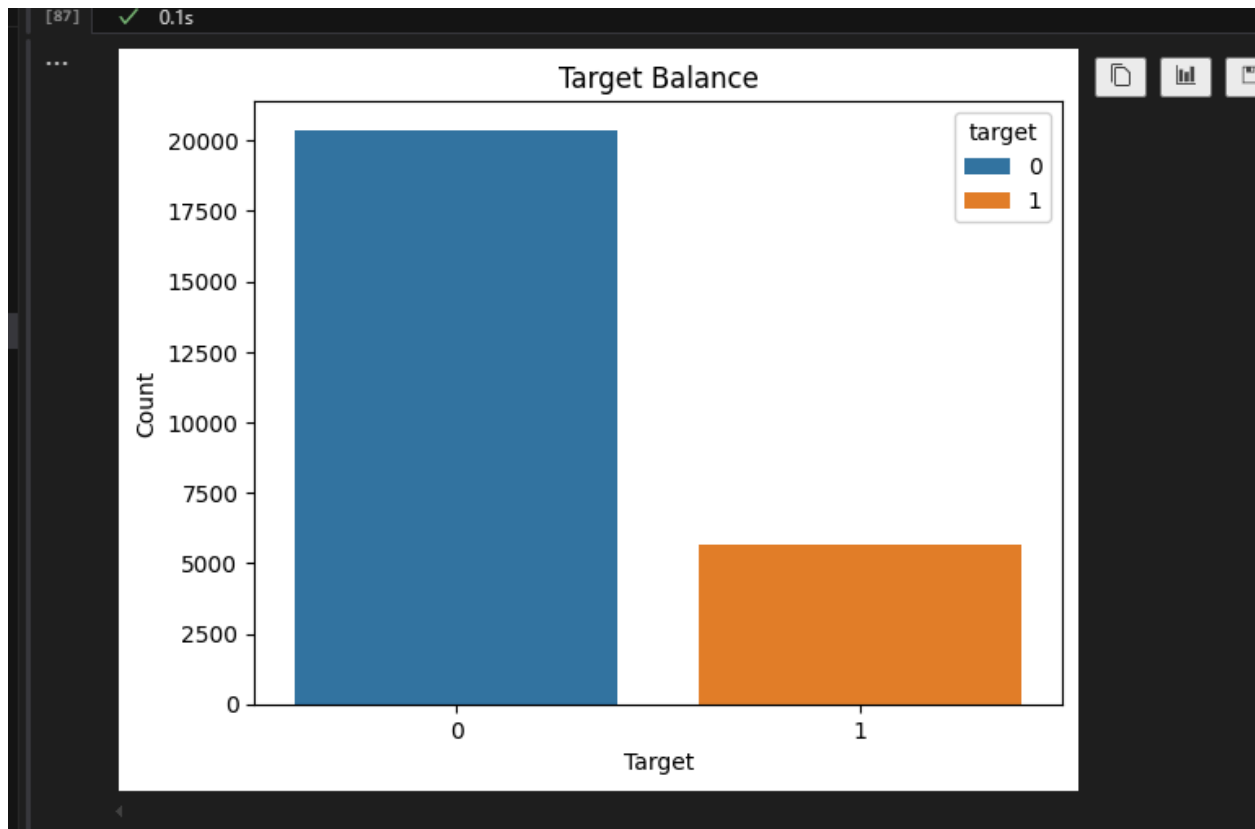
    Parameters:
    y_train (pd.Series or pd.DataFrame): Target variable data.
    target_column_name (str): Name of the target column if y_train is a DataFrame.

    Returns:
    None: This function will display a count plot of the target variable.
    """
    # Convert y_train to DataFrame if it is not already
    if isinstance(y_train, pd.Series):
        y_train = y_train.to_frame(name=target_column_name)

    # Create count plot
    sns.countplot(data=y_train, x=target_column_name, hue=target_column_name)
    plt.xlabel('Target')
    plt.ylabel('Count')
    plt.title('Target Balance')
    plt.show()
```

[86] ✓ 0.0s Python

- ii. Buatlah kesimpulan dari yang telah dilakukan serta tindakan kedepannya pada sel jupyter notebook selanjutnya
Lampirkan screenshot



✓ ☰ ... 🗑️

Kesimpulan dari EDA `y_train (loan_status)`:

1. terdapat 2 jenis pada kolom `loan_status` ini yaitu 1 = Meminjam dan 0 = Belum Meminjam.
2. Distribusi Belum meminjam (0) lebih banyak dibandingkan Peminjam.
3. Belum meminjam terdapat 20000 orang, sedangkan Peminjam terdapat 5000 orang.
4. untuk kedepannya mungkin bisa ditingkatkan lagi Campaign perusahaan sehingga peminjam semakin banyak.

🗨️ markdown

- i. [6 poin] Rangkum semua kesimpulan yang telah dibuat selama EDA pada cell selanjutnya

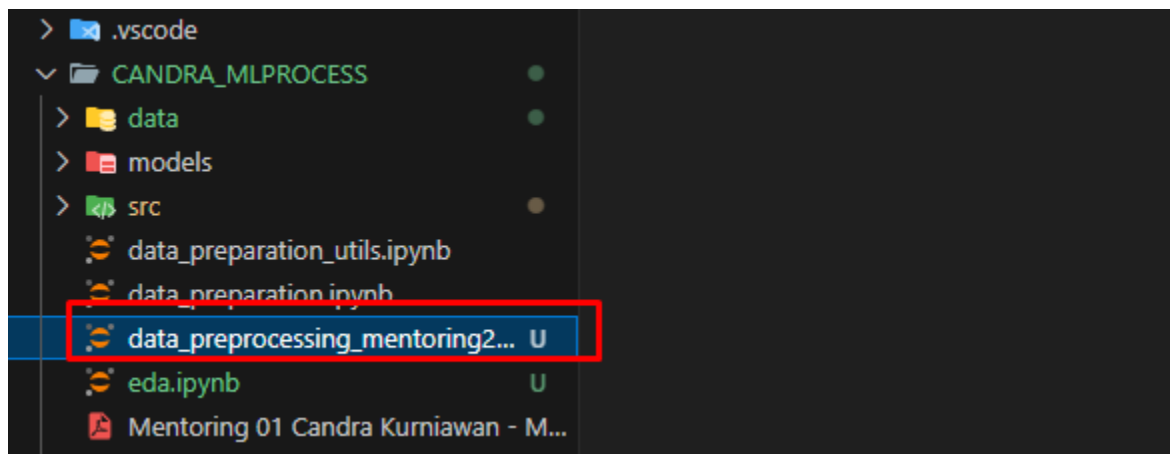
Kesimpulan dari EDA X_train dan y_train:

1. Pengecekan Duplicate data dilakukan agar dapat mencari tahu ada berapa banyak data/row yang duplikat yang mungkin terjadi karena human error saat melakukan input data. dan selanjutnya dapat di diskusikan kembali dengan business owner/analisis apakah data dup tsb lebih baik dihilangkan atau tetap dimasukkan kedalam train model.
2. Pengecekan pada Null values dilakukan untuk mengetahui ada berapa banyak sel yang mengalami kekosongan data. pada kasus ini lebih baik data yang kosong tersebut agar diisikan dengan dummy data menggunakan teknik mean, median, atau modus lalu di imput kan.
3. Pengecekan Distribusi data pada masing-masing kolom x_train dilakukan secara visualisasi agar kita dapat lebih mengetahui sebaran data nya mengalami skewed atau tidak. apabila skewed terlalu jauh. maka dapat didiskusikan kembali agar melakukan penyempitan data dengan men delete data-data outlier.

2. [40 poin] Data Preprocessing

- a. [1 poin] Buat satu file bernama data_preprocessing.ipynb di root project folder Anda

Lampirkan screenshot



- b. [1 poin] Import library yang dibutuhkan

Buka file data_preprocessing.ipynb dan import utils dari folder src, deepcopy dari library copy, OneHotEncoder dari sklearn.preprocessing, numpy dengan alias np dan pandas dengan alias pd

Lampirkan screenshot

```
CANDRA_MLPROCESS > data_preprocessing_mentoring2.ipynb > ...
+ Code + Markdown | ▶ Run All ⏮ Restart ⏭ Clear All Outputs | 📄 Variables 📄 Outline ... mlproces-pacmann (Python undefined.undefined.undefined)

# Change directory to the desired path
%cd D:\Documents\ai_engineer_pacmann\ai_engineer_pacmann\CANDRA_MLPROCESS\src

import utils

from utils import load_data
from utils import split_input_output
from utils import split_train_test
from utils import serialize_data
from utils import deserialize_data

from copy import deepcopy
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import pandas as pd

[2] ✓ 0.0s Python
D:\Documents\ai_engineer_pacmann\ai_engineer_pacmann\CANDRA_MLPROCESS\src
c:\Users\lolcandra\miniconda3\envs\mlproces-pacmann\lib\site-packages\IPython\core\magics\osm.py:417: UserWarning: This is now an optiona
self.shell.db['dhist'] = compress_dhist(dhist)[-100:]
```

c. [1 poin] Muat splitted dataset

i. Panggil fungsi deserialize_data() dari utils

- Berikan argumen berupa lokasi dimana data X dan y untuk train, test, dan validation Anda berada
- Simpan keluaran fungsi pada variabel sesuai dengan set datanya

Lampirkan screenshot

```
# Alamat file X dan y untuk di deserialize

X_TRAIN_PATH = '..\data\interim\X_train.pkl'
Y_TRAIN_PATH = '..\data\interim\y_train.pkl'
X_TEST_PATH = '..\data\interim\X_test.pkl'
Y_TEST_PATH = '..\data\interim\y_test.pkl'
X_VAL_PATH = '..\data\interim\X_valid.pkl'
Y_VAL_PATH = '..\data\interim\y_valid.pkl'

[3] ✓ 0.0s Python

# Melakukan deserialize data agar dapat digunakan didalam variable

X_train = deserialize_data(path=X_TRAIN_PATH)
y_train = deserialize_data(path=Y_TRAIN_PATH)
X_test = deserialize_data(path=X_TEST_PATH)
y_test = deserialize_data(path=Y_TEST_PATH)
X_valid = deserialize_data(path=X_VAL_PATH)
y_valid = deserialize_data(path=Y_VAL_PATH)

[6] ✓ 0.0s Python
```

d. [13 poin] Buang data duplikat

i. Buat fungsi untuk drop duplicate

- Buat fungsi bernama drop_duplicate_data()
- Fungsi tersebut memiliki parameter bernama X dan y
- Parameter X
 - a. Harus bertipe dataframe

- b. Merupakan set data (train, test, ataupun valid) yang ingin dibuang data duplicatenya
- Parameter y
 - a. Harus bertipe series
 - b. Merupakan data target dari data pada parameter X
- Buat 3 percabangan untuk validasi parameter

Buat 3 percabangan untuk

 - a. Percabangan pertama: melakukan komparasi antara parameter X dan dataframe
 - b. Percabangan kedua: melakukan komparasi antara parameter y dan series
 - c. Percabangan ketiga: print pesan “Fungsi drop_duplicate_data: parameter telah divalidasi.”
- Lakukan copy pada parameter X dan y dengan menggunakan fungsi copy() dari pandas dan simpan pada variabel yang sama
- Print pesan tentang shape dari data pada parameter X
 - a. Pesannya adalah “Fungsi drop_duplicate_data: shape dataset sebelum dropping duplicate adalah [shape dari data pada parameter X].”
 - b. Gunakan macro format (f) pada string yang akan diprint menggunakan fungsi print
 - c. Akses atribut shape pada data X untuk mendapatkan shape data tersebut
- Pengecekan duplicate
 - a. Buat satu variabel baru bernama X_duplicate
 - b. Variabel tersebut diisi oleh dataframe baru hasil seleksi dari data X yang duplicate
 - i. Gunakan teknik seleksi dataframe dengan menggunakan fungsi duplicated()
 - ii. Ambil hanya baris yang memiliki nilai kembalian dari fungsi duplicated() True
- Print pesan tentang shape dari data yang duplikat
 - a. Pesannya adalah “Fungsi drop_duplicate_data: shape dari data yang duplicate adalah [shape dari data yang duplikat].”
 - b. Gunakan macro format (f) pada string yang akan diprint menggunakan fungsi print
 - c. Akses attribute shape dari data X_duplicate untuk mendapatkan shape data tersebut
- Kalkulasi shape setelah drop duplicate
 - a. Buat satu variabel bernama X_clean
 - b. Variabel X_clean bertipe tuple

- c. Untuk elemen pertama, isi dengan selisih baris antara data X yang belum didrop duplicate dan data X_duplicate yang merupakan data duplicate dari X
 - d. Untuk elemen kedua, isi dengan nilai kolom dari data X
- Print pesan tentang hasil prediksi kalkulasi shape dari data X setelah didrop duplicate
 - a. Pesannya adalah "Fungsi drop_duplicate_data: shape dataset setelah drop duplicate seharusnya adalah [prediksi hasil shape]."
 - b. Gunakan macro format (f) pada string yang akan diprint menggunakan fungsi print
 - c. Gunakan X_clean sebagai hasil prediksi shape yang akan ditampilkan dari data X
- Lakukan drop duplicate pada data X dengan menggunakan fungsi drop_duplicates() dari pandas
 - a. Set parameter inplace menjadi True
- Lakukan seleksi pada data y dengan menggunakan index dari data X yang telah didrop duplicate
- Print pesan tentang shape dari data X setelah dilakukan dropping duplicate
 - a. Pesannya adalah "Fungsi drop_duplicate_data: shape dataset setelah dropping duplicate adalah [shape dari data X yang telah didrop duplicate]."
 - b. Gunakan macro format (f) pada string yang akan diprint menggunakan fungsi print
 - c. Akses atribut shape pada data X untuk mendapatkan nilai shape
- Kembalikan X dan y
- Buat docstring yang menjelaskan tujuan dari fungsi tersebut serta parameternya

Lampirkan screenshot

```
def drop_duplicate_data(X, y):
    """
    Fungsi untuk membuang data duplikat dari dataset.

    Parameters:
    X : pd.DataFrame
        | DataFrame yang berisi data fitur.
    y : pd.Series
        | Series yang berisi data target yang sesuai dengan data fitur pada X.

    Returns:
    X : pd.DataFrame
        | DataFrame yang telah dibuang data duplikatnya.
    y : pd.Series
        | Series yang telah disesuaikan dengan data fitur setelah membuang data duplikat.
    """

    # Validasi parameter X
    if not isinstance(X, pd.DataFrame):
        raise ValueError("Parameter X harus bertipe DataFrame.")

    # Validasi parameter y
    if not isinstance(y, pd.Series):
        raise ValueError("Parameter y harus bertipe Series.")

    print("Fungsi drop_duplicate_data: parameter telah divalidasi.")

    # Copy data
    X = X.copy()
    y = y.copy()

    # Print shape sebelum dropping duplicate
    print(f"Fungsi drop_duplicate_data: shape dataset sebelum dropping duplicate adalah {X.shape}.")

    # Pengecekan duplicate
    X_duplicate = X[X.duplicated(keep=False)]
    print(f"Fungsi drop_duplicate_data: shape dari data yang duplicate adalah {X_duplicate.shape}.")

    # Kalkulasi shape setelah drop duplicate
    X_clean = (X.shape[0] - X_duplicate.shape[0], X.shape[1])
    print(f"Fungsi drop_duplicate_data: shape dataset setelah drop duplicate seharusnya adalah {X_clean}.")

    # Drop duplicate
    X.drop_duplicates(inplace=True)

    # Seleksi data y dengan index dari data X yang telah didrop duplicate
    y = y[X.index]

    # Print shape setelah dropping duplicate
    print(f"Fungsi drop_duplicate_data: shape dataset setelah dropping duplicate adalah {X.shape}.")

    return X, y
```

[15] ✓ 0.0s

ii. Jalankan fungsi

- Panggil fungsi `drop_duplicate_data()`
- Berikan data `X_train` dan `y_train` sebagai argumen untuk parameter `X` dan `y` pada fungsi tersebut

- Simpan keluaran fungsi tersebut ke variabel X_train dan y_train (direplace)

Lampirkan screenshot

```
# Memanggil fungsi untuk membuang data duplikat pada dataset X_train y_train
X_train, y_train = drop_duplicate_data(X=X_train, y=y_train)
```

[16] ✓ 0.0s

... Fungsi drop_duplicate_data: parameter telah divalidasi.
 Fungsi drop_duplicate_data: shape dataset sebelum dropping duplicate adalah (26064, 11).
 Fungsi drop_duplicate_data: shape dari data yang duplicate adalah (192, 11).
 Fungsi drop_duplicate_data: shape dataset setelah drop duplicate seharusnya adalah (25872, 11).
 Fungsi drop_duplicate_data: shape dataset setelah dropping duplicate adalah (25968, 11).

e. [13 poin] Imputasi data numerik

i. Buat fungsi untuk imputasi data numerik

- Buat fungsi bernama median_imputation()
- Fungsi tersebut memiliki parameter bernama data, subset_data, dan fit
- Parameter data harus bertipe dataframe
- Parameter data merupakan set data (train, test, ataupun valid) yang ingin diimputasi datanya
- Parameter fit harus bertipe data boolean
- Jika parameter fit bernilai True maka fungsi akan melakukan kalkulasi nilai median dari tiap nama kolom yang telah dispesifikkan pada parameter subset_data
- Jika parameter fit bernilai False maka fungsi akan melakukan imputasi pada data berdasarkan kalkulasi sebelumnya (fit = True)
- Untuk melakukan imputasi, pengguna diharuskan untuk melakukan fit terlebih dahulu
- Parameter subset_data untuk fit = True bertipe list 1 dimensi
 - a. Parameter subset_data bertipe list merupakan daftar nama kolom dari data yang ingin diimputasi
- Parameter subset_data untuk fit = False bertipe dict
 - a. Parameter subset_data bertipe dict merupakan daftar nama kolom sebagai key beserta nilai mediannya sebagai value
- Buat percabangan untuk validasi parameter
 - a. Percabangan pertama
 - i. Melakukan komparasi antara parameter data dan dataframe
 - ii. Gunakan fungsi isinstance() untuk melakukan komparasi
 - iii. Jika parameter data tidak sama dengan dataframe, maka raise runtimeerror dengan pesan "Fungsi

median_imputation: parameter data haruslah bertipe DataFrame!"

- iv. Jika parameter data sama dengan dataframe, maka eksekusi kode diluar cakupan percabangan ini
- b. Percabangan kedua
 - i. Melakukan komparasi antara parameter fit dan True
 - ii. Gunakan operator equal untuk melakukan komparasi ini
 - iii. Jika parameter fit sama dengan True:
 - 1. Buat anak cabang pertama
 - a. Melakukan komparasi antara parameter subset_data dengan list
 - b. Gunakan fungsi instance untuk melakukan komparasi
 - c. Jika parameter subset_data tidak sama dengan list, maka raise runtimeerror dengan pesan "Fungsi median_imputation: untuk nilai parameter fit = True, subset_data harus bertipe list dan berisi daftar nama kolom yang ingin dicari nilai mediannya guna menjadi data imputasi pada kolom tersebut."
 - d. Jika parameter subset_data sama dengan list, maka eksekusi kode diluar cakupan percabangan ini
 - iv. Jika parameter fit tidak sama dengan True, maka eksekusi percabangan ketiga
- c. Percabangan ketiga
 - i. Melakukan komparasi antara parameter fit dan False
 - ii. Gunakan operator equal untuk melakukan komparasi ini
 - iii. Jika parameter fit sama dengan False:
 - 1. Buat anak cabang pertama
 - a. Melakukan komparasi antara parameter subset_data dengan dict
 - b. Gunakan fungsi instance untuk melakukan komparasi
 - c. Jika parameter subset_data tidak sama dengan dict, maka raise runtimeerror dengan pesan "Fungsi

median_imputation: untuk nilai parameter fit = False, subset_data harus bertipe dict dan berisi key yang merupakan nama kolom beserta value yang merupakan nilai median dari kolom tersebut.”

- d. Jika parameter subset_data sama dengan dict, maka eksekusi kode diluar cakupan percabangan ini
- iv. Jika parameter fit tidak sama dengan False, maka eksekusi percabangan keempat
- d. Percabangan keempat
 - i. Raise runtimeerror dengan pesan “Fungsi median_imputation: parameter fit haruslah bertipe boolean, bernilai True atau False.”
- Print pesan bahwa parameter telah divalidasi
 - a. Pesannya adalah “Fungsi median_imputation: parameter telah divalidasi.”
- Lakukan copy pada parameter data
 - a. Gunakan fungsi copy() dari pandas
 - b. Simpan pada variabel baru bernama data (direplace)
- Lakukan copy pada parameter subset_data
 - a. Gunakan fungsi deep_copy() dari library copy
 - b. Simpan pada variabel baru bernama subset_data (direplace)
- Buat percabangan untuk parameter fit bernilai True, False, dan selain keduanya
 - a. Pada percabangan dengan parameter fit bernilai True:
 - i. Buat satu variabel bernama imputation_data dan bertipe dict
 - ii. Buat perulangan dari subset_data yang akan memberikan tiap elemennya
 - iii. Simpan elemen tersebut pada variabel bernama subset
 - iv. Dalam perulangan tersebut:
 - 1. Seleksi data berdasarkan nama kolom yang terdapat pada variabel subset
 - 2. Kalkulasi nilai mediannya
 - 3. Simpan data nilai median beserta nama kolomnya dalam variabel imputation_data
 - v. Print pesan tentang hasil dari fitting
 - 1. Pesannya adalah “Fungsi median_imputation: proses fitting telah selesai, berikut hasilnya [hasil dari fitting].”

2. Gunakan macro format (f) pada string yang akan diprint menggunakan fungsi print
 3. Gunakan `imputation_data` sebagai hasil dari fitting.
- vi. Kembalikan variabel `imputation_data` dari fungsi
- b. Pada percabangan dengan parameter `fit` bernilai `False`:
- i. Print pesan tentang kondisi dari tiap kolom sebelum dilakukan imputasi
 1. Pesan pertama yang diprint adalah "Fungsi `median_imputation`: informasi count na sebelum dilakukan imputasi:"
 2. Pesan kedua yang diprint adalah keluaran dari chaining fungsi `isna()` dan `sum()` pada parameter data
 3. Pesan ketiga adalah string kosong, hal ini sebagai new line
 - ii. Lakukan imputasi dengan memanggil fungsi `fillna()`
 1. Gunakan `subset_data` sebagai argumen pada fungsi `fillna()`
 2. Gunakan nilai `True` pada parameter `inplace` di fungsi `fillna()`
 - iii. Print pesan tentang kondisi dari tiap kolom setelah dilakukan imputasi
 1. Pesan pertama yang diprint adalah "Fungsi `median_imputation`: informasi count na setelah dilakukan imputasi:"
 2. Pesan kedua yang diprint adalah keluaran dari chaining fungsi `isna()` dan `sum()` pada parameter data
 3. Pesan ketiga adalah string kosong, hal ini sebagai new line
 - iv. Kembalikan variabel data dari fungsi

Lampirkan screenshot

e. Imputasi Data Numerik

```
#import pandas as pd
#from copy import deepcopy

def median_imputation(data, subset_data, fit):
    """
    Fungsi untuk melakukan imputasi data numerik dengan nilai median.

    Parameters:
    data : pd.DataFrame
        DataFrame yang berisi data yang ingin diimputasi.
    subset_data : list atau dict
        Jika fit = True, subset_data adalah daftar nama kolom yang ingin dicari nilai mediannya.
        Jika fit = False, subset_data adalah dict dengan nama kolom sebagai key dan nilai median sebagai value.
    fit : bool
        Jika True, melakukan kalkulasi nilai median dari tiap kolom yang dispesifikkan pada subset_data.
        Jika False, melakukan imputasi pada data berdasarkan kalkulasi sebelumnya.

    Returns:
    Jika fit = True, mengembalikan dict dengan nama kolom dan nilai median.
    Jika fit = False, mengembalikan DataFrame dengan data yang telah diimputasi.
    """

    # Validasi parameter data
    if not isinstance(data, pd.DataFrame):
        raise RuntimeError("Fungsi median_imputation: parameter data haruslah bertipe DataFrame!")

    # Validasi parameter fit dan subset_data
    if fit == True:
        if not isinstance(subset_data, list):
            raise RuntimeError("Fungsi median_imputation: untuk nilai parameter fit = True, subset_data harus bertipe list dan berisi daftar nama kolom yang ingin dicari nilai")
    elif fit == False:
        if not isinstance(subset_data, dict):
            raise RuntimeError("Fungsi median_imputation: untuk nilai parameter fit = False, subset_data harus bertipe dict dan berisi key yang merupakan nama kolom beserta val")
        else:
            raise RuntimeError("Fungsi median_imputation: parameter fit haruslah bertipe boolean, bernilai True atau False.")

    print("Fungsi median_imputation: parameter telah divalidasi.")
```

```
# Copy data
data = data.copy()
subset_data = deepcopy(subset_data)

# Imputasi data
if fit == True:
    imputation_data = {}
    for subset in subset_data:
        median_value = data[subset].median()
        imputation_data[subset] = median_value
    print(f"Fungsi median_imputation: proses fitting telah selesai, berikut hasilnya {imputation_data}.")
    return imputation_data
else:
    print("Fungsi median_imputation: informasi count na sebelum dilakukan imputasi:")
    print(data.isna().sum())
    print("")
    data.fillna(subset_data, inplace=True)
    print("Fungsi median_imputation: informasi count na setelah dilakukan imputasi:")
    print(data.isna().sum())
    print("")
    return data
```

ii. Jalankan fungsi

- Buat variabel bernama subset_data
- Isi variabel tersebut dengan nama kolom yang hendak diimputasi
- Panggil fungsi median_imputation()
 - a. Gunakan X_train, subset_data, dan True sebagai argumen untuk parameter data, subset_data, dan fit pada fungsi tersebut secara berurutan
- Simpan keluaran fungsi pada variabel subset_data (direplace)
- Panggil ulang fungsi median_imputation()
 - a. Gunakan X_train, subset_data, dan False sebagai argumen untuk parameter parameter data, subset_data, dan fit pada fungsi tersebut secara berurutan
- Simpan keluaran fungsi pada variabel X_train
- Panggil ulang fungsi median_imputation

- a. Gunakan `X_test`, `subset_data`, dan `False` sebagai argumen untuk parameter parameter data, `subset_data`, dan `fit` pada fungsi tersebut secara berurutan
- Simpan keluaran fungsi pada variabel `X_test`
- Panggil ulang fungsi `median_imputation`
 - a. Gunakan `X_valid`, `subset_data`, dan `False` sebagai argumen untuk parameter parameter data, `subset_data`, dan `fit` pada fungsi tersebut secara berurutan
- Simpan keluaran fungsi pada variabel `X_valid`

Lampirkan screenshot

```
# Menentukan subset_data
subset_data = ['person_age', 'person_income', 'person_emp_length', 'loan_amnt', 'loan_int_rate', 'loan_percent_income', 'cb_person_cred_hist_length']

# Panggil fungsi median_imputation() untuk fitting
subset_data = median_imputation(X_train, subset_data, fit=True)
```

[29] ✓ 0.0s Python

... Fungsi median_imputation: parameter telah divalidasi.
Fungsi median_imputation: proses fitting telah selesai, berikut hasilnya {'person_age': np.float64(26.0), 'person_income': np.float64(55000.0), 'person_emp_length': np.float64(4.0)}

```
# Panggil fungsi median_imputation() untuk imputasi pada X_train
X_train = median_imputation(X_train, subset_data, fit=False)
```

[30] ✓ 0.0s

... Fungsi median_imputation: parameter telah divalidasi.
Fungsi median_imputation: informasi count na sebelum dilakukan imputasi:

person_age	0
person_income	0
person_home_ownership	0
person_emp_length	734
loan_intent	0
loan_grade	0
loan_amnt	0
loan_int_rate	2491
loan_percent_income	0
cb_person_default_on_file	0
cb_person_cred_hist_length	0

dtype: int64

Fungsi median_imputation: informasi count na setelah dilakukan imputasi:

person_age	0
person_income	0
person_home_ownership	0
person_emp_length	0
loan_intent	0
loan_grade	0
loan_amnt	0
loan_int_rate	0
loan_percent_income	0
cb_person_default_on_file	0
cb_person_cred_hist_length	0

dtype: int64

```

# Panggil fungsi median_imputation() untuk imputasi pada X_test
X_test = median_imputation(X_test, subset_data, fit=False)

```

[31] ✓ 0.0s

... Fungsi median_imputation: parameter telah divalidasi.
Fungsi median_imputation: informasi count na sebelum dilakukan imputasi:

person_age	0
person_income	0
person_home_ownership	0
person_emp_length	77
loan_intent	0
loan_grade	0
loan_amnt	0
loan_int_rate	303
loan_percent_income	0
cb_person_default_on_file	0
cb_person_cred_hist_length	0

dtype: int64

Fungsi median_imputation: informasi count na setelah dilakukan imputasi:

person_age	0
person_income	0
person_home_ownership	0
person_emp_length	0
loan_intent	0
loan_grade	0
loan_amnt	0
loan_int_rate	0
loan_percent_income	0
cb_person_default_on_file	0
cb_person_cred_hist_length	0

dtype: int64

```

# Panggil fungsi median_imputation() untuk imputasi pada X_valid
X_valid = median_imputation(X_valid, subset_data, fit=False)

```

[32] ✓ 0.0s

... Fungsi median_imputation: parameter telah divalidasi.
Fungsi median_imputation: informasi count na sebelum dilakukan imputasi:

person_age	0
person_income	0
person_home_ownership	0
person_emp_length	80
loan_intent	0
loan_grade	0
loan_amnt	0
loan_int_rate	312
loan_percent_income	0
cb_person_default_on_file	0
cb_person_cred_hist_length	0

dtype: int64

Fungsi median_imputation: informasi count na setelah dilakukan imputasi:

person_age	0
person_income	0
person_home_ownership	0
person_emp_length	0
loan_intent	0
loan_grade	0
loan_amnt	0
loan_int_rate	0
loan_percent_income	0
cb_person_default_on_file	0
cb_person_cred_hist_length	0

dtype: int64

f. [10 poin] Encoding data kategorik

i. Buat fungsi untuk membuat encoder data kategorik

- Buat fungsi bernama create_onehot_encoder()
- Fungsi tersebut memiliki parameter bernama categories dan path
- Parameter categories harus bertipe list 1 dimensi
- Parameter categories berisi daftar kategorik yang akan dibuat encodernya

- Parameter path harus bertipe string
- Parameter path berisi lokasi pada disk komputer dimana encoder yang dibuat akan disimpan
- Buat percabangan untuk validasi parameter
 - a. Percabangan pertama
 - i. Melakukan komparasi antara parameter categories dan list
 - ii. Gunakan fungsi isinstance() untuk melakukan komparasi
 - iii. Jika parameter categories tidak sama dengan list, maka raise runtimeerror dengan pesan “Fungsi create_onehot_encoder: parameter categories haruslah bertipe list, berisi kategori yang akan dibuat encodernya.”
 - iv. Jika parameter categories sama dengan list, maka eksekusi kode diluar cakupan percabangan ini
 - b. Percabangan kedua
 - i. Melakukan komparasi antara parameter path dan str
 - ii. Gunakan fungsi isinstance() untuk melakukan komparasi
 - iii. Jika parameter path tidak sama dengan str, maka raise runtimeerror dengan pesan “Fungsi create_onehot_encoder: parameter path haruslah bertipe string, berisi lokasi pada disk komputer dimana encoder akan disimpan.”
 - iv. Jika parameter path sama dengan str, maka eksekusi kode diluar cakupan percabangan ini
- Buat instance one hot encoder dengan cara memanggil fungsi OneHotEncoder()
 - a. Simpan instance tersebut pada variabel bernama ohe
- Melakukan fitting encoder
 - a. Ubah tipe data pada parameter categories yang awalnya list 1 dimensi menjadi numpy array 2 dimensi dengan menggunakan fungsi array() dari numpy yang dichain dengan fungsi reshape()
 - i. Gunakan data pada parameter categories sebagai argumen pada fungsi array()
 - ii. Berikan argument -1, 1 pada fungsi reshape()
 - b. Gunakan hasil dari ubahan tipe data parameter categories sebagai argumen pada fungsi fit dari instance ohe
- Panggil fungsi serialize_data() untuk menyimpan encoder yang baru saja difitting karena akan digunakan di masa depan

- a. Gunakan ohe dan data pada parameter path sebagai argumen pada fungsi `serialize_data()`
- Lakukan print dengan pesan “Kategori yang telah dipelajari adalah “ diikuti dengan daftar kategori yang telah dipelajari oleh ohe
 - a. Gunakan macro format (f) pada string yang akan diprint pada fungsi print
 - b. Dapatkan daftar kategori yang telah dipelajari dengan cara mengakses atribut `categories_` index ke 0
 - i. Ubah ke list dengan menggunakan fungsi `tolist()` setelah mengakses index ke 0 dari data `categories_`
 - ii. Gunakan hasil perubahan tersebut sebagai argumen pada fungsi print untuk ditampilkan beserta pesan sebelumnya
- Kembalikan ohe dari fungsi

Lampirkan screenshot

f. Encoding data kategorikal

```
# Fungsi untuk membuat OneHotEncoder dan menyimpannya ke directory computer
def create_onehot_encoder(categories, path):
    """
    Fungsi untuk membuat OneHotEncoder dan menyimpannya ke directory.

    Parameters:
    categories : list
        Daftar kategorik yang akan dibuat encodernya.
    path : str
        Lokasi pada disk komputer dimana encoder yang dibuat akan disimpan.

    Returns:
    ohe : OneHotEncoder
        Instance dari OneHotEncoder yang telah difit dengan data kategorik.
    """

    # Validasi parameter categories
    if not isinstance(categories, list):
        raise RuntimeError("Fungsi create_onehot_encoder: parameter categories haruslah bertipe list, berisi kategori yang akan dibuat encodernya.")

    # Validasi parameter path
    if not isinstance(path, str):
        raise RuntimeError("Fungsi create_onehot_encoder: parameter path haruslah bertipe string, berisi lokasi pada disk komputer dimana encoder akan disimpan.")

    # Buat instance OneHotEncoder
    ohe = OneHotEncoder(handle_unknown='ignore')

    # Melakukan fitting encoder
    categories_array = np.array(categories).reshape(-1, 1)
    ohe.fit(categories_array)

    # Menyimpan encoder ke disk
    serialize_data(ohe, path)

    # Print kategori yang telah dipelajari
    learned_categories = ohe.categories_[0].tolist()
    print(f"Kategori yang telah dipelajari adalah {learned_categories}")

    print(f"Kategori yang telah dipelajari adalah {learned_categories}")

    return ohe
```

ii. Jalankan fungsi pembuatan encoder

- Buat empat konstan variabel bertipe list dan bernama `person_home_ownership`, `loan_intent`, `loan_grade`, `cb_person_default_on_file`
- Isi variabel tersebut dengan kategorik dari kolom masing-masing
- Panggil fungsi `create_onehot_encoder()`

- a. Berikan variabel `person_home_ownership` dan path `models/ohe_home_ownership.pkl` sebagai argumen untuk parameter `categories` dan `path` secara berurutan
- b. Simpan keluaran fungsi pada variabel bernama `ohe_home_ownership`
- Panggil ulang fungsi `create_onehot_encoder()`
 - a. Berikan variabel `loan_intent` dan path `models/ohe_loan_intent.pkl` sebagai argumen untuk parameter `categories` dan `path` secara berurutan
 - b. Simpan keluaran fungsi pada variabel bernama `ohe_loan_intent`
- Panggil ulang fungsi `create_onehot_encoder()`
 - a. Berikan variabel `loan_grade` dan path `models/ohe_loan_grade.pkl` sebagai argumen untuk parameter `categories` dan `path` secara berurutan
 - b. Simpan keluaran fungsi pada variabel bernama `ohe_loan_grade`
- Panggil ulang fungsi `create_onehot_encoder()`
 - a. Berikan variabel `cb_person_default_on_file` dan path `models/ohe_default_on_file.pkl` sebagai argumen untuk parameter `categories` dan `path` secara berurutan
 - b. Simpan keluaran fungsi pada variabel bernama `ohe_default_on_file`

Lampirkan screenshot

```

nprint(X_train['person_home_ownership'].unique())
print(X_train['loan_intent'].unique())
print(X_train['loan_grade'].unique())
print(X_train['cb_person_default_on_file'].unique())

[38] ✓ 0.0s

... ['MORTGAGE' 'RENT' 'OWN' 'OTHER']
['EDUCATION' 'PERSONAL' 'MEDICAL' 'VENTURE' 'HOMEIMPROVEMENT'
'DEBTCONSOLIDATION']
['A' 'B' 'C' 'D' 'F' 'E' 'G']
['N' 'Y']

# Daftar kategorik dari kolom masing-masing
person_home_ownership = ['MORTGAGE' 'RENT' 'OWN' 'OTHER']
loan_intent = ['EDUCATION', 'PERSONAL', 'MEDICAL', 'VENTURE', 'HOMEIMPROVEMENT', 'DEBTCONSOLIDATION']
loan_grade = ['A', 'B', 'C', 'D', 'F', 'E', 'G']
cb_person_default_on_file = ['N', 'Y']

# Membuat encoder untuk masing-masing kategorik dan menyimpannya
ohe_home_ownership = create_onehot_encoder(person_home_ownership, '../models/ohe_home_ownership.pkl')
ohe_loan_intent = create_onehot_encoder(loan_intent, '../models/ohe_loan_intent.pkl')
ohe_loan_grade = create_onehot_encoder(loan_grade, '../models/ohe_loan_grade.pkl')
ohe_default_on_file = create_onehot_encoder(cb_person_default_on_file, '../models/ohe_default_on_file.pkl')

[43] ✓ 0.2s

... Kategori yang telah dipelajari adalah ['MORTGAGERENTOWNOTHER']
Kategori yang telah dipelajari adalah ['DEBTCONSOLIDATION', 'EDUCATION', 'HOMEIMPROVEMENT', 'MEDICAL', 'PERSONAL', 'VENTURE']
Kategori yang telah dipelajari adalah ['A', 'B', 'C', 'D', 'F', 'E', 'G']
Kategori yang telah dipelajari adalah ['N', 'Y']

```

- iii. Buat fungsi untuk melakukan encoding data kategorik
 - Buat fungsi bernama `ohe_transform()`

- Fungsi tersebut memiliki parameter bernama dataset, subset, prefix dan ohe
- Parameter dataset harus bertipe dataframe
- Parameter dataset merupakan set data yang ingin dilakukan pengkodean
- Parameter subset harus bertipe string
- Parameter subset merupakan nama kolom yang terdapat pada data di parameter dataset
- Parameter prefix harus bertipe string
- Parameter prefix merupakan nama awalan yang akan disematkan pada kolom hasil pengkodean
- Parameter ohe harus bertipe OneHotEncoder dari sklearn.preprocessing
- Parameter ohe merupakan encoder yang sebelumnya telah dilatih oleh data kategorik khusus
- Buat percabangan guna validasi paramater
 - a. Percabangan pertama
 - i. Melakukan komparasi antara parameter dataset dan dataframe
 - ii. Gunakan fungsi isinstance() untuk melakukan komparasi
 - iii. Jika parameter dataset tidak sama dengan dataset, maka raise runtimeerror dengan pesan "Fungsi ohe_transform: parameter dataset harus bertipe DataFrame!"
 - iv. Jika parameter dataset sama dengan dataframe, maka eksekusi kode diluar cakupan percabangan ini
 - b. Percabangan kedua
 - i. Melakukan komparasi antara parameter ohe dan OneHotEncoder
 - ii. Gunakan fungsi isinstance() untuk melakukan komparasi
 - iii. Jika parameter ohe tidak sama dengan OneHotEncoder, maka raise runtimeerror dengan pesan "Fungsi ohe_transform: parameter ohe harus bertipe OneHotEncoder!"
 - iv. Jika parameter ohe sama dengan OneHotEncoder, maka eksekusi kode diluar cakupan percabangan ini
 - c. Percabangan ketiga

- i. Melakukan komparasi antara parameter prefix dan str
 - ii. Gunakan fungsi `isinstance()` untuk melakukan komparasi
 - iii. Jika parameter prefix tidak sama dengan str, maka `raise runtimeerror` dengan pesan "Fungsi `ohe_transform`: parameter prefix harus bertipe str!"
 - iv. Jika parameter prefix sama dengan str, maka eksekusi kode diluar cakupan percabangan ini
 - d. Percabangan keempat
 - i. Melakukan komparasi antara parameter subset dan str
 - ii. Gunakan fungsi `isinstance()` untuk melakukan komparasi
 - iii. Jika parameter subset tidak sama dengan str, maka `raise runtimeerror` dengan pesan "Fungsi `ohe_transform`: parameter subset harus bertipe str!"
 - iv. Jika parameter subset sama dengan str, maka eksekusi kode diluar cakupan percabangan ini
 - e. Percabangan kelima
 - i. Melakukan pengecekan data pada parameter subset di daftar nama kolom pada parameter dataset
 - ii. Buat blok pengetesan kode program untuk melakukan hal tersebut
 - 1. Pada bagian try, buat list dari semua nama kolom yang ada pada parameter dataset
 - a. Gunakan fungsi `index` untuk mendapatkan nama kolom yang diinginkan
 - b. Berikan parameter subset sebagai argumen pada fungsi `index`
 - 2. Pada bagian except
 - a. `Raise runtimeerror` dengan pesan "Fungsi `ohe_transform`: parameter subset string namun data tidak ditemukan dalam daftar kolom yang terdapat pada parameter dataset."
- Print pesan "Fungsi `ohe_transform`: parameter telah divalidasi."
- Buat duplikat dari data pada parameter dataset dan simpan duplikatnya pada variabel bernama dataset (`direplace`)
- Print pesan yang menampilkan daftar nama kolom sebelum dilakukan pengkodean

- a. Gunakan macro format (f) pada string yang akan diprint pada fungsi print
 - b. Pesannya adalah “Fungsi ohe_transform: daftar nama kolom sebelum dilakukan pengkodean adalah [daftar nama kolom].”
 - c. Akses data nama kolom pada dataset dengan cara mengakses atribut columns
 - d. Ubah daftar nama kolom ke tipe data list dengan fungsi list
 - e. Gunakan daftar nama kolom tersebut sebagai argumen pada fungsi print
- Buat satu variabel bernama col_names untuk menyimpan nama kolom yang telah dikodekan
 - a. Nama kolom dibuat dengan menggunakan list comprehension
 - b. Buat perulangan dalam list comprehension dari daftar kategori yang ada pada parameter ohe
 - i. Data daftar kategori dapat diakses melalui atribut categories_index ke 0
 - ii. Ubah tipe data daftar kategori tersebut ke list dengan menggunakan fungsi tolist()
 - iii. Simpan nama kategori di tiap iterasi perulangan ke variabel bernama col_name
 - iv. Dalam tiap iterasi perulangan, gabungkan data pada parameter prefix, karakter garis bawah, serta data pada variabel col_name
- Proses pengkodean:
 - a. Buat satu variabel bernama encoded
 - b. Variabel tersebut diisi oleh sebuah dataframe
 - c. Data pada dataframe tersebut didapatkan dari hasil transformasi ohe
 - d. Proses transformasi ohe:
 - i. Panggil fungsi transform dari variabel ohe
 - ii. Ubah subset ke tipe data list dengan cara menambahkan kurung siku pada kedua sisinya
 - iii. Gunakan list subset tersebut sebagai selektor dari dataset
 - iv. Hasil seleksi dataset dijadikan argumen dari fungsi transform ohe
 - v. Ubah tipe data hasil transform ohe ke array dengan cara chaining fungsi toarray()
 - vi. Gunakan array hasil transformasi tersebut sebagai argumen pertama pada pembuatan dataframe di poin c

- e. Gunakan `col_names` sebagai argumen dari parameter `columns` pada pembuatan dataframe di poin c
- f. Set index dari dataframe yang dibuat di poin c dengan cara
 - i. Ambil index dari dataset dengan mengakses atribut `indexnya`
 - ii. Gunakan data tersebut sebagai argumen dari parameter `index` pada pembuatan dataframe di poin c
- Proses penyatuan hasil pengkodean dengan data sebelum pengkodean
 - a. Panggil fungsi `concat()` dari `pandas`
 - b. Untuk argumen pertama, buat list yang berisi data sebelum pengkodean dan data setelah pengkodean
 - c. Set parameter `axis` menjadi satu (1)
 - d. Simpan hasil dari fungsi `concat()` ke variabel bernama `dataset` (`direplace`)
- Proses penghapusan kolom yang tidak diperlukan
 - a. Menghapus kolom dari dataframe yang telah dikodekan
 - b. Panggil fungsi `drop` dari dataframe
 - c. Untuk parameter `columns` pada fungsi `drop`, berikan list dari subset sebagai argumen
 - i. Ubah tipe data subset ke list dapat dilakukan dengan cara memberikan kurung siku pada kedua sisinya
 - d. Untuk parameter `inplace` pada fungsi `drop`, berikan nilai `True`
- Print pesan yang menandakan bahwa proses pengkodean telah berhasil
 - a. Gunakan macro format (f) pada string yang akan diprint pada fungsi `print`
 - b. Pesannya adalah "Fungsi `ohe_transform`: daftar nama kolom setelah dilakukan pengkodean adalah [daftar nama kolom]."
 - c. Akses data nama kolom pada dataset dengan cara mengakses atribut `columns`
 - d. Ubah daftar nama kolom ke tipe data list dengan fungsi `list`
 - e. Gunakan daftar nama kolom tersebut sebagai argumen pada fungsi `print`
 - f. Kembalikan dataset dari fungsi

Lampirkan screenshot

Transform OHE

```
def ohe_transform(dataset, subset, prefix, ohe):
    """
    Function to apply OneHotEncoder to a specified column in a dataset.

    Parameters:
    dataset : pd.DataFrame
        The dataset to which encoding will be applied.
    subset : str
        The name of the column in the dataset to be encoded.
    prefix : str
        The prefix to be added to the names of the new encoded columns.
    ohe : OneHotEncoder
        The OneHotEncoder instance that has been fitted to the categorical data.

    Returns:
    dataset : pd.DataFrame
        The dataset with the encoded column.
    """

    # Validate parameters
    if not isinstance(dataset, pd.DataFrame):
        raise RuntimeError("Fungsi ohe_transform: parameter dataset harus bertipe DataFrame!")

    if not isinstance(ohe, OneHotEncoder):
        raise RuntimeError("Fungsi ohe_transform: parameter ohe harus bertipe OneHotEncoder!")

    if not isinstance(prefix, str):
        raise RuntimeError("Fungsi ohe_transform: parameter prefix harus bertipe str!")

    if not isinstance(subset, str):
        raise RuntimeError("Fungsi ohe_transform: parameter subset harus bertipe str!")

    try:
        dataset.columns.tolist().index(subset)
    except ValueError:
        raise RuntimeError("Fungsi ohe_transform: parameter subset string namun data tidak ditemukan dalam daftar kolom yang terdapat pada parameter dataset.")

    print("Fungsi ohe_transform: parameter telah divalidasi.")

    # Duplicate the dataset to avoid modifying the original data
    dataset = dataset.copy()

    print(f"Fungsi ohe_transform: daftar nama kolom sebelum dilakukan pengkodean adalah {list(dataset.columns)}.")

    # Create new column names for the encoded columns
    col_names = [f"{prefix}_{col_name}" for col_name in ohe.categories_[0].tolist()]

    # Perform encoding
    encoded = pd.DataFrame(ohe.transform(dataset[[subset]]).toarray(), columns=col_names, index=dataset.index)

    # Concatenate the original dataset with the new encoded columns
    dataset = pd.concat([dataset, encoded], axis=1)

    # Drop the original column that was encoded
    dataset.drop(columns=[subset], inplace=True)

    print(f"Fungsi ohe_transform: daftar nama kolom setelah dilakukan pengkodean adalah {list(dataset.columns)}.")

    return dataset
```

[44] ✓ 0.0s

iv. Jalankan fungsi untuk melakukan encoding data kategorik

- Untuk X_train

- a. Untuk kolom person_home_ownership

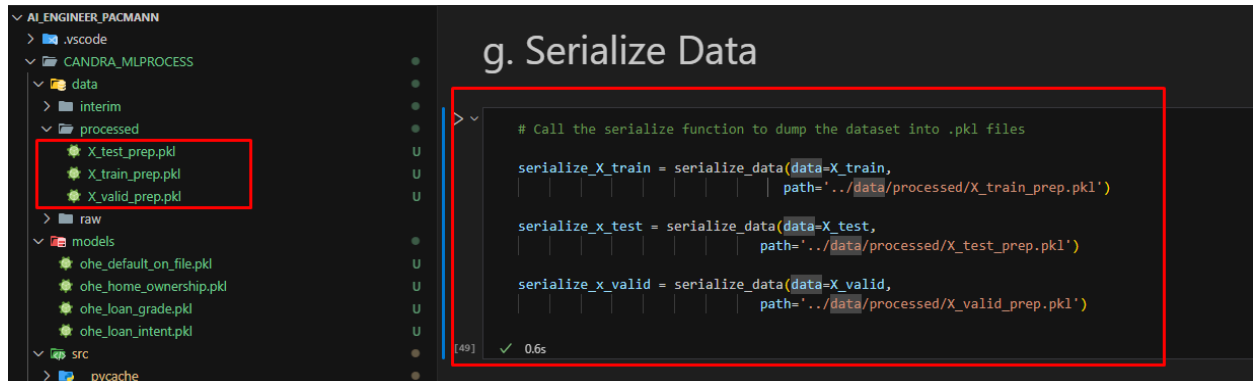
- i. Panggil fungsi ohe_transform()
 - ii. Berikan X_train sebagai argumen pertama
 - iii. Berikan string "person_home_ownership" sebagai argumen kedua
 - iv. Berikan string "home_ownership" sebagai argumen ketiga
 - v. Berikan OneHotEncoder ohe_home_ownership sebagai argumen keempat

- vi. Simpan hasil transform ke variabel bernama X_train
- b. Untuk kolom loan_intent
 - i. Panggil fungsi ohe_transform()
 - ii. Berikan X_train sebagai argumen pertama
 - iii. Berikan string "loan_intent" sebagai argumen kedua
 - iv. Berikan string "loan_intent" sebagai argumen ketiga
 - v. Berikan OneHotEncoder ohe_loan_intent sebagai argumen keempat
 - vi. Simpan hasil transform ke variabel bernama X_train
- c. Untuk kolom loan_grade
 - i. Panggil fungsi ohe_transform()
 - ii. Berikan X_train sebagai argumen pertama
 - iii. Berikan string "loan_grade" sebagai argumen kedua
 - iv. Berikan string "loan_grade" sebagai argumen ketiga
 - v. Berikan OneHotEncoder ohe_loan_grade sebagai argumen keempat
 - vi. Simpan hasil transform ke variabel bernama X_train
- d. Untuk kolom cb_person_default_on_file
 - i. Panggil fungsi ohe_transform()
 - ii. Berikan X_train sebagai argumen pertama
 - iii. Berikan string "cb_person_default_on_file" sebagai argumen kedua
 - iv. Berikan string "default_onfile" sebagai argumen ketiga
 - v. Berikan OneHotEncoder ohe_defaul_on_file sebagai argumen keempat
 - vi. Simpan hasil transform ke variabel bernama X_train
- Lakukan hal yang sama dengan poin "Untuk X_train" pada X_test dan juga X_valid, ubah argumen pertama dari X_train ke X_test dan X_valid

Lampirkan screenshot

- Berikan string data/processed/X_train_prep.pkl sebagai argumen kedua
- Ulangi 3 poin diatas untuk X_test dan X_valid
 - a. Ubah argumen pertama ke X_test dan X_valid
 - b. Ubah nama file dalam argumen kedua dengan X_test_prep.pkl dan X_valid_prep.pkl
 - c. Lokasi folder dalam string untuk argumen kedua tetap sama

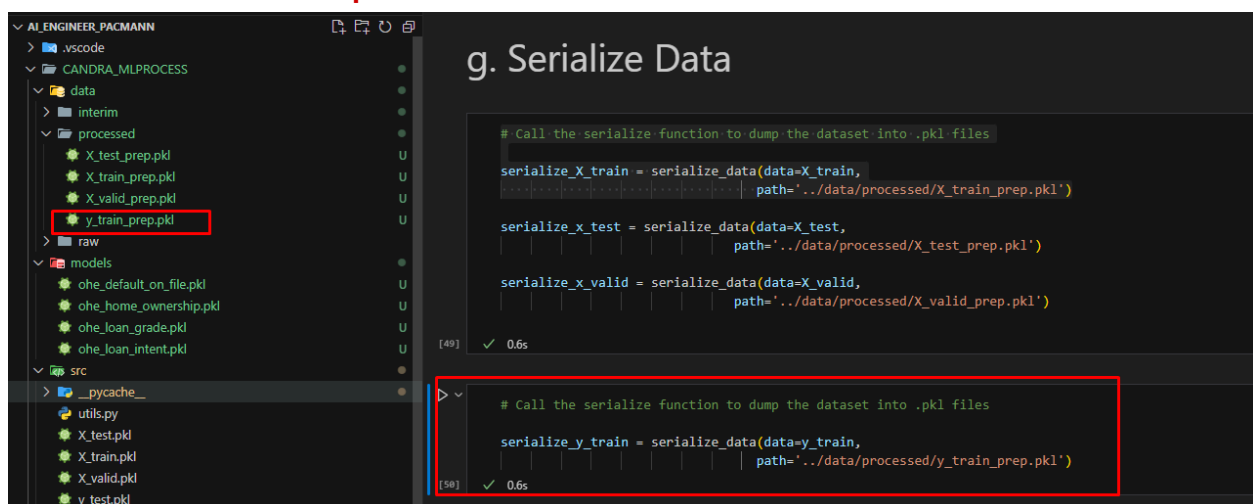
Lampirkan screenshot



ii. Serialize y data

- Panggil fungsi serialize_data
- Berikan dataframe y_train sebagai argumen pertama
- Berikan string data/processed/y_train_prep.pkl sebagai argumen kedua

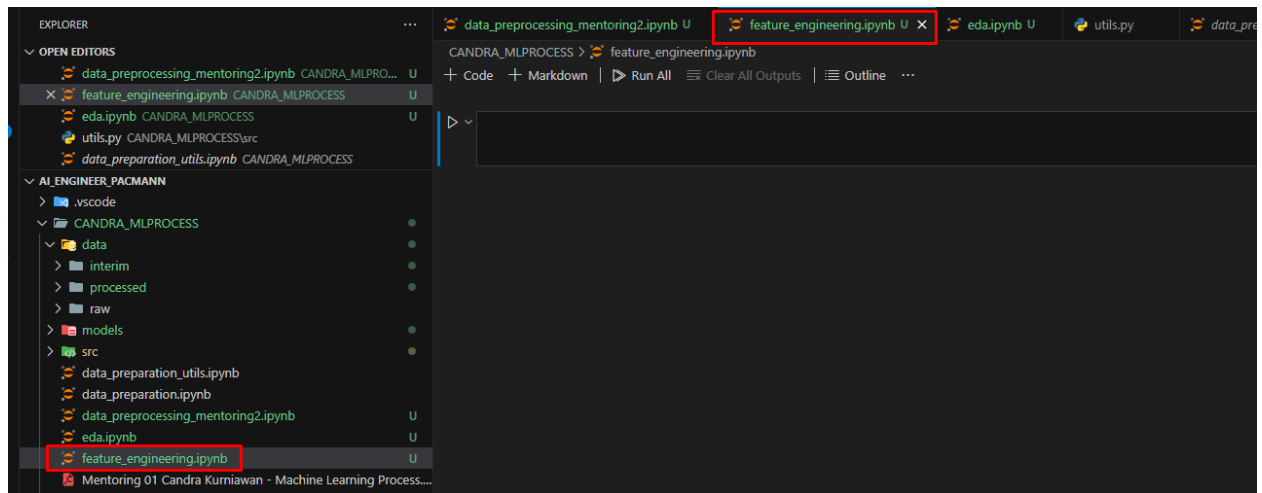
Lampirkan screenshot



3. [10 poin] Feature Engineering

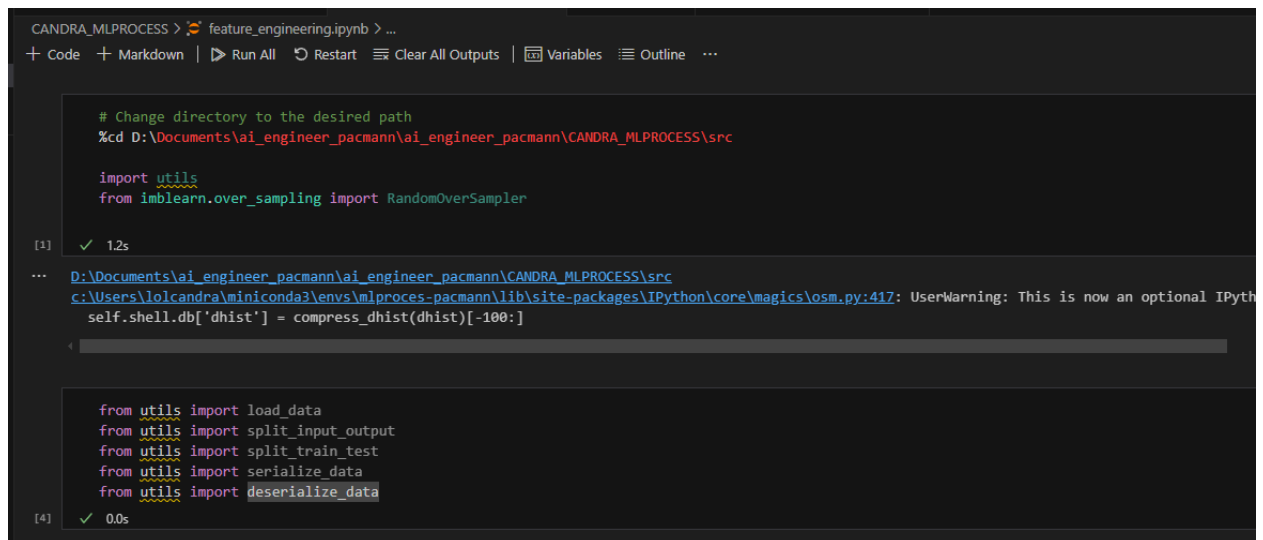
- a. [1 poin] Buat satu file bernama feature_engineering.ipynb

Lampirkan screenshot



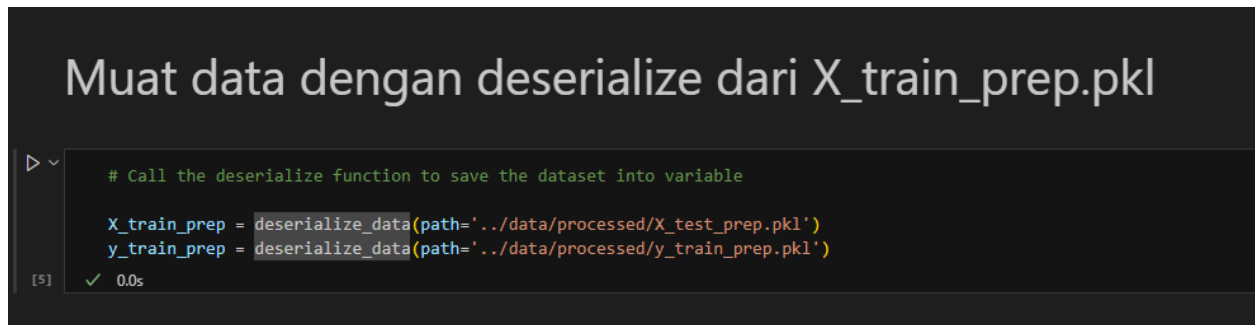
- b. [1 poin] Import utils dari src dan import RandomOverSampler dari imblearn.over_sampling

Lampirkan screenshot



- c. [1 poin] Muat data train set yang telah dilakukan preprocessing
- Panggil fungsi `deserialize_data()` dari `utils`
 - Berikan argumen berupa lokasi berkas `X_train_prep.pkl`
 - Simpan keluaran fungsi ke `X_train_prep`
 - Panggil ulang fungsi `deserialize_data()` dari `utils`
 - Berikan argumen berupa lokasi berkas `y_train_prep.pkl`
 - Simpan keluaran fungsi ke `y_train_prep`

Lampirkan screenshot



The screenshot shows a Jupyter Notebook interface with a dark theme. The title of the cell is "Muat data dengan deserialize dari X_train_prep.pkl". The code cell contains two lines of Python code: `# Call the deserialize function to save the dataset into variable` and `X_train_prep = deserialize_data(path='../data/processed/X_test_prep.pkl')` followed by `y_train_prep = deserialize_data(path='../data/processed/y_train_prep.pkl')`. The output of the cell is `[5] ✓ 0.0s`.

```
# Call the deserialize function to save the dataset into variable

X_train_prep = deserialize_data(path='../data/processed/X_test_prep.pkl')
y_train_prep = deserialize_data(path='../data/processed/y_train_prep.pkl')
```

[5] ✓ 0.0s

- d. [1 poin] Buat instance `RandomOverSampler()`

Lampirkan screenshot



The screenshot shows a Jupyter Notebook interface with a dark theme. The title of the cell is "RandomOverSampler()". The code cell contains two lines of Python code: `# Membuat instance RandomOverSampler` and `ros = RandomOverSampler(random_state=42)`. The output of the cell is `[6] ✓ 0.0s`.

```
# Membuat instance RandomOverSampler
ros = RandomOverSampler(random_state=42)
```

[6] ✓ 0.0s

- e. [1 poin] Simpan instance tersebut dalam variabel bernama `ros`

Lampirkan screenshot

RandomOverSampler()



```
# Membuat instance RandomOverSampler  
ros = RandomOverSampler(random_state=42)
```

[6]



0.0s

- f. [1 poin] Buat grafik tentang kondisi label sebelum dilakukan oversampling
- Gunakan fungsi countplot dari seaborn
 - Berikan y_train_prep yang telah diubah ke dataframe sebagai argumen untuk parameter data
 - Berikan string loan_status sebagai argumen untuk parameter x dan hue
- Lampirkan screenshot**

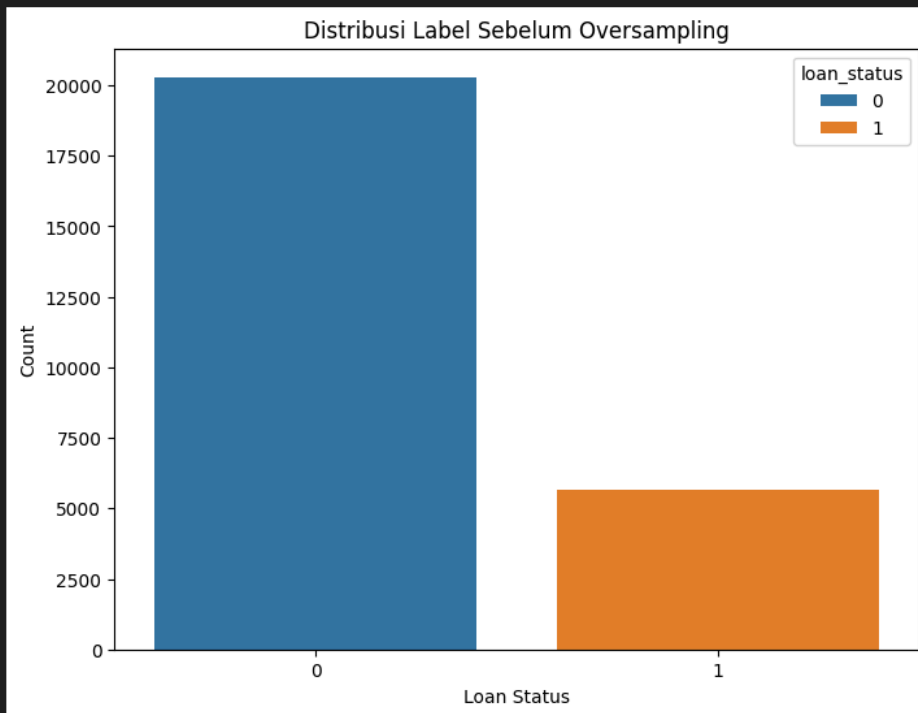
```

# Mengubah y_train_prep dari Series menjadi DataFrame
y_train_prep_df = pd.DataFrame(y_train_prep, columns=['loan_status'])

# Membuat grafik countplot menggunakan seaborn
plt.figure(figsize=(8, 6))
sns.countplot(data=y_train_prep_df, x='loan_status', hue='loan_status')
plt.title("Distribusi Label Sebelum Oversampling")
plt.xlabel("Loan Status")
plt.ylabel("Count")
plt.show()

```

[10] ✓ 0.1s



- g. [2 poin] Lakukan oversampling data tersebut
- Panggil fungsi `fit_resample` dari variabel `ros`
 - Berikan `X_train_prep` sebagai argumen pertama
 - Berikan `y_train_prep` sebagai argumen kedua
 - Simpan keluaran dari fungsi `fit_resample()` ke dua variabel bernama `X_train_ros` dan `y_train_ros`

Lampirkan screenshot

Melakukan OverSampling

```
from imblearn.over_sampling import RandomOverSampler

# Melakukan oversampling
X_train_ros, y_train_ros = ros.fit_resample(X_train_prep, y_train_prep_df)

# Menampilkan hasil oversampling
print("Sebelum oversampling:")
print(f"X_train_prep shape: {X_train_prep.shape}")
print(f"y_train_prep distribution:\n{y_train_prep_df.value_counts()}")

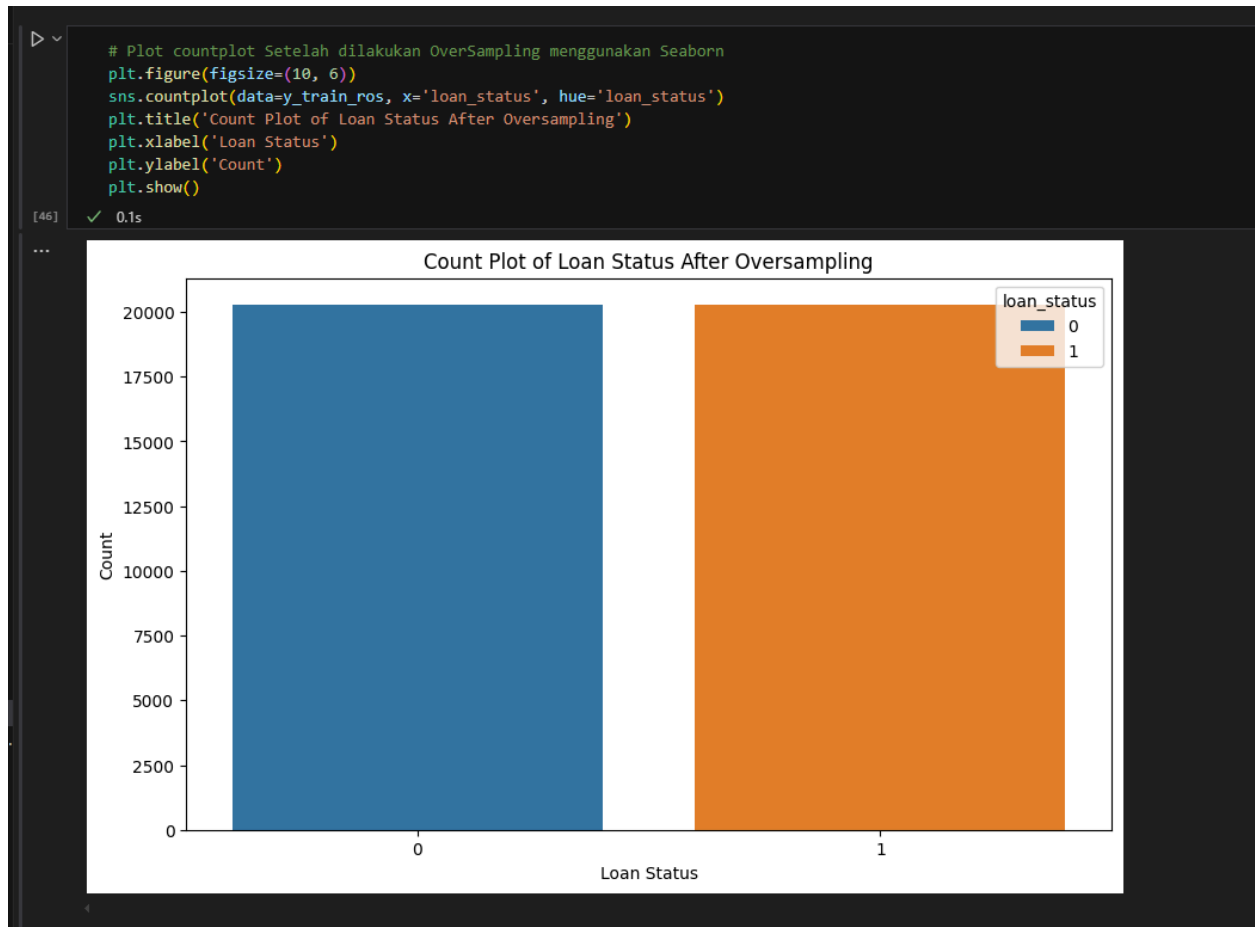
print("\nSetelah oversampling:")
print(f"X_train_ros shape: {X_train_ros.shape}")
print(f"y_train_ros distribution:\n{y_train_ros.value_counts()}")
```

[42] ✓ 0.0%

```
... Sebelum oversampling:
X_train_prep shape: (25968, 23)
y_train_prep distribution:
loan_status
0          20291
1           5677
Name: count, dtype: int64

Setelah oversampling:
X_train_ros shape: (40582, 23)
y_train_ros distribution:
loan_status
0          20291
1          20291
Name: count, dtype: int64
```

- h. [1 poin] Buat grafik tentang kondisi label setelah dilakukan oversampling
- Gunakan fungsi countplot dari seaborn
 - Berikan y_train_ros yang telah diubah ke dataframe sebagai argumen untuk parameter data
 - Berikan string loan_status sebagai argumen untuk parameter x dan hue
- Lampirkan screenshot**



- i. [1 poin] Serialize train set setelah resample
 - i. Panggil fungsi `serialize_data()` dari `utils`
 - ii. Berikan `X_train_ros` sebagai argumen pertama
 - iii. Berikan string `data/processed/X_train_ros.pkl` sebagai argumen kedua
 - iv. Panggil ulang fungsi `serialize_data()` dari `utils`
 - v. Berikan `y_train_ros` sebagai argumen pertama
 - vi. Berikan string `data/processed/y_train_ros.pkl` sebagai argumen kedua

Lampirkan screenshot

