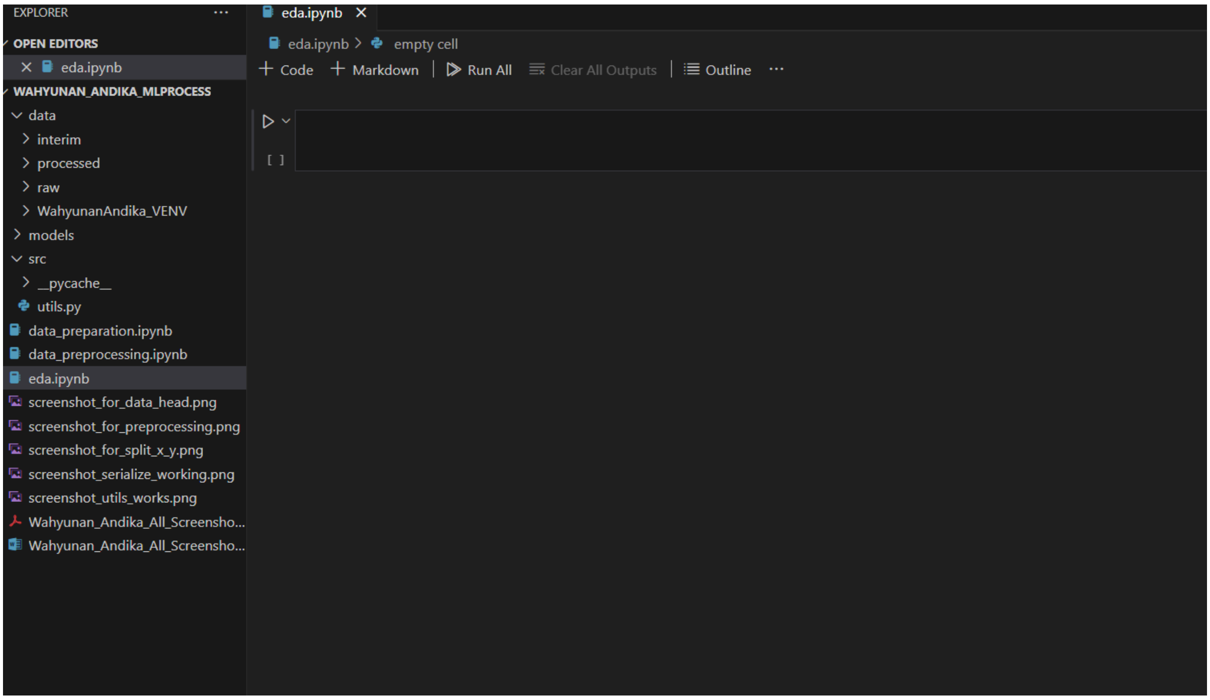
Buatlah satu file bernama eda.ipynb di root folder project Anda



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

```python
import pandas as pd
import seaborn as sns
import sys
sys.path.append('./src')
import utils
```

[2] ✓ 0.8s

Muat X_train data

```python
X_TRAIN_PATH = "data/interim/X_train.pkl"
x_train = utils.deserialize_data(X_TRAIN_PATH)
x_train.head()
```

✓ 0.0s Open 'X_train' in Data Wrangler Python

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_percent_income	cb_person_
15884	25	241875	MORTGAGE	4.0	EDUCATION	A	16000	7.05	0.07	
15138	21	18000	RENT	5.0	PERSONAL	B	1500	12.18	0.08	
7474	25	53000	MORTGAGE	10.0	MEDICAL	B	16000	12.53	0.30	
18212	28	16800	OWN	NaN	MEDICAL	C	5000	13.98	0.30	
6493	25	50000	MORTGAGE	2.0	VENTURE	A	10000	7.90	0.20	

Muat Y_train data

```python
Y_TRAIN_PATH = "data/interim/y_train.pkl"

y_train = utils.deserialize_data(Y_TRAIN_PATH)

y_train.head()
```

✓ 0.1s Open 'y_train' in Data Wrangler

```
15884    0
15138    1
7474     0
18212    1
6493     0
Name: loan_status, dtype: int64
```

X_train.head()

```python
X_train.head()
```

✓ 0.0s Open 'X_train' in Data Wrangler Python

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_percent_income	cb_person_c
15884	25	241875	MORTGAGE	4.0	EDUCATION	A	16000	7.05	0.07	
15138	21	18000	RENT	5.0	PERSONAL	B	1500	12.18	0.08	
7474	25	53000	MORTGAGE	10.0	MEDICAL	B	16000	12.53	0.30	
18212	28	16800	OWN	NaN	MEDICAL	C	5000	13.98	0.30	
6493	25	50000	MORTGAGE	2.0	VENTURE	A	10000	7.90	0.20	

Split numerical and categorical

```python
num_col = ['person_age', 'person_income', 'person_emp_length', 'loan_amnt', 'loan_int_rate', 'loan_percent_income', 'cb_person_cred_hist_length']
cat_col = ['person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file']
```

✓ 0.0s Pyth

Info

```python
X_train.info()
```

✓ 0.0s

```
<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
```

Conclusion

```markdown
# Conclusion

Just wrapped up the initial look at the `X_train` dataset, and here’s what stands out:

1. **Data Types**:
   - The numerical columns are sorted out, with integers and floats identified. However, the `person_emp_length` column might need a second glance to make sure it’s classified correctly.
   - The categorical columns are clear, which will be helpful for the next steps.

2. **Missing Values**:
   - Spotted some missing values in the `person_emp_length` column. Gotta tackle those before moving on, either by filling them in or dropping those rows.

3. **Data Consistency**:
   - The categorical data, like `person_home_ownership` and `loan_intent`, look pretty consistent, but a quick check wouldn’t hurt to catch any sneaky input errors.

## Next Steps
- **Fixing Missing Values**: Time to think about how to deal with those gaps in `person_emp_length`. Filling them in or removing those entries is on the agenda.
- **Data Cleaning**: Checking for duplicates and ensuring all columns have the right data types is essential.
- **Descriptive Analysis**: A deeper dive into the stats will help understand how the data is distributed and what it’s all about.
- **Visuals**: Creating some visualizations can really help in seeing the relationships between different variables before diving into modeling.
```

Urutkan data berdasarkan data pada kolom person income

```python
duplicates = X_train[X_train.duplicated(keep=False)]

duplicates_sorted = duplicates.sort_values(by='person_income')

duplicates_sorted
```

✓ 0.0s Open 'duplicates_sorted' in Data Wrangler Python

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_percent_income	cb
15952	24	7800	RENT	1.0	EDUCATION	B	1000	11.36	0.13	
16821	24	7800	RENT	1.0	EDUCATION	B	1000	11.36	0.13	
2431	21	15600	RENT	0.0	MEDICAL	A	2800	7.40	0.18	
17758	21	15600	RENT	0.0	MEDICAL	A	2800	7.40	0.18	
28295	32	18000	OWN	0.0	VENTURE	A	4750	7.88	0.26	
...	...	...	...	...	...	...	...	...	...	
27677	35	160000	OWN	10.0	VENTURE	B	24000	11.83	0.15	
32047	36	250000	RENT	2.0	DEBTCONSOLIDATION	A	20000	7.88	0.08	
29160	36	250000	RENT	2.0	DEBTCONSOLIDATION	A	20000	7.88	0.08	
27881	28	604000	MORTGAGE	12.0	PERSONAL	B	25000	9.01	0.04	
28770	28	604000	MORTGAGE	12.0	PERSONAL	B	25000	9.01	0.04	

192 rows × 11 columns

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

```
### Conclusion

Just finished checking for duplicates in the `X_train` dataset, and here's what I found:

1. **Duplicate Entries**: Found some duplicate rows in the dataset. It's important to clean these up to ensure the analysis and model training are accurate.
2. **Sorting**: Sorted the duplicates by `person_income`, which makes it easier to identify patterns or issues related to income.

### Next Steps
- **Handling Duplicates**: Decide whether to drop all duplicate entries or just one of each set. This will depend on the context of the data and whether the duplicates represent valid data points.
- **Data Cleaning**: Once duplicates are handled, a general data cleaning step will be next, including checking for any remaining missing values or inconsistencies.
- **Further Analysis**: After cleaning, diving deeper into the dataset to explore relationships between variables would be beneficial. This can help in understanding the data better before moving on to modeling.
```

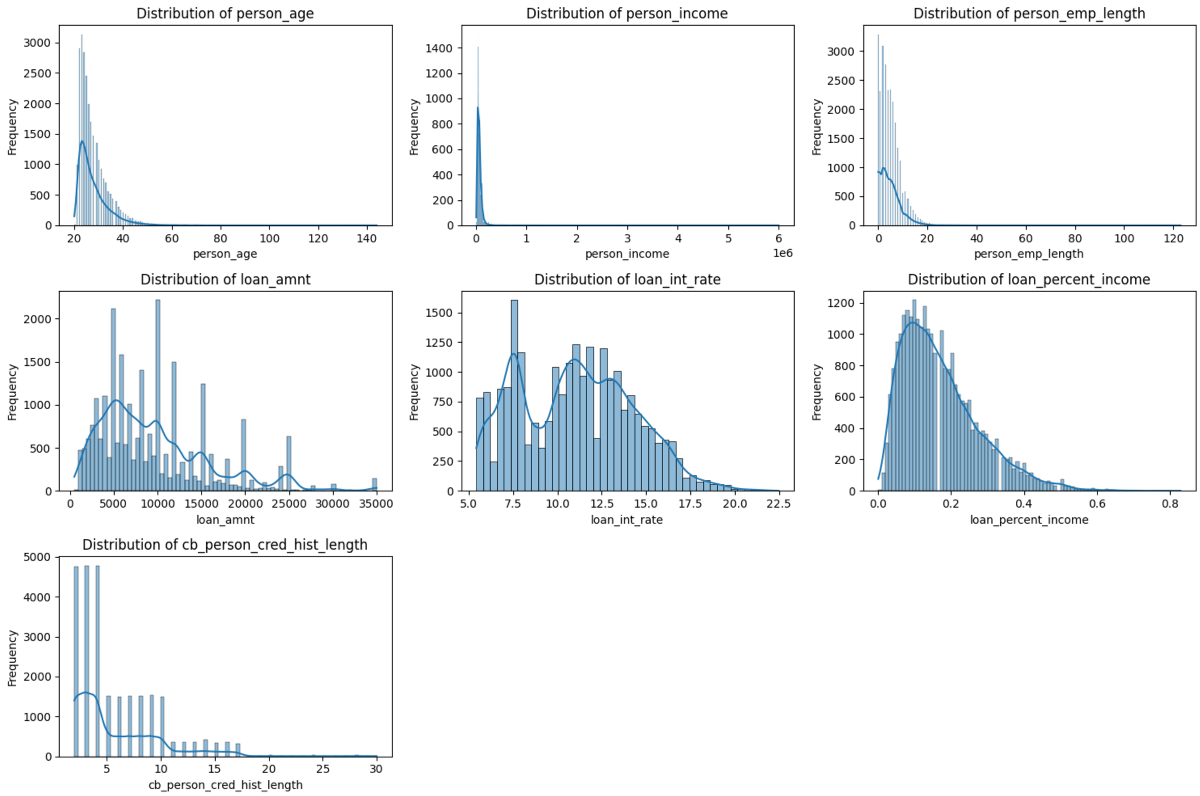
Null value

```python
null_values = X_train.isnull().sum()

null_values[null_values > 0]
```

✓ 0.0s

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

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)



Pada tiap iterasi perulangan, panggil fungsi displot() dari library seaborn

```
import matplotlib.pyplot as plt

for col in X_train_.columns:
    sns.displot(X_train_[col], bins=20, kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

✓ 2.3s

Conclusion

```
### Conclusion and Next Steps

From the distribution plots generated for each numeric column, I can see how the data is spread out. For example, some features like
`loan_int_rate` seem to have a skewed distribution, which might affect certain machine learning algorithms.

To prepare the data for modeling, I might consider applying transformations (like log transformation) to handle skewness. Additionally, I need to
look into the null values in the `person_emp_length` and `loan_int_rate` columns. Depending on their impact, I could either fill these missing
values or drop those rows entirely.

Next, I'll clean up the data based on these observations, addressing the skewness and dealing with null values to ensure the dataset is ready for
the modeling phase.
```

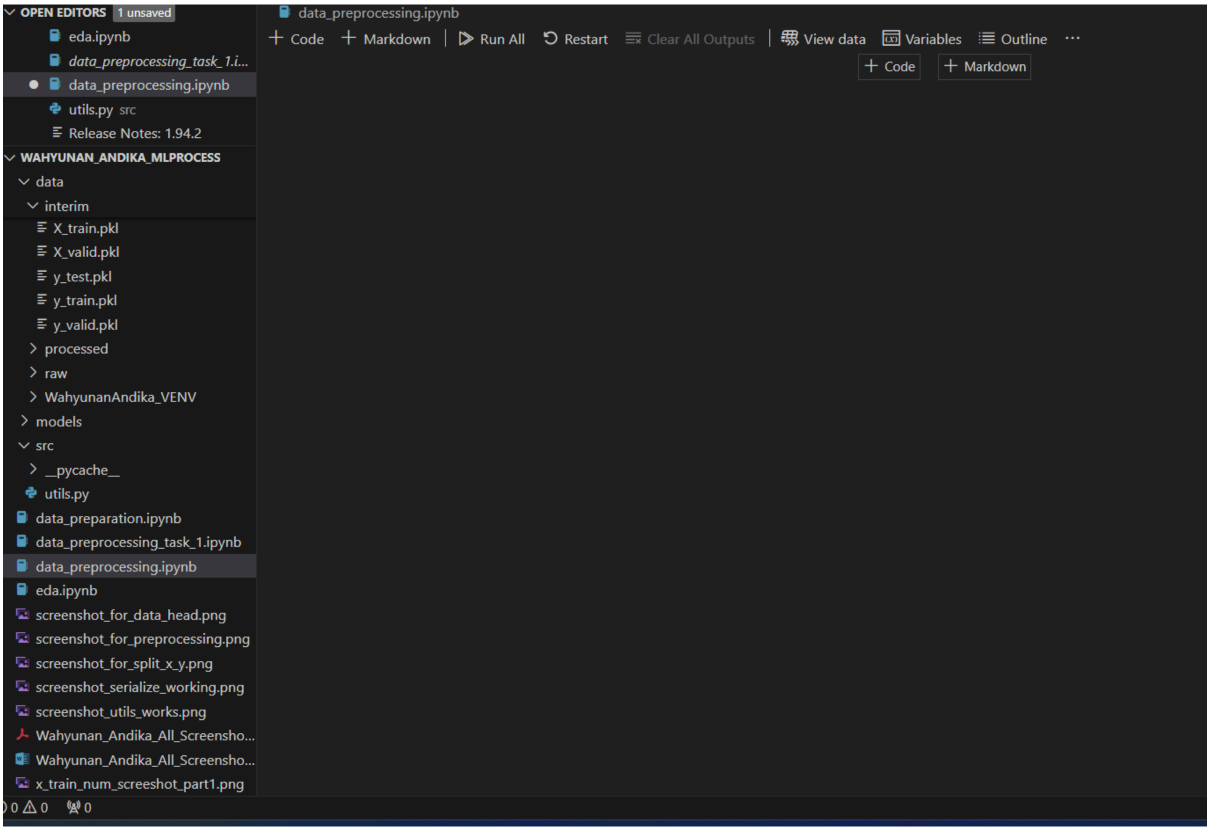
Conlusion y_target

```markdown
### Conclusion

After checking the balance of the target variable (`loan_status`), it is clear that both classes (0 and 1) are represented in the training data.
This distribution provides a good foundation for building a predictive model, as there is sufficient data for both outcomes.

#### Next Steps
Moving forward, I will take the following actions:
1. **Explore Class Imbalance**: Although both classes are present, I will analyze the distribution further to determine if there is a significant
imbalance that may affect model performance.
2. **Preprocessing**: Based on the insights gained from this distribution analysis, I will decide if any resampling techniques (oversampling,
undersampling) are necessary to ensure balanced training.
3. **Feature Analysis**: Continue with exploratory data analysis to understand the relationships between features and the target variable, which
can inform feature selection for modeling.

These steps will help ensure that the model developed is robust and performs well on unseen data.
```

Summary eda

```markdown
### Summary of Conclusions from EDA

1. **Data Loading and Structure**:
   - Successfully loaded the training datasets `X_train` and `y_train`, providing a foundation for analysis.
   - Inspected the structure of `X_train`, noting the presence of both numeric and categorical columns.

2. **Data Types Identification**:
   - Identified numeric columns (integers and floats) and categorical columns, which will guide preprocessing steps. Numeric columns include
   features like `person_age`, `loan_amnt`, and `loan_int_rate`, while categorical columns include `person_home_ownership`, `loan_grade`, and
   `loan_intent`.

3. **Handling Duplicates**:
   - Checked for duplicates in `X_train`, confirming that there were no duplicate rows, ensuring data integrity.

4. **Null Value Check**:
   - Found columns with null values: `person_emp_length` (738 nulls) and `loan_int_rate` (2501 nulls). This indicates a need for appropriate
   imputation or handling strategies for these columns.

5. **Distribution Analysis**:
   - Visualized the distribution of numerical features, allowing for insights into their characteristics. This step helps identify potential
   outliers or skewness that may require preprocessing adjustments.

6. **Target Variable Balance**:
   - Analyzed the balance of the target variable (`loan_status`), finding a representation of both classes (0 and 1). This distribution suggests a
   balanced dataset, but further exploration is needed to confirm no significant imbalance.

#### Next Steps
- Conduct further analysis on the distribution of features to inform preprocessing strategies.
- Evaluate class imbalance and consider resampling techniques if necessary.
- Proceed with feature engineering and selection based on insights gained during EDA.

This summary encapsulates the findings and outlines the direction for subsequent analysis and modeling efforts.
```

Create data_preprocessing



Import libs

```
import sys
sys.path.append('./src')
import utils
from copy import deepcopy
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import pandas as pd
```

✓ 1.1s

Drop_duplicate_data

```python
def drop_duplicate_data(X: pd.DataFrame, y: pd.Series) -> tuple:
    """
    Function to remove duplicate rows from a dataset.

    Parameters:
    X : pd.DataFrame
        The dataset from which to remove duplicate rows. Must be of type DataFrame.
    y : pd.Series
        The target data corresponding to the dataset X. Must be of type Series.

    Returns:
    tuple:
        A tuple containing:
        - X (pd.DataFrame): The dataset after duplicate rows have been dropped.
        - y (pd.Series): The target data aligned with the dataset after duplicates are removed.
    """
    if not isinstance(X, pd.DataFrame):
        raise TypeError("Parameter X must be of type DataFrame.")
    if not isinstance(y, pd.Series):
        raise TypeError("Parameter y must be of type Series.")

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

    X = X.copy()
    y = y.copy()

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

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

    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}.")

    X.drop_duplicates(inplace=True)
```

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

X = X.copy()
y = y.copy()

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

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

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}.")

X.drop_duplicates(inplace=True)

y = y[X.index]

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

return X, y
```

Executing drop_duplicates

```python
X_train, y_train = drop_duplicate_data(X_train, y_train)
```

✓ 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 (96, 11).
Fungsi drop_duplicate_data: shape dataset setelah drop duplicate seharusnya adalah (25968, 11).
Fungsi drop_duplicate_data: shape dataset setelah dropping duplicate adalah (25968, 11).
```

Function median

```python
def median_imputation(data: pd.DataFrame, subset_data, fit: bool) -> dict:
    """
    Impute missing values in specified columns of a DataFrame using the median.

    Parameters:
    - data: The DataFrame containing the data to be imputed.
    - subset_data:
        - If fit is True: a list of column names to calculate medians.
        - If fit is False: a dictionary where keys are column names and values are their median values.
    - fit: A boolean indicating whether to fit the model (True) or perform imputation (False).

    Returns:
    - If fit is True, returns a dictionary of median values for the specified columns.
    - If fit is False, returns the DataFrame with missing values imputed.
    """

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

    if fit:
        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 kolo
        imputation_data = {}
        for subset in subset_data:
            imputation_data[subset] = data[subset].median()

        print(f"Fungsi median_imputation: proses fitting telah selesai, berikut hasilnya {imputation_data}.")
        return imputation_data
```

```python
else:
    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 merupa

    print("Fungsi median_imputation: informasi count na sebelum dilakukan imputasi:")
    print(data.isna().sum())
    print()

    data.fillna(value=subset_data, inplace=True)

    print("Fungsi median_imputation: informasi count na setelah dilakukan imputasi:")
    print(data.isna().sum())
    print()

    return data
```

Execute median

```
Fungsi median_imputation: proses fitting telah selesai, berikut hasilnya {'person_emp_length': np.float64(4.0), 'loan_int_rate': np.float64(10.99)}.
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
...
10142                         3
28566                        10
```

Def OHE

```python
from utils import serialize_data
def create_onehot_encoder(categories: list, path: str) -> OneHotEncoder:
    """
    Create and fit a OneHotEncoder with the specified categories, then save it to the given path.

    Parameters:
    - categories: A list of categories for which the encoder will be created.
    - path: The location on the disk where the encoder will be saved.

    Returns:
    - ohe: The fitted OneHotEncoder instance.
    """

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

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

    ohe = OneHotEncoder(sparse_output=False)

    ohe.fit(np.array(categories).reshape(-1, 1))

    serialize_data(ohe, path)

    print(f"Kategori yang telah dipelajari adalah {ohe.categories_[0].tolist()}")

    return ohe
```

Execute ohe

```python
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']

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')
```

✓ 0.0s

```
Kategori yang telah dipelajari adalah ['MORTGAGE', 'OTHER', 'OWN', 'RENT']
Kategori yang telah dipelajari adalah ['DEBTCONSOLIDATION', 'EDUCATION', 'HOMEIMPROVEMENT', 'MEDICAL', 'PERSONAL', 'VENTURE']
Kategori yang telah dipelajari adalah ['A', 'B', 'C', 'D', 'E', 'F', 'G']
Kategori yang telah dipelajari adalah ['N', 'Y']
```

Def ohe_transform

```python
def ohe_transform(dataset: pd.DataFrame, subset: str, prefix: str, ohe: OneHotEncoder) -> pd.DataFrame:
    """
    Perform one-hot encoding on the specified subset of the dataset.

    Parameters:
    - dataset: The DataFrame containing the data to be encoded.
    - subset: The name of the column to be encoded.
    - prefix: The prefix to be added to the encoded columns.
    - ohe: The fitted OneHotEncoder instance.

    Returns:
    - dataset: The DataFrame with the encoded columns added and the original subset column dropped.
    """

    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:
        col_names = list(dataset.columns)
        _ = col_names.index(subset)
    except ValueError:
        raise RuntimeError("Fungsi ohe_transform: parameter subset string namun data tidak ditemukan dalam daftar kolom yang terdapat pada parameter d

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

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

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

dataset = dataset.copy()

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

col_names = [f"{prefix}_{col_name}" for col_name in ohe.categories_[0].tolist()]

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

dataset = pd.concat([dataset, encoded], axis=1)

dataset.drop(columns=[subset], inplace=True)

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

return dataset
```

Execute ohe transform

```python
X_train = ohe_transform(X_train, "person_home_ownership", "home_ownership", ohe_home_ownership)
X_train = ohe_transform(X_train, "loan_intent", "loan_intent", ohe_loan_intent)
X_train = ohe_transform(X_train, "loan_grade", "loan_grade", ohe_loan_grade)
X_train = ohe_transform(X_train, "cb_person_default_on_file", "default_onfile", ohe_default_on_file)

X_test = ohe_transform(X_test, "person_home_ownership", "home_ownership", ohe_home_ownership)
X_test = ohe_transform(X_test, "loan_intent", "loan_intent", ohe_loan_intent)
X_test = ohe_transform(X_test, "loan_grade", "loan_grade", ohe_loan_grade)
X_test = ohe_transform(X_test, "cb_person_default_on_file", "default_onfile", ohe_default_on_file)

X_valid = ohe_transform(X_valid, "person_home_ownership", "home_ownership", ohe_home_ownership)
X_valid = ohe_transform(X_valid, "loan_intent", "loan_intent", ohe_loan_intent)
X_valid = ohe_transform(X_valid, "loan_grade", "loan_grade", ohe_loan_grade)
X_valid = ohe_transform(X_valid, "cb_person_default_on_file", "default_onfile", ohe_default_on_file)
```

✓ 0.0s   Python

```
Fungsi ohe_transform: parameter telah divalidasi.
Fungsi ohe_transform: daftar nama kolom sebelum dilakukan pengkodean adalah ['person_age', 'person_income', 'person_home_ownership', 'person_emp_length',
Fungsi ohe_transform: daftar nama kolom setelah dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_intent', 'loan_gra
Fungsi ohe_transform: parameter telah divalidasi.
Fungsi ohe_transform: daftar nama kolom sebelum dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_intent', 'loan_gra
Fungsi ohe_transform: daftar nama kolom setelah dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_grade', 'loan_amnt
Fungsi ohe_transform: parameter telah divalidasi.
Fungsi ohe_transform: daftar nama kolom sebelum dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_grade', 'loan_amnt
Fungsi ohe_transform: daftar nama kolom setelah dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_amnt', 'loan_int_r
Fungsi ohe_transform: parameter telah divalidasi.
Fungsi ohe_transform: daftar nama kolom sebelum dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_amnt', 'loan_int_r
Fungsi ohe_transform: daftar nama kolom setelah dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_amnt', 'loan_int_r
Fungsi ohe_transform: parameter telah divalidasi.
Fungsi ohe_transform: daftar nama kolom sebelum dilakukan pengkodean adalah ['person_age', 'person_income', 'person_home_ownership', 'person_emp_length',
Fungsi ohe_transform: daftar nama kolom setelah dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_intent', 'loan_gra
Fungsi ohe_transform: parameter telah divalidasi.
Fungsi ohe_transform: daftar nama kolom sebelum dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_intent', 'loan_gra
Fungsi ohe_transform: daftar nama kolom setelah dilakukan pengkodean adalah ['person_age', 'person_income', 'person_emp_length', 'loan_grade', 'loan_amnt
```

Serialize x data

```python
serialize_data(X_train, "data/processed/X_train_prep.pkl")

serialize_data(X_test, "data/processed/X_test_prep.pkl")

serialize_data(X_valid, "data/processed/X_valid_prep.pkl")
```

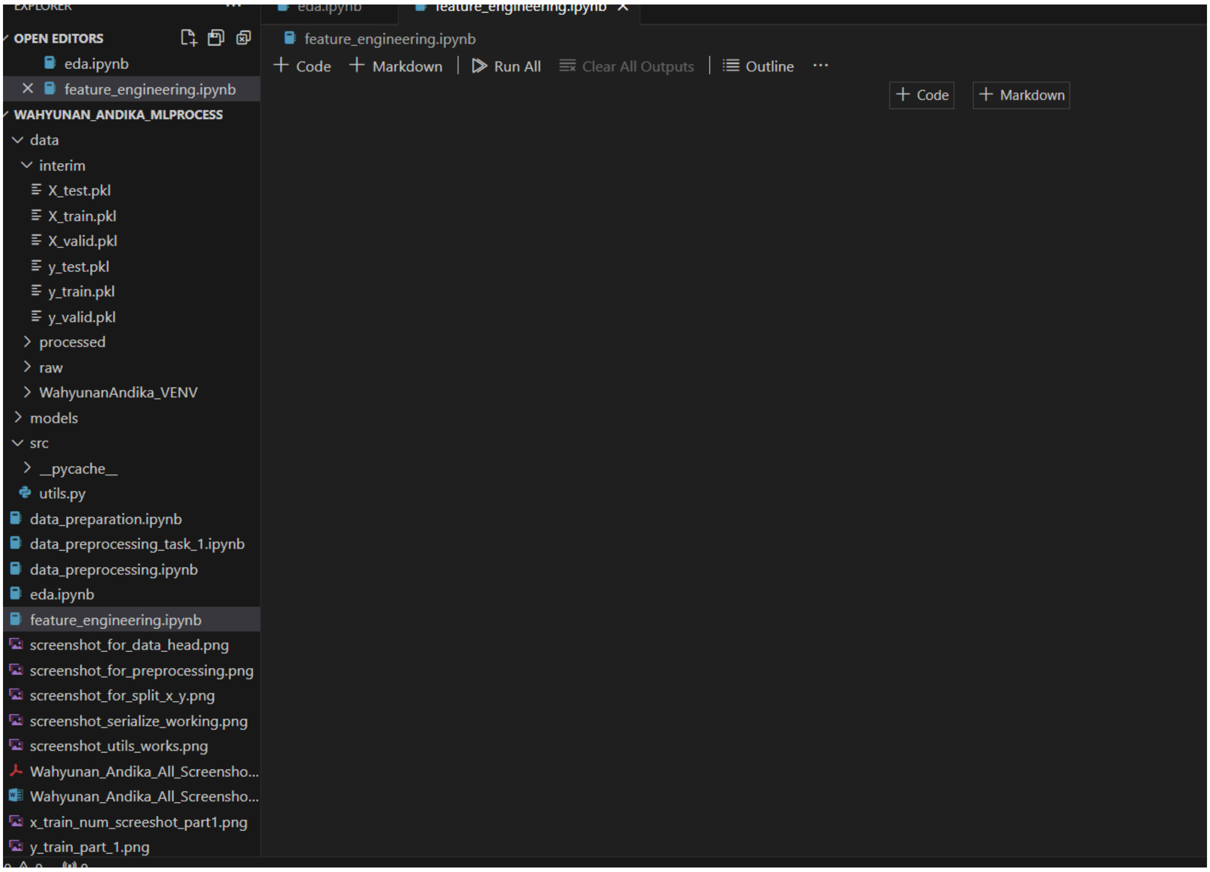
✓ 0.0s

Serialize y data

```python
serialize_data(y_train, "data/processed/y_train_prep.pkl")
```

✓ 0.0s

Feature engineering



Import

```python
import sys
sys.path.append('./src')
import utils
from imblearn.over_sampling import RandomOverSampler
```

✓ 0.0s

Execute deseriealize

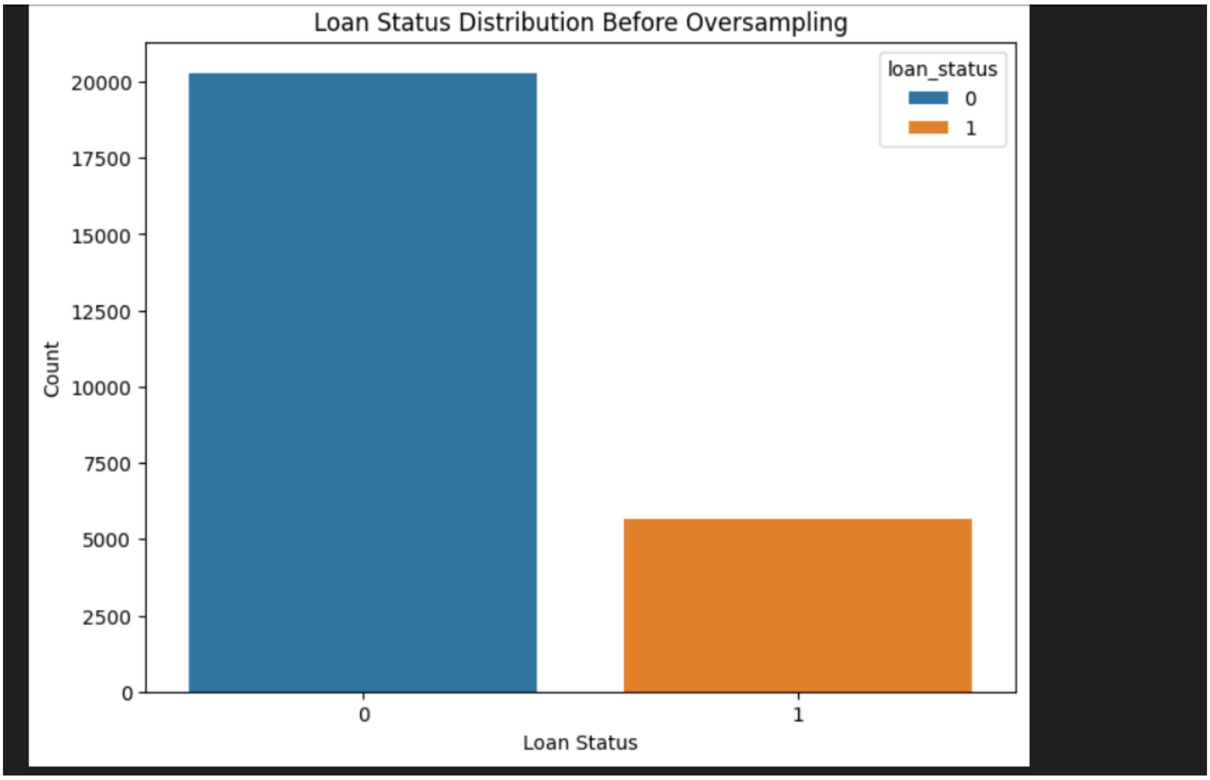
```python
from utils import deserialize_data

X_train_prep = deserialize_data('data/processed/X_train_prep.pkl')

y_train_prep = deserialize_data('data/processed/y_train_prep.pkl')

print(f'X_train_prep shape: {X_train_prep.shape}')
print(f'y_train_prep shape: {y_train_prep.shape}')
```
✓ 0.0s

```
X_train_prep shape: (25968, 26)
y_train_prep shape: (25968,)
```

Ros

```python
ros = RandomOverSampler(random_state=42)
```

Distribution



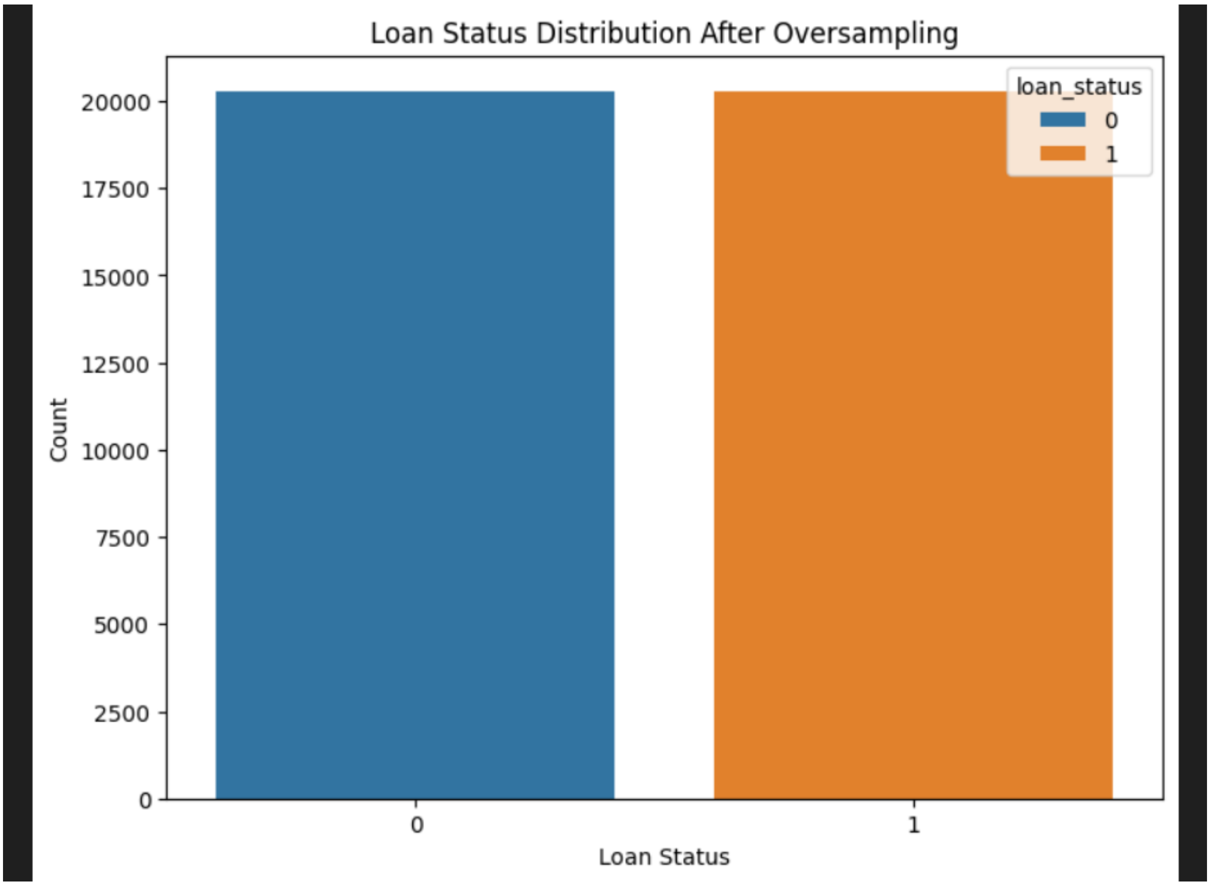
X_train_ros

```
X_train_ros, y_train_ros = ros.fit_resample(X_train_prep, y_train_prep)

print(f'Original X_train_prep shape: {X_train_prep.shape}')
print(f'Oversampled X_train_ros shape: {X_train_ros.shape}')
print(f'Original y_train_prep shape: {y_train_prep.shape}')
print(f'Oversampled y_train_ros shape: {y_train_ros.shape}')
```

✓ 0.1s

```
Original X_train_prep shape: (25968, 26)
Oversampled X_train_ros shape: (40582, 26)
Original y_train_prep shape: (25968,)
Oversampled y_train_ros shape: (40582,)
```

Result ros



Serialize

```python
from utils import serialize_data

serialize_data(X_train_ros, 'data/processed/X_train_ros.pkl')

serialize_data(y_train_ros, 'data/processed/y_train_ros.pkl')

print("Data successfully serialized and saved as .pkl files.")
```

✓ 0.1s

```
Data successfully serialized and saved as .pkl files.
```