

Nama : Az – Zahra Chikal E

NIM : 1103213039

Kelas : TK-45-05

LAPORAN UTS MACHINE LEARNING

LINK YOUTUBE :

- Regression Model : <https://youtu.be/SLsBdDHb0Aw>
- Classification Model : <https://youtu.be/JvwvCocO-ho>

A. REGRESSION MODEL

1. Load Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer
```

Pada code diatas digunakan untuk melakukan import library yang akan digunakan untuk melakukan progress regression dan classification. Pandas as pd merupakan Pustaka python untuk analisis data. Sedangkan Numpy untuk komputasi numerik. Kemudian untuk matplotlib.pyplot untuk membuat grafik. Lalu seaborn adalah Pustaka visualisasi berbasis matplotlib. Kemudian sklearn pada beberapa di gambar tersebut digunakan untuk membagi data menjadi subset untuk train, ada juga yang digunakan untuk model seperti decision tree regressor, knn regressor dan xgb regressor. Kemudian sklearn juga bisa digunakan untuk menghitung mean squared error dan r2 score.

```
[ ] df = pd.read_csv("RegresiUTSTelkom.csv")
df.head()
```

	2001	49.94357	31.47114	73.8775	6.74861	-17.48638	-13.89985	-25.81282	-12.23217	7.83889	...	13.41642	-14.48948	58.99367	15.37344	1.11144	-23.68793	68.48795	-1.83223	-27.46348	2.36337
0	2001	48.73215	18.42930	70.32679	12.94536	-10.32437	-24.83777	8.78630	-0.92019	18.70548	...	5.88812	-19.88073	33.04964	42.87836	-9.90378	-32.22788	70.49388	12.04941	58.43453	28.92061
1	2001	50.95714	31.85602	55.81851	13.41683	-4.57868	-18.54940	-3.27872	-2.35035	16.07017	...	3.03800	26.05866	-50.92779	10.93792	-0.07568	43.20130	-115.00698	-0.05859	39.67068	-0.66345
2	2001	48.24750	-1.89837	36.29772	2.58776	0.97170	-26.21883	5.05097	-10.34124	3.55005	...	34.57337	-171.70734	-16.98705	-46.67617	-12.51516	82.58061	-72.08993	9.90558	199.62971	18.85382
3	2001	50.97020	42.20998	67.09964	8.46791	-15.85279	-18.81409	-12.48207	-9.37636	12.63669	...	9.92861	-55.95724	64.92712	-17.72522	-1.48237	-7.50035	51.76631	7.88713	55.68926	28.74903
4	2001	50.54767	0.31568	92.35066	22.38696	-25.51870	-19.04928	20.67345	-5.19943	3.63566	...	6.59753	-50.89577	26.02574	18.94430	-0.33730	6.09352	35.18381	5.00283	-11.02257	0.02283

Pada code diatas digunakan untuk membaca file dataset yang akan digunakan. Untuk membaca dataset yang akan digunakan cukup masukan code `df = pd.read_csv("dataset yang akan dipakai.csv")`. dikarenakan dile ny berbentuk csv maka gunakan `.csv` , jika file nya berupa excel gunakan `.xlsx`. kemudian `df.head()` digunakan untuk menampilkan kolom baris teratas.

```
[ ] # Rename columns to "tahun", "x1", "x2", ..., "x90"
df.columns = ["Tahun"] + [f"X{i}" for i in range(1, 91)]
df.head()
```

	Tahun	x1	x2	x3	x4	x5	x6	x7	x8	x9	...	x81	x82	x83	x84	x85	x86	x87	x88	x89	x90
0	2001	48.73215	18.42930	70.32679	12.94536	-10.32437	-24.83777	8.78630	-0.92019	18.70548	...	5.88812	-19.88073	33.04964	42.87836	-9.90378	-32.22788	70.49388	12.04941	58.43453	28.92061
1	2001	50.95714	31.85602	55.81851	13.41683	-4.57868	-18.54940	-3.27872	-2.35035	16.07017	...	3.03800	26.05866	-50.92779	10.93792	-0.07568	43.20130	-115.00698	-0.05859	39.67068	-0.66345
2	2001	48.24750	-1.89837	36.29772	2.58776	0.97170	-26.21883	5.05097	-10.34124	3.55005	...	34.57337	-171.70734	-16.98705	-46.67617	-12.51516	82.58061	-72.08993	9.90558	199.62971	18.85382
3	2001	50.97020	42.20998	67.09964	8.46791	-15.85279	-18.81409	-12.48207	-9.37636	12.63669	...	9.92861	-55.95724	64.92712	-17.72522	-1.48237	-7.50035	51.76631	7.88713	55.68926	28.74903
4	2001	50.54767	0.31568	92.35066	22.38696	-25.51870	-19.04928	20.67345	-5.19943	3.63566	...	6.59753	-50.89577	26.02574	18.94430	-0.33730	6.09352	35.18381	5.00283	-11.02257	0.02283

Pada code diatas digunakan untuk mengganti nama kolom pada DataFrame df menjadi format yang lebih terstruktur. Nama kolom pertama diubah menjadi "Tahun", yang diasumsikan mewakili data tahun, sementara 90 kolom berikutnya diberi nama secara berurutan dengan format "X1", "X2", hingga "X90". Untuk membuat nama kolom tersebut, digunakan kombinasi sebuah daftar tunggal berisi "Tahun" dengan daftar hasil list comprehension yang menghasilkan nama kolom seperti "X1", "X2", dan seterusnya.

```
[ ] df.info()
```

<class 'pandas.core.frame.DataFrame'> RangeIndex: 515344 entries, 0 to 515344 Data columns (total 91 columns): # Column Non-Null Count Dtype --- 0 Tahun 515344 non-null int64 1 X1 515344 non-null float64 2 X2 515344 non-null float64 3 X3 515344 non-null float64 4 X4 515344 non-null float64 5 X5 515344 non-null float64 6 X6 515344 non-null float64 7 X7 515344 non-null float64 8 X8 515344 non-null float64 9 X9 515344 non-null float64 10 X10 515344 non-null float64 11 X11 515344 non-null float64 12 X12 515344 non-null float64 13 X13 515344 non-null float64 14 X14 515344 non-null float64 15 X15 515344 non-null float64 16 X16 515344 non-null float64 17 X17 515344 non-null float64 18 X18 515344 non-null float64 19 X19 515344 non-null float64 20 X20 515344 non-null float64 21 X21 515344 non-null float64 22 X22 515344 non-null float64 23 X23 515344 non-null float64 24 X24 515344 non-null float64 25 X25 515344 non-null float64 26 X26 515344 non-null float64	27 X27 515344 non-null float64 28 X28 515344 non-null float64 29 X29 515344 non-null float64 30 X30 515344 non-null float64 31 X31 515344 non-null float64 32 X32 515344 non-null float64 33 X33 515344 non-null float64 34 X34 515344 non-null float64 35 X35 515344 non-null float64 36 X36 515344 non-null float64 37 X37 515344 non-null float64 38 X38 515344 non-null float64 39 X39 515344 non-null float64 40 X40 515344 non-null float64 41 X41 515344 non-null float64 42 X42 515344 non-null float64 43 X43 515344 non-null float64 44 X44 515344 non-null float64 45 X45 515344 non-null float64 46 X46 515344 non-null float64 47 X47 515344 non-null float64 48 X48 515344 non-null float64 49 X49 515344 non-null float64 50 X50 515344 non-null float64 51 X51 515344 non-null float64 52 X52 515344 non-null float64 53 X53 515344 non-null float64 54 X54 515344 non-null float64 55 X55 515344 non-null float64 56 X56 515344 non-null float64 57 X57 515344 non-null float64	58 X58 515344 non-null float64 59 X59 515344 non-null float64 60 X60 515344 non-null float64 61 X61 515344 non-null float64 62 X62 515344 non-null float64 63 X63 515344 non-null float64 64 X64 515344 non-null float64 65 X65 515344 non-null float64 66 X66 515344 non-null float64 67 X67 515344 non-null float64 68 X68 515344 non-null float64 69 X69 515344 non-null float64 70 X70 515344 non-null float64 71 X71 515344 non-null float64 72 X72 515344 non-null float64 73 X73 515344 non-null float64 74 X74 515344 non-null float64 75 X75 515344 non-null float64 76 X76 515344 non-null float64 77 X77 515344 non-null float64 78 X78 515344 non-null float64 79 X79 515344 non-null float64 80 X80 515344 non-null float64 81 X81 515344 non-null float64 82 X82 515344 non-null float64 83 X83 515344 non-null float64 84 X84 515344 non-null float64 85 X85 515344 non-null float64 86 X86 515344 non-null float64 87 X87 515344 non-null float64 88 X88 515344 non-null float64 89 X89 515344 non-null float64 90 X90 515344 non-null float64 dtypes: float64(90), int64(1) memory usage: 357.8 MB
---	--	---

Pada code diatas digunakan untuk mengetahui informasi dataset oleh karena itu code yang digunakan adalah `df.info()` dengan code tersebut dapat diketahui terdapat 515344 dataset dengann 91 kolom

```
[ ] # Sampling data (contoh: 1% dari total data)
    sampled_df = df.sample(frac=0.1, random_state=42)

# Menampilkan informasi dataset setelah sampling
print(f"Jumlah data setelah sampling: {len(sampled_df)}")
print(sampled_df.info())
```

```
➡ Jumlah data setelah sampling: 51534
<class 'pandas.core.frame.DataFrame'>
Index: 51534 entries, 201297 to 102108
Data columns (total 91 columns):
```

Pada code diatas digunakan untuk melakukan pengambilan sampel data sebesar 10% dari total data di DataFrame df menggunakan fungsi sample() dari Pandas. Parameter frac=0.1 menunjukkan bahwa ukuran sampel adalah 10% dari jumlah total data, sementara random_state=42 memastikan bahwa proses sampling bersifat deterministik, artinya hasilnya akan selalu sama jika kode dijalankan ulang, karena nilai acak yang digunakan diatur dengan angka tertentu. DataFrame hasil sampel disimpan ke dalam variabel sampled_df.

2. EDA

```
[ ] # Statistik deskriptif
    print(sampled_df.describe())
```

```
➡
```

	Tahun	X1	X2	X3	X4
count	51534.000000	51534.000000	51534.000000	51534.000000	51534.000000
mean	1998.463286	43.381234	1.116871	8.630126	1.153814
std	10.841441	6.096472	51.990030	35.316084	16.230094
min	1922.000000	4.836880	-305.422000	-245.390230	-119.573670
25%	1994.000000	39.941252	-26.215203	-11.522355	-8.466527
50%	2002.000000	44.264790	8.598380	10.463270	-0.654040
75%	2006.000000	47.866132	36.043190	29.765515	8.748738
max	2010.000000	61.138540	303.977370	322.851430	122.987250

	X5	X6	X7	X8	X9
count	51534.000000	51534.000000	51534.000000	51534.000000	51534.000000
mean	-6.456766	-9.471798	-2.367160	-1.767880	3.650041
std	22.919380	12.826267	14.556213	7.990899	10.606833
min	-158.917880	-70.872270	-97.704900	-55.461240	-97.016570
25%	-20.545362	-18.385047	-10.778025	-6.448095	-2.340263
50%	-5.962905	-11.094560	-2.095320	-1.667395	3.825960
75%	7.912790	-2.328865	6.564875	2.974860	9.908535
max	204.414630	81.401690	172.402680	57.682080	85.522510

Pada code diatas digunakan untuk menghitung dan menampilkan **statistik deskriptif** dari DataFrame sampled_df, yaitu data hasil pengambilan sampel. Fungsi describe() dari Pandas secara otomatis menghitung ringkasan statistik untuk semua kolom numerik dalam DataFrame.

```
[ ] # Cek missing values
print(sampled_df.isnull().sum())
```

```
Tahun      0
X1          0
X2          0
X3          0
X4          0
..
X86         0
X87         0
X88         0
X89         0
X90         0
Length: 91, dtype: int64
```

Pada code diatas digunakan untuk melakukan pengambilan sampel data sebesar 10% dari total data di DataFrame df menggunakan fungsi sample() dari Pandas. Parameter frac=0.1 menunjukkan bahwa ukuran sampel adalah 10% dari jumlah total data, sementara random_state=42 memastikan bahwa proses sampling bersifat deterministik, artinya hasilnya akan selalu sama jika kode dijalankan ulang, karena nilai acak yang digunakan diatur dengan angka tertentu. DataFrame hasil sampel disimpan ke dalam variabel sampled_df.

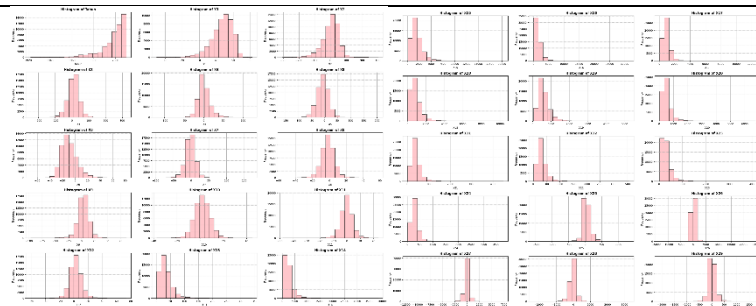
```
[ ] # Menentukan jumlah kolom dan baris
n_features = sampled_df.shape[1]
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols

# Membuat subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 100)) # Ukuran diperbesar
axes = axes.flatten()

# Plot histogram untuk setiap fitur
for i, col in enumerate(sampled_df.columns):
    sampled_df[col].hist(ax=axes[i], bins=20, alpha=0.75, color='#FFB3BA', edgecolor='black')
    axes[i].set_title(f'Histogram of {col}', fontsize=12, fontweight='bold')
    axes[i].set_xlabel(col, fontsize=10)
    axes[i].set_ylabel('Frequency', fontsize=10)
    axes[i].grid(axis='y', linestyle='--', alpha=0.7)

# Menyembunyikan subplot kosong (jika ada)
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Menyempurnakan tata letak
plt.tight_layout()
plt.subplots_adjust(top=0.98) # Memberikan margin tambahan di atas
plt.show()
```



Pada code diatas digunakan untuk membuat histogram dari semua kolom dalam DataFrame sampled_df dan menampilkannya dalam bentuk grid yang rapi. Jumlah baris dan kolom grid dihitung berdasarkan jumlah fitur dalam DataFrame. Setiap

histogram diberi judul, label sumbu, dan garis bantu untuk memudahkan pembacaan. Subplot kosong (jika ada) disembunyikan, dan tata letak disesuaikan agar grafik tidak tumpang tindih. Akhirnya, histogram ditampilkan dengan menggunakan `plt.show()`, memberikan visualisasi distribusi data setiap fitur.

```
[ ] # Menghitung matriks korelasi
correlation_matrix = sampled_df.corr()

# Menghitung korelasi absolut antar fitur
correlation_abs = correlation_matrix.abs()

# Menjumlahkan korelasi absolut untuk setiap fitur
correlation_sums = correlation_matrix.abs().sum().sort_values(ascending=False)

# Menampilkan fitur dengan korelasi tertinggi terhadap yang lain
print("Fitur dengan korelasi total tertinggi terhadap fitur lain:")
print(correlation_sums)

# Memilih target terbaik
best_target = correlation_sums.index[1] # Index 0 adalah 'Tahun' atau target asli
print(f"Fitur terbaik untuk dijadikan target: {best_target}")
```

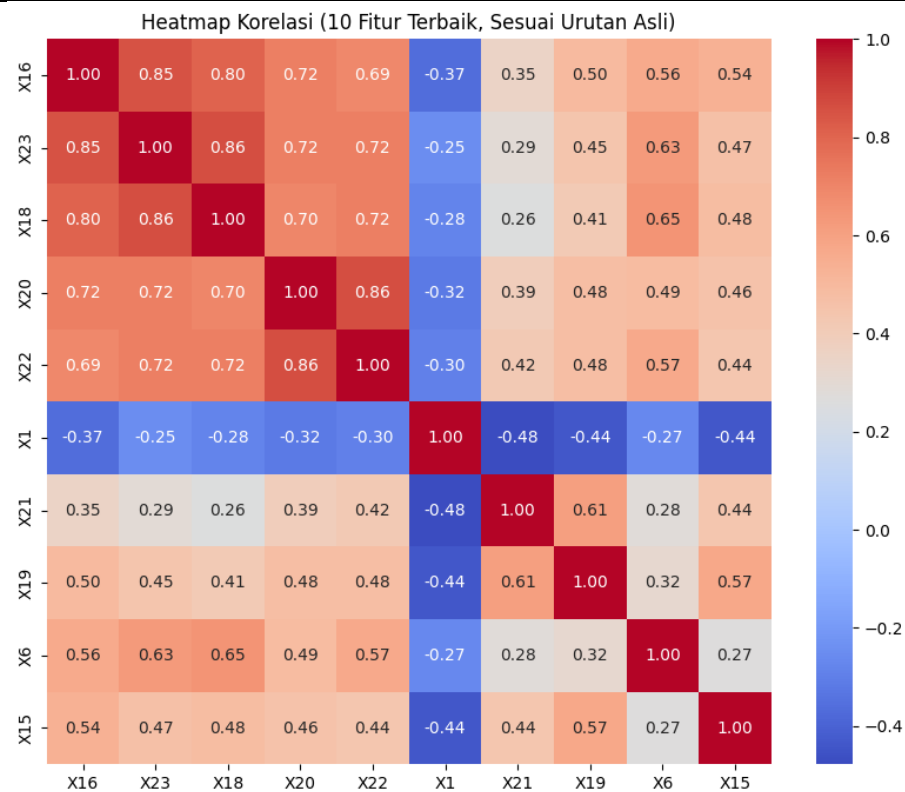
```
Fitur dengan korelasi total tertinggi terhadap fitur lain
X16      20.796226
X23      20.369078
X18      19.563088
X20      18.517183
X22      18.337466
...
X89       6.200842
X74       6.189592
X12       5.804907
Tahun     5.606775
X87       5.208901
Length: 91, dtype: float64
Fitur terbaik untuk dijadikan target: X23
```

Pada code diatas digunakan untuk menghitung dan menganalisis korelasi antar fitur dalam DataFrame `sampled_df`. Pertama, matriks korelasi dihitung dengan menggunakan fungsi `.corr()`, yang mengukur kekuatan dan arah hubungan linier antara setiap pasangan fitur. Kemudian, korelasi absolut dihitung dengan mengambil nilai mutlak dari matriks korelasi, yang memungkinkan untuk melihat seberapa kuat hubungan antar fitur tanpa memperhatikan arah korelasinya (positif atau negatif). Selanjutnya, jumlah korelasi absolut untuk setiap fitur dihitung dan diurutkan berdasarkan nilai korelasi tertinggi menggunakan `.sum()` dan `.sort_values()`. Fitur dengan jumlah korelasi tertinggi terhadap fitur lain ditampilkan, yang menunjukkan fitur yang memiliki hubungan paling kuat dengan fitur lainnya.

```
[ ] # Memilih 10 fitur terbaik berdasarkan korelasi tertinggi
top_10_features = correlation_sums.index[:10]

# Membuat subset matriks korelasi untuk fitur-fitur tersebut
top_10_corr_matrix = correlation_matrix.loc[top_10_features, top_10_features]

# Membuat heatmap
plt.figure(figsize=(10, 8))
# Use top_10_corr_matrix instead of sorted_corr_matrix
sns.heatmap(top_10_corr_matrix, annot=True, cmap="coolwarm", fmt=".2f", square=True)
plt.title("Heatmap Korelasi (10 Fitur Terbaik, Sesuai Urutan Asli)")
plt.show()
```



Pada code diatas digunakan untuk memilih dan memvisualisasikan 10 fitur dengan korelasi tertinggi dalam DataFrame `sampled_df`. Pertama, 10 fitur dengan jumlah korelasi tertinggi dihitung dan dipilih dengan mengambil 10 indeks teratas dari `correlation_sums`. Kemudian, sebuah subset matriks korelasi dibuat hanya untuk fitur-fitur terpilih menggunakan `.loc[]`, yang menghasilkan matriks korelasi yang lebih kecil antara 10 fitur terbaik. Selanjutnya, sebuah heatmap dibuat untuk menampilkan matriks korelasi ini menggunakan `sns.heatmap()`. Visualisasi ini memungkinkan pengguna untuk melihat hubungan antar fitur secara lebih jelas, dengan angka korelasi ditampilkan dalam setiap sel dan warna yang menunjukkan kekuatan korelasi (merah untuk korelasi positif tinggi dan biru untuk negatif).

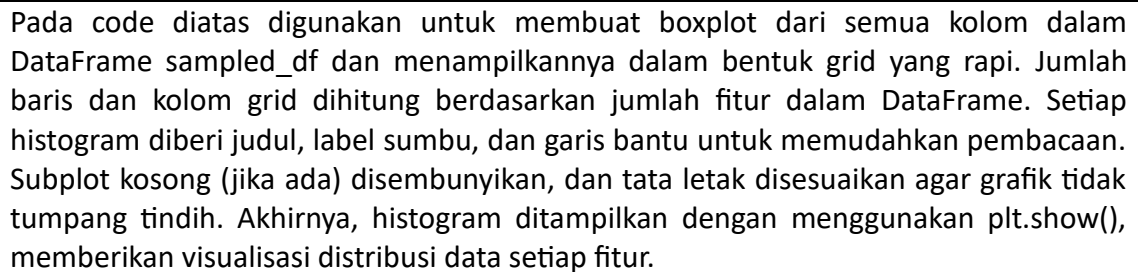
```
[ ] # Menentukan jumlah kolom dan baris
n_features = sampled_df.shape[1]
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols

# Membuat subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 100)) # Ukuran diperbesar
axes = axes.flatten()

# Plot boxplot untuk setiap fitur
for i, col in enumerate(sampled_df.columns):
    axes[i].boxplot(sampled_df[col], patch_artist=True, boxprops=dict(facecolor='#FFB3BA', color='black'),
                    medianprops=dict(color='black'), whiskerprops=dict(color='black'), capprops=dict(color='black'))
    axes[i].set_title(f'Boxplot of {col}', fontsize=12, fontweight='bold')
    axes[i].set_ylabel(col, fontsize=10)

# Menyembunyikan subplot kosong (jika ada)
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Menyempurnakan tata letak
plt.tight_layout()
plt.subplots_adjust(top=0.98) # Memberikan margin tambahan di atas
plt.show()
```



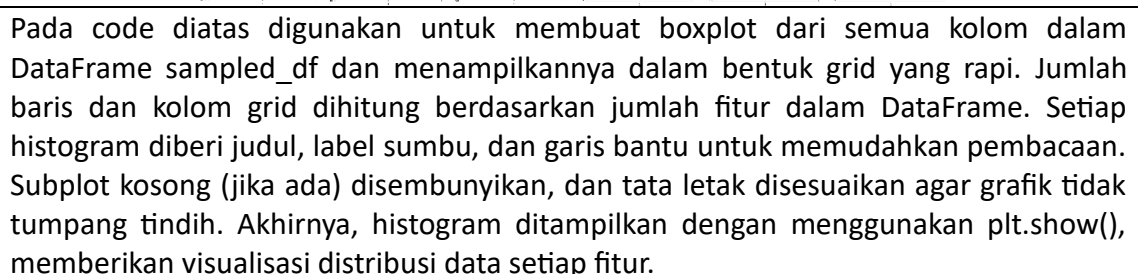
```
[ ] # Menentukan jumlah kolom dan baris
n_features = sampled_df.shape[1]
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols

# Membuat subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 100)) # Ukuran diperbesar
axes = axes.flatten()

# Plot boxplot untuk setiap fitur
for i, col in enumerate(sampled_df.columns):
    axes[i].boxplot(sampled_df[col], patch_artist=True, boxprops=dict(facecolor='#FFB3BA', color='black'),
                    medianprops=dict(color='black'), whiskerprops=dict(color='black'), capprops=dict(color='black'))
    axes[i].set_title(f'Boxplot of {col}', fontsize=12, fontweight='bold')
    axes[i].set_ylabel(col, fontsize=10)

# Menyembunyikan subplot kosong (jika ada)
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Menyempurnakan tata letak
plt.tight_layout()
plt.subplots_adjust(top=0.98) # Memberikan margin tambahan di atas
plt.show()
```



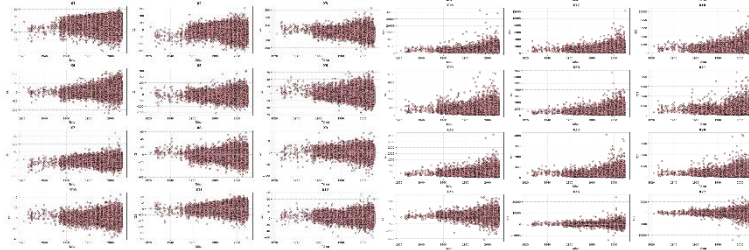
```
[ ] # Menentukan jumlah kolom dan baris
n_features = sampled_df.shape[1] - 1 # Fitur selain kolom pertama
n_cols = 3
n_rows = (n_features + n_cols - 1) // n_cols

# Membuat subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(20, 100)) # Ukuran diperbesar
axes = axes.flatten()

# Plot scatterplot untuk setiap fitur dibandingkan dengan kolom pertama
for i, col in enumerate(sampled_df.columns[1:]): # Mulai dari kolom kedua
    axes[i].scatter(sampled_df['Tahun'], sampled_df[col], alpha=0.6, color='#FFB3BA', edgecolor='black')
    axes[i].set_title(f'{col}', fontsize=12, fontweight='bold')
    axes[i].set_xlabel('Tahun', fontsize=10)
    axes[i].set_ylabel(col, fontsize=10)
    axes[i].grid(alpha=0.7, linestyle='--')

# Menyembunyikan subplot kosong (jika ada)
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Menyempurnakan tata letak
plt.tight_layout()
plt.subplots_adjust(top=0.98) # Memberikan margin tambahan di atas
plt.show()
```



Pada code diatas digunakan untuk membuat scatter plot dari semua kolom dalam DataFrame sampled_df dan menampilkannya dalam bentuk grid yang rapi. Jumlah baris dan kolom grid dihitung berdasarkan jumlah fitur dalam DataFrame. Setiap histogram diberi judul, label sumbu, dan garis bantu untuk memudahkan pembacaan. Subplot kosong (jika ada) disembunyikan, dan tata letak disesuaikan agar grafik tidak tumpang tindih. Akhirnya, histogram ditampilkan dengan menggunakan plt.show(), memberikan visualisasi distribusi data setiap fitur.

3. PIPELINE

```
[ ] # Memisahkan fitur dan target
X = sampled_df.drop(columns=['Tahun'])
y = sampled_df['Tahun']

# Membagi dataset menjadi training dan testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Pada code ini memisahkan dataset menjadi fitur (*features*) dan target, serta membaginya ke dalam data pelatihan dan data pengujian. Fitur (X) diperoleh dengan menghapus kolom "Tahun" dari dataset, sementara kolom "Tahun" itu sendiri digunakan sebagai target (y). Setelah itu, dataset dibagi menjadi dua bagian menggunakan fungsi train_test_split(), di mana 80% data digunakan sebagai data pelatihan dan 20% sisanya sebagai data pengujian, yang ditentukan oleh parameter test_size=0.2. Pengaturan parameter random_state=42 memastikan pembagian dataset dilakukan secara deterministik, sehingga hasil pembagian akan selalu konsisten setiap kali kode dijalankan.


```
[ ] # Pipeline untuk Polynomial Regression
poly_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')), # Add an Imputer step
    ('poly_features', PolynomialFeatures(degree=2, include_bias=False)),
    ('scaler', StandardScaler()),
    ('regressor', DecisionTreeRegressor(random_state=42))
])

# Training dan evaluasi
poly_pipeline.fit(X_train, y_train)
y_pred_poly = poly_pipeline.predict(X_test)

print("Polynomial Regression Results:")
print("MSE:", mean_squared_error(y_test, y_pred_poly))
print("R2 Score:", r2_score(y_test, y_pred_poly))
```



Polynomial Regression Results:

MSE: 189.54943242456582

R2 Score: -0.6384769501643688

Pada code ini membuat *pipeline* untuk regresi polinomial menggunakan *Polynomial Features* dan *Decision Tree Regressor*. Pipeline mencakup langkah-langkah: imputasi nilai hilang dengan rata-rata, transformasi fitur menjadi derajat polinomial ke-2, standarisasi fitur, dan prediksi menggunakan *Decision Tree Regressor*. Pipeline dilatih dengan data pelatihan dan diuji dengan data pengujian untuk menghasilkan prediksi. Hasil evaluasi ditampilkan menggunakan *Mean Squared Error* (MSE) untuk mengukur kesalahan rata-rata kuadrat dan *R2 Score* untuk menilai kemampuan model menjelaskan variabilitas data.

```
[ ] # Pipeline untuk Decision Tree Regression
dt_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', DecisionTreeRegressor(random_state=42, max_depth=10))
])

# Training dan evaluasi
dt_pipeline.fit(X_train, y_train)
y_pred_dt = dt_pipeline.predict(X_test)

print("Decision Tree Regression Results:")
print("MSE:", mean_squared_error(y_test, y_pred_dt))
print("R2 Score:", r2_score(y_test, y_pred_dt))
```




Decision Tree Regression Results:


MSE: 113.43825474621066

R2 Score: 0.01943275012174861

Pada code ini membangun *pipeline* untuk regresi menggunakan *Decision Tree Regressor*. Pipeline terdiri dari dua langkah: standarisasi fitur menggunakan *StandardScaler* untuk memastikan skala yang seragam, dan model *Decision Tree Regressor* dengan kedalaman maksimum 10 serta parameter acak tetap. Pipeline dilatih menggunakan data pelatihan, lalu memprediksi target pada data pengujian. Hasil evaluasi mencakup *Mean Squared Error* (MSE) untuk mengukur kesalahan rata-rata kuadrat dan *R2 Score* untuk menilai kemampuan model menjelaskan variabilitas target.

	<pre>[] # Pipeline untuk XGBoost Regression xgb_pipeline = Pipeline([('scaler', StandardScaler()), ('regressor', XGBRegressor(objective='reg:squarederror', random_state=42))]) # Training dan evaluasi xgb_pipeline.fit(X_train, y_train) y_pred_xgb = xgb_pipeline.predict(X_test) print("XGBoost Regression Results:") print("MSE:", mean_squared_error(y_test, y_pred_xgb)) print("R2 Score:", r2_score(y_test, y_pred_xgb))</pre>	
	 XGBoost Regression Results: MSE: 86.50281784705658 R2 Score: 0.2522643208503723	
<p>Pada code ini membangun *pipeline* untuk regresi menggunakan *XGBoost Regressor*. Pipeline terdiri dari dua langkah: standarisasi fitur menggunakan *StandardScaler* dan penerapan model *XGBoost Regressor* dengan tujuan prediksi regresi kuadrat kesalahan (*reg:squarederror*). Pipeline dilatih pada data pelatihan, kemudian digunakan untuk memprediksi target pada data pengujian. Evaluasi dilakukan dengan menghitung *Mean Squared Error* (MSE) untuk mengukur kesalahan rata-rata kuadrat dan *R2 Score* untuk menilai kemampuan model dalam menjelaskan variabilitas target.</p>		

4. Hyperparameter Tuning

	<pre>[] from sklearn.model_selection import GridSearchCV # --- Hyperparameter Tuning untuk Decision Tree Regression --- dt_pipeline = Pipeline([('scaler', StandardScaler()), ('regressor', DecisionTreeRegressor(random_state=42))]) # Hyperparameter grid untuk DecisionTreeRegressor param_grid_dt = { 'regressor__max_depth': [5, 10, 15, None], 'regressor__min_samples_split': [2, 5, 10], 'regressor__min_samples_leaf': [1, 2, 4] } grid_search_dt = GridSearchCV(dt_pipeline, param_grid_dt, cv=5, scoring='neg_mean_squared_error', n_jobs=-1, verbose=2) grid_search_dt.fit(X_train, y_train) best_dt_model = grid_search_dt.best_estimator_ # Prediksi dan evaluasi model terbaik y_pred_dt = best_dt_model.predict(X_test) print("Decision Tree Regression (Tuning) Results:") print("Best Parameters:", grid_search_dt.best_params_) print("MSE:", mean_squared_error(y_test, y_pred_dt)) print("R2 Score:", r2_score(y_test, y_pred_dt))</pre>	
	 Fitting 5 folds for each of 36 candidates, totalling 180 fits Decision Tree Regression (Tuning) Results: Best Parameters: {'regressor__max_depth': 5, 'regressor__min_samples_leaf': 4, 'regressor__min_samples_split': 10} MSE: 99.95353835354904 R2 Score: 0.13599546785857175	
<p>Pada Kode ini melakukan hyperparameter tuning pada model Decision Tree Regressor menggunakan GridSearchCV. Pipeline dibuat dengan langkah standarisasi dan model regresi. Grid parameter didefinisikan untuk mencari kombinasi terbaik dari kedalaman pohon (max_depth), jumlah minimum sampel untuk pemisahan (min_samples_split), dan jumlah minimum sampel per daun (min_samples_leaf). GridSearchCV menggunakan 5-fold cross-validation dengan metrik neg_mean_squared_error. Model</p>		

terbaik dilatih pada data pelatihan, lalu dievaluasi pada data pengujian. Hasil meliputi parameter terbaik, Mean Squared Error (MSE), dan R2 Score.

```
[ ] # --- Hyperparameter Tuning untuk XGBoost Regression ---
xgb_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('regressor', XGBRegressor(objective='reg:squarederror', random_state=42))
])

# Hyperparameter grid untuk XGBRegressor
param_grid_xgb = {
    'regressor__max_depth': [3, 5, 7],
    'regressor__learning_rate': [0.01, 0.05, 0.1],
    'regressor__n_estimators': [50, 100, 200],
    'regressor__subsample': [0.8, 1.0]
}

grid_search_xgb = GridSearchCV(xgb_pipeline, param_grid_xgb, cv=5, scoring='neg_mean_squared_error', n_jobs=-1, verbose=2)
grid_search_xgb.fit(X_train, y_train)
best_xgb_model = grid_search_xgb.best_estimator_

# Prediksi dan evaluasi model terbaik
y_pred_xgb = best_xgb_model.predict(X_test)
print("XGBoost Regression (Tuning) Results:")
print("Best Parameters:", grid_search_xgb.best_params_)
print("MSE:", mean_squared_error(y_test, y_pred_xgb))
print("R2 Score:", r2_score(y_test, y_pred_xgb))

Fitting 5 folds for each of 54 candidates, totalling 270 fits
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker stopped while some jobs were given to it. This could be caused by a long running python process or by jobs that never finished.
warnings.warn(
XGBoost Regression (Tuning) Results:
Best Parameters: {'regressor__learning_rate': 0.05, 'regressor__max_depth': 7, 'regressor__n_estimators': 200, 'regressor__subsample': 0.8}
MSE: 80.24166173329088
R2 Score: 0.30638617277145386
```

Pada Kode ini melakukan hyperparameter tuning pada model XGBoost Regressor menggunakan GridSearchCV. Pipeline dibuat dengan langkah standarisasi dan model regresi XGBoost. Grid parameter meliputi kedalaman pohon (max_depth), laju pembelajaran (learning_rate), jumlah pohon (n_estimators), dan proporsi data yang digunakan (subsample). GridSearchCV menggunakan 5-fold cross-validation dengan metrik neg_mean_squared_error. Setelah menemukan parameter terbaik, model dilatih dan digunakan untuk memprediksi data pengujian. Evaluasi hasil mencakup parameter terbaik, Mean Squared Error (MSE), dan R2 Score.

```
[ ] # Hyperparameter grid
param_grid_poly = {
    'poly_features_degree': [2], # Degree polinomial
    'regressor_max_depth': [5, 10], # Kedalaman maksimum untuk DecisionTreeRegressor
    'regressor_min_samples_split': [2, 5] # Minimum sampel untuk split
}

# Grid Search CV untuk mencari parameter terbaik
grid_search_poly = GridSearchCV(
    poly_pipeline,
    param_grid_poly,
    cv=3,
    scoring='neg_mean_squared_error',
    n_jobs=1, # Gunakan 1 core untuk menghindari konflik memori
    verbose=2
)

# Training model
grid_search_poly.fit(X_train, y_train)

# Model terbaik
best_poly_model = grid_search_poly.best_estimator_

# Prediksi dengan model terbaik
y_pred_poly = best_poly_model.predict(X_test)

# --- Evaluasi ---
# Menghitung metrik evaluasi
mse = mean_squared_error(y_test, y_pred_poly)
r2 = r2_score(y_test, y_pred_poly)

# Menampilkan hasil
print("=" * 50)
print("Polynomial Regression (Tuning) Results:")
print(f"Best Parameters: {grid_search_poly.best_params_}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R2 Score (R2): {r2:.4f}")
print("=" * 50)
```

```

Fitting 3 folds for each of 4 candidates, totalling 12 fits
[CV] END poly_features_degree=2, regressor_max_depth=5, regressor_min_samples_split=2; total time= 1.9min
[CV] END poly_features_degree=2, regressor_max_depth=5, regressor_min_samples_split=2; total time= 1.9min
[CV] END poly_features_degree=2, regressor_max_depth=5, regressor_min_samples_split=2; total time= 1.9min
[CV] END poly_features_degree=2, regressor_max_depth=5, regressor_min_samples_split=5; total time= 1.9min
[CV] END poly_features_degree=2, regressor_max_depth=5, regressor_min_samples_split=5; total time= 1.9min
[CV] END poly_features_degree=2, regressor_max_depth=5, regressor_min_samples_split=5; total time= 2.0min
[CV] END poly_features_degree=2, regressor_max_depth=10, regressor_min_samples_split=2; total time= 3.5min
[CV] END poly_features_degree=2, regressor_max_depth=10, regressor_min_samples_split=2; total time= 3.4min
[CV] END poly_features_degree=2, regressor_max_depth=10, regressor_min_samples_split=5; total time= 3.3min
[CV] END poly_features_degree=2, regressor_max_depth=10, regressor_min_samples_split=5; total time= 3.4min
[CV] END poly_features_degree=2, regressor_max_depth=10, regressor_min_samples_split=5; total time= 3.4min
[CV] END poly_features_degree=2, regressor_max_depth=10, regressor_min_samples_split=5; total time= 3.4min
=====
Polynomial Regression (Tuning) Results:
Best Parameters: {'poly_features_degree': 2, 'regressor_max_depth': 5, 'regressor_min_samples_split': 5}
Mean Squared Error (MSE): 98.6960
R2 Score (R2): 0.1469
=====

```

Pada Kode ini melakukan hyperparameter tuning pada regresi polinomial dengan GridSearchCV. Pipeline menggunakan Polynomial Features (derajat 2), StandardScaler, dan Decision Tree Regressor. Parameter yang diuji meliputi derajat polinomial, kedalaman maksimum pohon (max_depth), dan jumlah minimum sampel untuk pemisahan (min_samples_split). GridSearchCV menggunakan 3-fold cross-validation dan metrik neg_mean_squared_error. Model terbaik dilatih dengan parameter optimal dan digunakan untuk memprediksi data pengujian. Evaluasi dilakukan dengan menghitung Mean Squared Error (MSE) dan R² Score untuk mengukur performa model. Hasil tuning termasuk parameter terbaik, MSE, dan nilai R².

B. CLASSIFICATION MODEL

1. Load Dataset

```
[ ] import pandas as pd
import re
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

Pada code diatas digunakan untuk melakukan import library yang akan digunakan untuk melakukan progress regression dan classification. Pandas as pd merupakan Pustaka python untuk analisis data. Sedangkan Numpy untuk komputasi numerik. Kemudian untuk matplotlib.pyplot untuk membuat grafik. Lalu seaborn adalah Pustaka visualisasi berbasis matplotlib. Kemudian sklearn pada bebeapa di gambar tersebut digunakan untuk membagi data menjadi subset untuk train, ada juga yang digunakan untuk model seperti decision tree regessor, knn regressor dan xgb regressor. Kemudian sklearn juga bisa digunakan untuk menghitung mean squared error dan r2 score.

```
[ ] with open('adult.names') as fp:
    cols = []
    for line in fp:
        sre = re.match(r'(?P<colname>[a-z\-\-]+):.*\.', line)
        if sre:
            cols.append(sre.group('colname'))
            cols.append('label')

    options = {'header': None, 'names': cols, 'skipinitialspace': True}

    train_df = pd.read_csv('adult.data', **options)

    test_df = pd.read_csv('adult.test', skiprows=1, **options)
    test_df['label'] = test_df['label'].str.rstrip('.')

    train_df.head(5)
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	label
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215648	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	335409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Pada kode ini membaca file metadata ('adult.names') dan file data ('adult.data' dan 'adult.test') dari dataset "Adult". Kolom dataset diekstraksi dari file 'adult.names' menggunakan ekspresi reguler untuk mencocokkan nama kolom, kemudian kolom "label" ditambahkan secara manual. File 'adult.data' dibaca ke dalam 'train_df', sementara 'adult.test' dibaca ke dalam 'test_df', dengan baris pertama dilewati karena merupakan header. Pada 'test_df', nilai kolom "label" dihapuskan tanda titik di akhir menggunakan 'str.rstrip('.')'. Akhirnya, lima baris pertama dari 'train_df' ditampilkan.

```
[ ] train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 32561 entries, 0 to 32560  
Data columns (total 15 columns):  
#   Column             Non-Null Count  Dtype    
---  ---               
0   age                32561 non-null  int64    
1   workclass          32561 non-null  object   
2   fnlwgt             32561 non-null  int64    
3   education          32561 non-null  object   
4   education-num      32561 non-null  int64    
5   marital-status     32561 non-null  object   
6   occupation         32561 non-null  object   
7   relationship       32561 non-null  object   
8   race               32561 non-null  object   
9   sex                32561 non-null  object   
10  capital-gain       32561 non-null  int64    
11  capital-loss       32561 non-null  int64    
12  hours-per-week     32561 non-null  int64    
13  native-country     32561 non-null  object   
14  label              32561 non-null  object   
dtypes: int64(6), object(9)  
memory usage: 3.7+ MB
```

Pada code ini menampilkan ringkasan informasi tentang dataframe `train_df`. Outputnya mencakup jumlah total baris, jumlah nilai non-null per kolom, tipe data masing-masing kolom, serta penggunaan memori keseluruhan. Pada dataset ini terdapat 32561 dengan 15 kolom.

```
[ ] # Mengganti missing values '?' menjadi NaN  
train_df = train_df.replace('?', np.nan)  
  
[ ] # Membersihkan kolom label dari spasi  
train_df['label'] = train_df['label'].str.strip()  
  
[ ] categorical_columns = ['workclass', 'education', 'marital-status', 'occupation', 'relationship', 'race', 'sex', 'native-country', 'label']  
for col in categorical_columns:  
    train_df[col] = train_df[col].astype('category')
```

Pada code ini mengganti nilai '?' dalam `train_df` dengan `NaN` untuk menandai data yang hilang. Selanjutnya, kolom "label" dibersihkan dari spasi di awal dan akhir nilai. Untuk kolom yang bersifat kategorikal, seperti workclass dan education, data diubah menjadi tipe category untuk menghemat memori dan mempermudah pengolahan data.

```
[ ] train_df.isnull().sum()
```

	0
age	0
workclass	1836
fnlwgt	0
education	0
education-num	0
marital-status	0
occupation	1843
relationship	0
race	0
sex	0
capital-gain	0
capital-loss	0
hours-per-week	0
native-country	583
label	0
dtype: int64	

Pada code ini menghitung jumlah nilai yang hilang (*missing values*) di setiap kolom pada `train_df`. Hasilnya memberikan gambaran tentang sejauh mana data yang hilang terdapat dalam dataset, sehingga membantu menentukan langkah-langkah pembersihan atau imputasi data.

Drop baris yang memiliki missing values train_df = train_df.dropna()

Pada code ini menghapus semua baris pada `train_df` yang mengandung nilai hilang (missing values). Langkah ini memastikan bahwa dataset bersih dari data kosong sebelum analisis atau pemodelan, tetapi juga dapat mengurangi jumlah data yang tersedia.

train_df.head()

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	label
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

Pada code Kode ini menampilkan lima baris pertama dari dataset `train_df`. Hal ini digunakan untuk memeriksa isi data, memastikan struktur dataset sesuai harapan, dan melihat contoh data setelah dilakukan pembersihan.


```
[ ] train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 30162 entries, 0 to 32560
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   age                    30162 non-null  int64
1   workclass              30162 non-null  category
2   fnlwgt                 30162 non-null  int64
3   education              30162 non-null  category
4   education-num          30162 non-null  int64
5   marital-status         30162 non-null  category
6   occupation             30162 non-null  category
7   relationship           30162 non-null  category
8   race                   30162 non-null  category
9   sex                    30162 non-null  category
10  capital-gain            30162 non-null  int64
11  capital-loss            30162 non-null  int64
12  hours-per-week         30162 non-null  int64
13  native-country         30162 non-null  category
14  label                   30162 non-null  category
dtypes: category(9), int64(6)
memory usage: 1.9 MB
```

Pada code ini menampilkan ringkasan informasi tentang dataframe `train_df`. Outputnya mencakup jumlah total baris, jumlah nilai non-null per kolom, tipe data masing-masing kolom, serta penggunaan memori keseluruhan. Setelah melakukan drop data yang nan, dataset ini menjadi 30162 baris dengan 15 kolom.

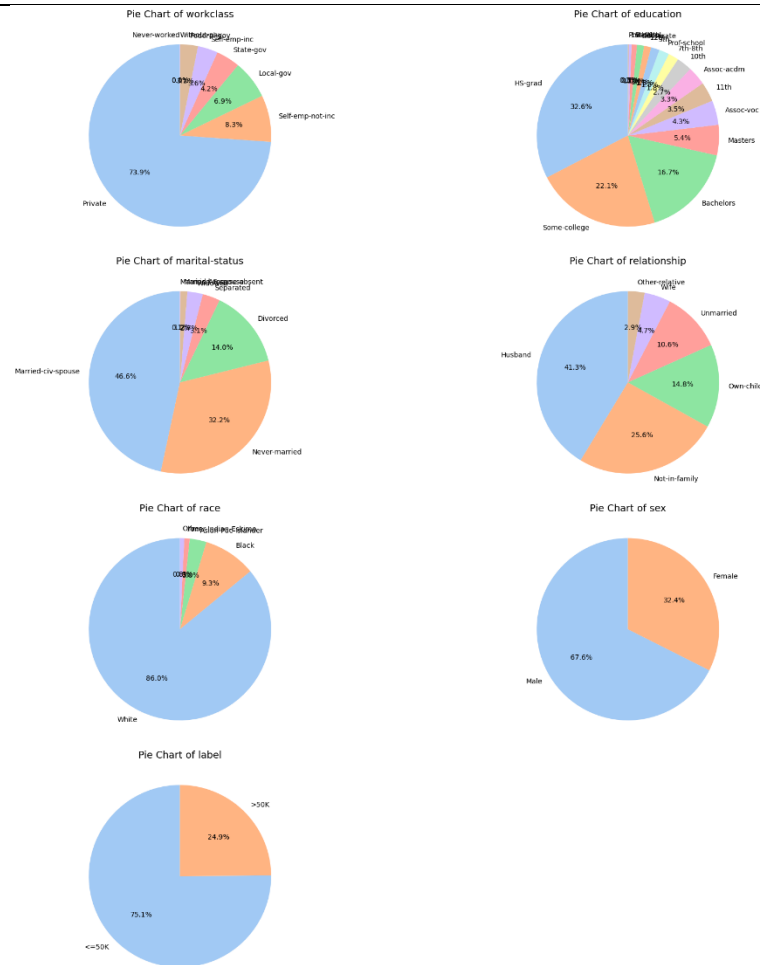
```
[ ] train_df.describe()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
count	30162.000000	3.016200e+04	30162.000000	30162.000000	30162.000000	30162.000000
mean	38.437902	1.897938e+05	10.121312	1092.007858	88.372489	40.931238
std	13.134665	1.056530e+05	2.549995	7406.346497	404.298370	11.979984
min	17.000000	1.376900e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.176272e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.784250e+05	10.000000	0.000000	0.000000	40.000000
75%	47.000000	2.376285e+05	13.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

Pada code ini menghasilkan statistik deskriptif dari dataset `train_df` untuk kolom numerik. Informasi yang ditampilkan mencakup jumlah nilai (*count*), rata-rata (*mean*), standar deviasi (*std*), nilai minimum (*min*), persentil (25%, 50%, 75%), dan nilai maksimum (*max*). Statistik ini membantu memahami distribusi data numerik dalam dataset.

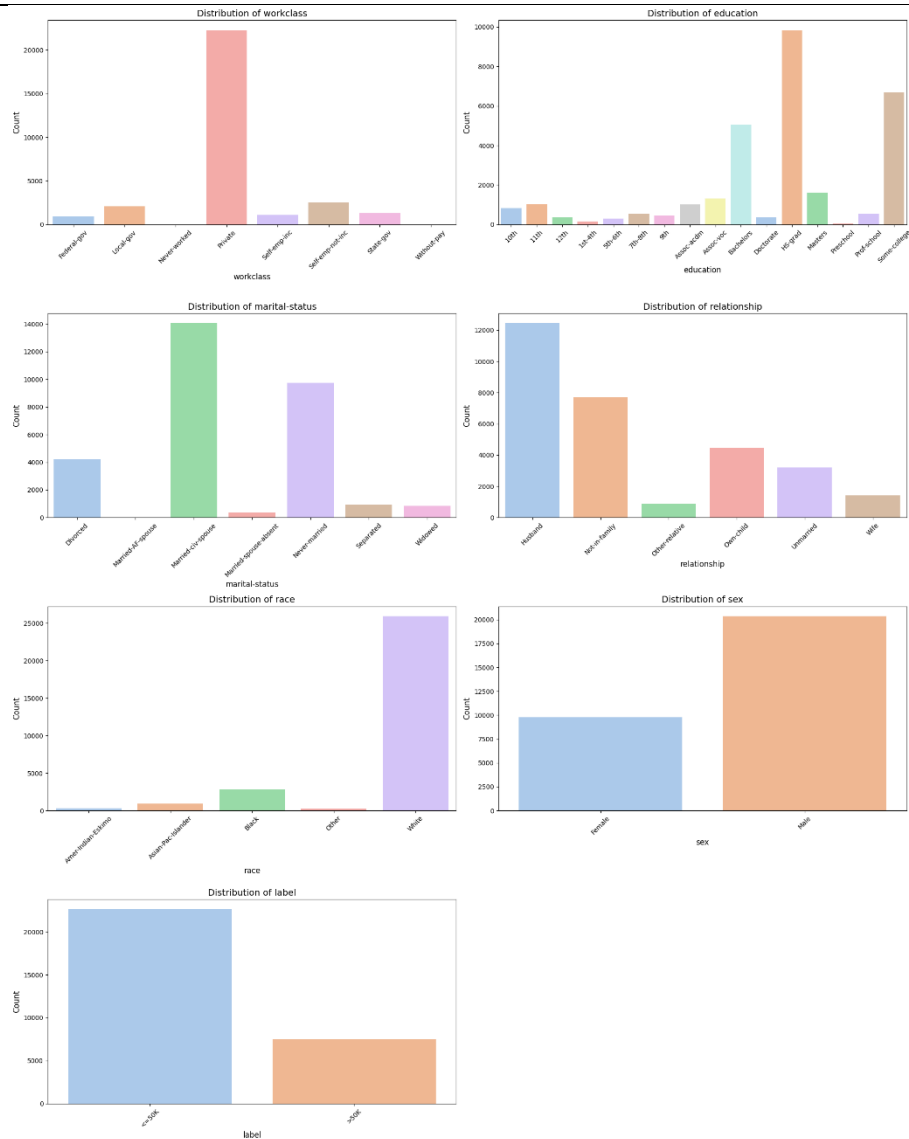
2. EDA

```
[ ] # PIE CHART: Proporsi dari fitur kategorikal
categorical_columns = ['workclass', 'education', 'marital-status', 'relationship', 'race', 'sex', 'label']
plt.figure(figsize=(20, 20))
for i, col in enumerate(categorical_columns, 1):
    plt.subplot(4, 2, i)
    data = train_df[col].value_counts()
    plt.pie(data, labels=data.index, autopct='%1.1f%%', startangle=90, colors=sns.color_palette('pastel'))
    plt.title(f'Pie Chart of {col}', fontsize=14)
plt.tight_layout()
plt.show()
```



Pada code ini membuat diagram pie chart untuk menunjukkan proporsi kategori dari fitur kategorikal dalam `train_df`. Kolom yang divisualisasikan meliputi workclass, education, marital-status, relationship, race, sex, dan label. Untuk setiap kolom, distribusi nilai dihitung, lalu pie chart digambarkan dengan label kategori, persentase, dan warna dari palet pastel. Visualisasi ini memberikan gambaran proporsi setiap kategori untuk membantu analisis data.

```
[ ] # BAR CHART: Distribusi kategori untuk fitur kategorikal
plt.figure(figsize=(20, 25))
for i, col in enumerate(categorical_columns, 1):
    plt.subplot(4, 2, i)
    sns.barplot(
        x=train_df[col].value_counts().index,
        y=train_df[col].value_counts().values,
        palette='pastel'
    )
    plt.title(f'Distribution of {col}', fontsize=14)
    plt.xlabel(col, fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

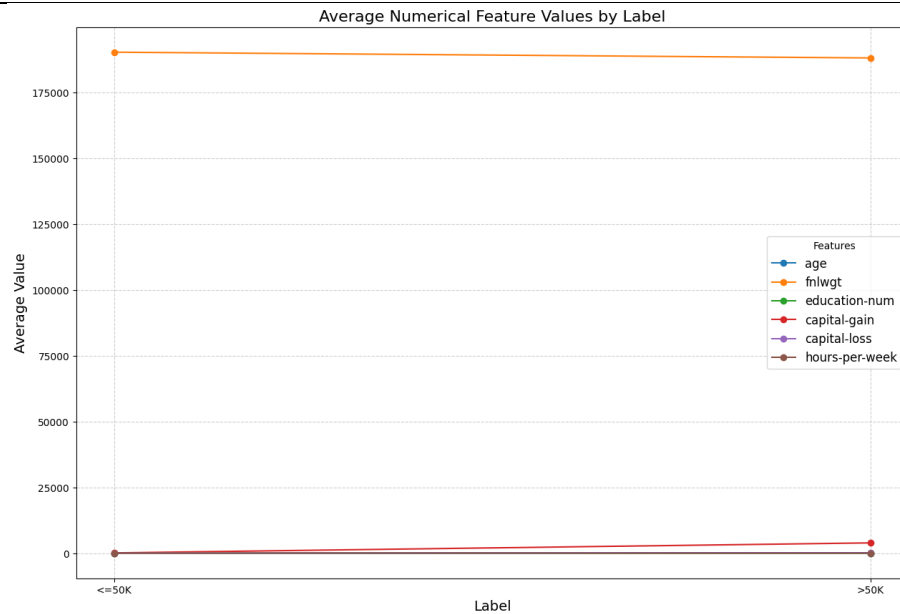


Pada code ini membuat bar chart untuk menampilkan distribusi kategori dari fitur kategorikal dalam `train_df`. Untuk setiap kolom, jumlah kemunculan setiap kategori dihitung, lalu divisualisasikan menggunakan bar chart dengan warna dari palet pastel. Grafik ini menampilkan jumlah tiap kategori dengan sumbu horizontal menunjukkan kategori dan sumbu vertikal menunjukkan jumlahnya. Tujuannya adalah memahami frekuensi masing-masing kategori secara visual.

```
[ ] # LINE GRAPH: Tren nilai rata-rata fitur numerik per kategori
plt.figure(figsize=(15, 10))
numerical_columns = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']
avg_values = train_df.groupby('label')[numerical_columns].mean()

for col in numerical_columns:
    plt.plot(avg_values.index, avg_values[col], marker='o', label=col)

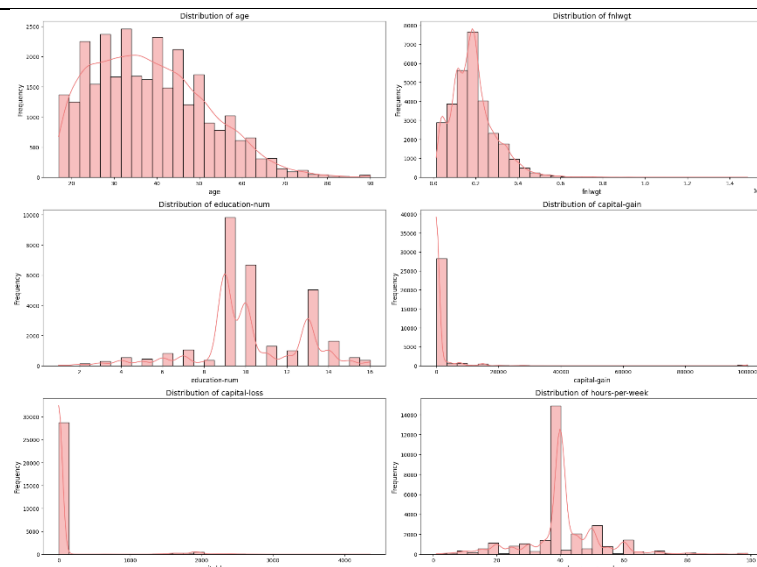
plt.title('Average Numerical Feature Values by Label', fontsize=16)
plt.xlabel('Label', fontsize=14)
plt.ylabel('Average Value', fontsize=14)
plt.legend(title='Features', fontsize=12)
plt.grid(visible=True, linestyle='--', alpha=0.6)
plt.show()
```



Pada code Kode ini membuat grafik garis untuk menunjukkan tren nilai rata-rata fitur numerik berdasarkan kategori "label" dalam `train_df`. Setiap garis mewakili rata-rata nilai fitur numerik seperti age, fnlwgt, education-num, capital-gain, capital-loss, dan hours-per-week untuk setiap kategori "label". Visualisasi ini membantu memahami bagaimana nilai rata-rata masing-masing fitur numerik berbeda di setiap kategori label, dengan menambahkan garis dan marker untuk setiap fitur.

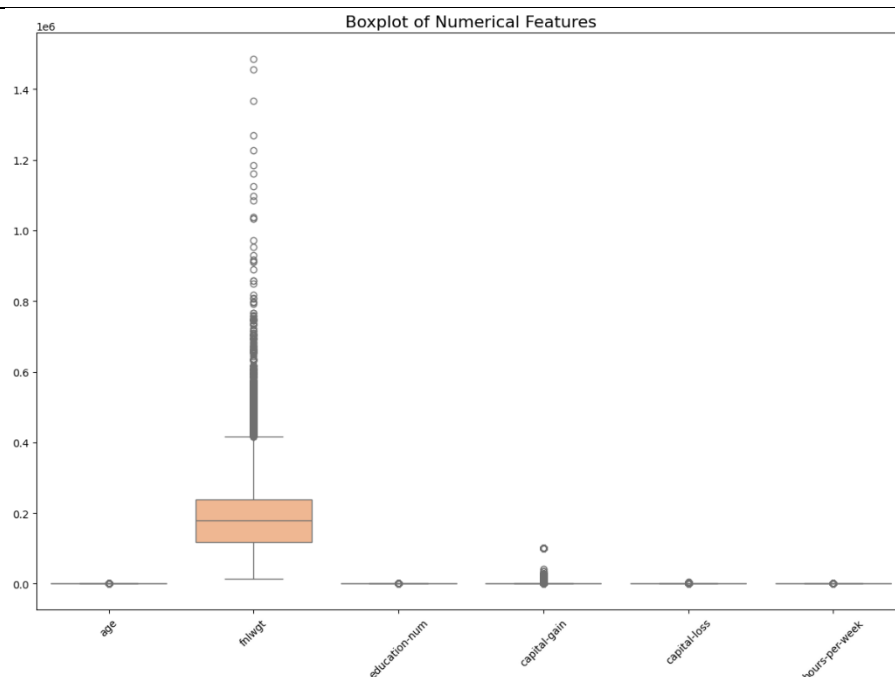
```
[ ] # Histogram untuk fitur numerik
numerical_columns = ['age', 'fnlwgt', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

plt.figure(figsize=(20, 15))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(3, 2, i)
    sns.histplot(train_df[col], kde=True, color='lightcoral', bins=30)
    plt.title(f'Distribution of {col}', fontsize=14)
    plt.xlabel(col, fontsize=12)
    plt.ylabel('Frequency', fontsize=12)
plt.tight_layout()
plt.show()
```



Pada code ini membuat histogram untuk menampilkan distribusi fitur numerik dalam `train_df`, seperti age, fnlwgt, education-num, capital-gain, capital-loss, dan hours-per-week. Setiap subplot menunjukkan distribusi data dengan menggunakan histogram dan garis KDE (Kernel Density Estimate) untuk memperlihatkan pola distribusi data secara lebih halus. Visualisasi ini membantu memahami sebaran nilai setiap fitur numerik.

```
[ ] # Boxplot untuk fitur numerik (mendeteksi outlier)
plt.figure(figsize=(15, 10))
sns.boxplot(data=train_df[numerical_columns], palette='pastel')
plt.title('Boxplot of Numerical Features', fontsize=16)
plt.xticks(rotation=45)
plt.show()
```



Pada code Kode ini membuat boxplot untuk fitur numerik dalam `train_df`, seperti age, fnlwgt, education-num, capital-gain, capital-loss, dan hours-per-week. Visualisasi ini digunakan untuk mendeteksi adanya outlier atau pencilan dalam data, dengan menampilkan rentang nilai, kuartil, dan titik-titik yang menunjukkan nilai ekstrim. Boxplot ini membantu mengidentifikasi distribusi dan potensi masalah data dalam fitur numerik.

```
[ ] # Menghitung matriks korelasi hanya untuk kolom numerik
corr_matrix = train_df.select_dtypes(include=np.number).corr()

# Menampilkan hasil korelasi dengan angka
print("Correlation Matrix:")
print(corr_matrix)
```

```

Correlation Matrix:
      age    fnlwgt  education-num  capital-gain  capital-loss
age      1.000000 -0.076511      0.043526      0.080154      0.060165
fnlwgt   -0.076511  1.000000     -0.044992      0.000422     -0.009750
education-num  0.043526 -0.044992      1.000000      0.124416      0.079646
capital-gain  0.080154  0.000422      0.124416      1.000000     -0.032229
capital-loss  0.060165 -0.009750      0.079646     -0.032229      1.000000
hours-per-week 0.101599 -0.022886      0.152522      0.080432      0.052417

      hours-per-week
age              0.101599
fnlwgt           -0.022886
education-num     0.152522
capital-gain      0.080432
capital-loss      0.052417
hours-per-week    1.000000

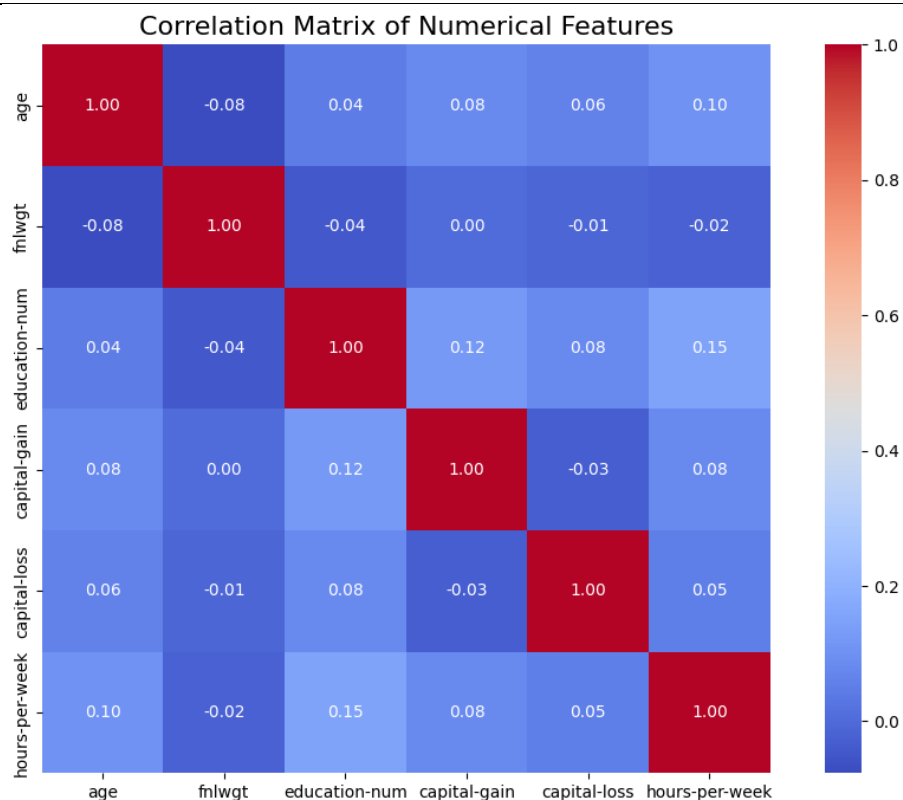
```

Pada code Kode ini menghitung matriks korelasi hanya untuk kolom numerik dalam `train_df`. Dengan menggunakan metode `.corr()`, matriks korelasi dihitung untuk menilai hubungan linier antar fitur numerik. Hasilnya ditampilkan dalam bentuk tabel yang menunjukkan nilai korelasi antar setiap pasangan fitur numerik, yang membantu mengidentifikasi hubungan atau ketergantungan antar fitur.

```

[ ] # Heatmap korelasi antar fitur numerik
plt.figure(figsize=(12, 8))
corr_matrix = train_df[numerical_columns].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', square=True)
plt.title('Correlation Matrix of Numerical Features', fontsize=16)
plt.show()

```



Pada code ini membuat heatmap untuk menampilkan matriks korelasi antar fitur numerik dalam `train_df`, seperti age, fnlwgt, education-num, capital-gain, capital-loss, dan hours-per-week. Dengan menggunakan fungsi `sns.heatmap`, visualisasi ini menunjukkan hubungan linier antar fitur numerik dengan pewarnaan yang memudahkan identifikasi tingkat korelasi. Angka korelasi ditampilkan di setiap sel untuk memberikan informasi lebih detail tentang hubungan antar fitur.

3. PIPELINE

```
[ ] from sklearn.model_selection import train_test_split, cross_val_score
    from sklearn.preprocessing import StandardScaler, OneHotEncoder
    from sklearn.compose import ColumnTransformer
    from sklearn.pipeline import Pipeline
    from sklearn.linear_model import LogisticRegression
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.neighbors import KNeighborsClassifier
    from xgboost import XGBClassifier
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
    from sklearn.preprocessing import LabelEncoder

    # 1. Memisahkan fitur dan label
    X = train_df.drop('label', axis=1)
    y = train_df['label']

    # Encode the target variable using LabelEncoder
    le = LabelEncoder()
    y = le.fit_transform(y)

    # 2. Identifikasi kolom numerik dan kategorikal
    numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
    categorical_features = X.select_dtypes(include=['category']).columns

    # 3. Split data menjadi training dan testing
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

    # 4. Preprocessing pipeline
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', StandardScaler(), numerical_features), # Scaling untuk fitur numerik
            ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features) # Encoding fitur kategorikal
        ]
    )
```

Pada code ini melakukan persiapan data untuk model klasifikasi dengan beberapa langkah penting. Pertama, fitur (`X`) dan label (`y`) dipisahkan, dan label dikodekan menggunakan `LabelEncoder` untuk mengubah kategori menjadi nilai numerik. Selanjutnya, kolom numerik dan kategorikal diidentifikasi. Data kemudian dibagi menjadi set pelatihan dan pengujian dengan menggunakan `train_test_split`. Sebuah pipeline preprocessing dibuat menggunakan `ColumnTransformer`, di mana fitur numerik diskalakan dengan `StandardScaler`, sementara fitur kategorikal diubah menjadi variabel dummy melalui `OneHotEncoder`. Ini mempersiapkan data untuk diterapkan pada model klasifikasi.


```
# 5. Pipeline model
models = {
    "Logistic Regression": LogisticRegression(max_iter=500),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "k-NN": KNeighborsClassifier(n_neighbors=5),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
}

# 6. Training dan evaluasi
for name, model in models.items():
    print(f"Model: {name}")
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor), # Preprocessing
        ('classifier', model) # Model
    ])

    # Training
    pipeline.fit(X_train, y_train)

    # Prediksi
    y_pred = pipeline.predict(X_test)

    # Evaluasi
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
    print("-" * 50)
```

```
Model: Logistic Regression
Confusion Matrix:
[[4204  327]
 [ 593  909]]

Classification Report:
              precision    recall  f1-score   support

     0       0.88        0.93        0.90        4531
     1       0.74        0.61        0.66        1502

 accuracy          0.85
 macro avg         0.81
 weighted avg      0.84

Accuracy: 0.85
-----

Model: Decision Tree
Confusion Matrix:
[[3938  593]
 [ 561  941]]

Classification Report:
              precision    recall  f1-score   support

     0       0.88        0.87        0.87        4531
     1       0.61        0.63        0.62        1502

 accuracy          0.81
 macro avg         0.74
 weighted avg      0.81

Accuracy: 0.81
-----
```

```

Model: k-NN
Confusion Matrix:
[[4072  459]
 [ 585  917]]

Classification Report:
              precision    recall  f1-score   support

     0       0.87       0.90       0.89       4531
     1       0.67       0.61       0.64       1502

 accuracy          0.83          6033
 macro avg         0.77          0.75       0.76       6033
 weighted avg      0.82          0.83       0.82       6033

Accuracy: 0.83
-----
Model: XGBoost
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158:
Parameters: { "use_label_encoder" } are not used.

warnings.warn(msg, UserWarning)
Confusion Matrix:
[[4226  305]
 [ 514  988]]

Classification Report:
              precision    recall  f1-score   support

     0       0.89       0.93       0.91       4531
     1       0.76       0.66       0.71       1502

 accuracy          0.86          6033
 macro avg         0.83          0.80       0.81       6033
 weighted avg      0.86          0.86       0.86       6033

Accuracy: 0.86
-----

```

Pada code ini melatih dan mengevaluasi empat model klasifikasi yang berbeda: Logistic Regression, Decision Tree, k-NN, dan XGBoost. Setiap model dimasukkan ke dalam pipeline yang mencakup preprocessing data menggunakan `ColumnTransformer`, diikuti dengan classifier yang sesuai. Setelah pelatihan, model diuji pada data pengujian dan hasil evaluasi ditampilkan, termasuk matriks kebingungan, laporan klasifikasi, dan akurasi untuk setiap model. Proses ini memungkinkan perbandingan kinerja berbagai algoritma klasifikasi dalam menyelesaikan masalah yang ada.

4. Hyperparameter Tuning

```
[ ] from sklearn.model_selection import GridSearchCV

# 1. Definisikan parameter untuk setiap model
param_grid = {
    "Logistic Regression": {
        'classifier__C': [0.01, 0.1, 1, 10],
        'classifier__solver': ['liblinear', 'lbfgs']
    },
    "Decision Tree": {
        'classifier__max_depth': [None, 10, 20, 30],
        'classifier__min_samples_split': [2, 5, 10],
        'classifier__min_samples_leaf': [1, 2, 4]
    },
    "k-NN": {
        'classifier__n_neighbors': [3, 5, 7, 9],
        'classifier__weights': ['uniform', 'distance'],
        'classifier__metric': ['euclidean', 'manhattan']
    },
    "XGBoost": {
        'classifier__n_estimators': [50, 100, 150],
        'classifier__max_depth': [3, 5, 7],
        'classifier__learning_rate': [0.01, 0.1, 0.2]
    }
}

# 2. Perform Hyperparameter Tuning dan evaluasi
best_models = {}

for name, model in models.items():
    print(f"Tuning Model: {name}")

    # Pipeline model
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])

[ ] # Grid Search
    grid_search = GridSearchCV(
        estimator=pipeline,
        param_grid=param_grid[name],
        cv=5,
        scoring='accuracy',
        n_jobs=-1
    )

    # Fit data
    grid_search.fit(X_train, y_train)

    # Best Model
    best_model = grid_search.best_estimator_
    best_models[name] = best_model

    # Evaluasi
    y_pred = best_model.predict(X_test)

    print("Best Parameters:", grid_search.best_params_)
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
    print("-" * 50)
```

```

Tuning Model: Logistic Regression
Best Parameters: {'classifier__C': 0.1, 'classifier__solver': 'lbfgs'}
Confusion Matrix:
[[4207 324]
 [ 606 896]]

Classification Report:
              precision    recall  f1-score   support

     0       0.87       0.93       0.90       4531
     1       0.73       0.60       0.66       1502

 accuracy      0.85      0.85      0.85      6033
 macro avg     0.80      0.76      0.78      6033
 weighted avg   0.84      0.85      0.84      6033

Accuracy: 0.85
-----
Tuning Model: Decision Tree
Best Parameters: {'classifier__max_depth': 10, 'classifier__min_samples_leaf': 4, 'classifier__min_samples_split': 2}
Confusion Matrix:
[[4249 282]
 [ 620 882]]

Classification Report:
              precision    recall  f1-score   support

     0       0.87       0.94       0.90       4531
     1       0.76       0.59       0.66       1502

 accuracy      0.85      0.85      0.85      6033
 macro avg     0.82      0.76      0.78      6033
 weighted avg   0.84      0.85      0.84      6033

Accuracy: 0.85
-----
Tuning Model: k-NN
/usr/local/lib/python3.10/dist-packages/numpy/ma/core.py:2820: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=copy,
Best Parameters: {'classifier__metric': 'euclidean', 'classifier__n_neighbors': 9, 'classifier__weights': 'uniform'}
Confusion Matrix:
[[4107 424]
 [ 581 921]]

Classification Report:
              precision    recall  f1-score   support

     0       0.88       0.91       0.89       4531
     1       0.68       0.61       0.65       1502

 accuracy      0.83      0.83      0.83      6033
 macro avg     0.78      0.76      0.77      6033
 weighted avg   0.83      0.83      0.83      6033

Accuracy: 0.83
-----
Tuning Model: XGBoost
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [15:02:58] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

warnings.warn(msg, UserWarning)
Best Parameters: {'classifier__learning_rate': 0.1, 'classifier__max_depth': 5, 'classifier__n_estimators': 150}
Confusion Matrix:
[[4255 276]
 [ 533 969]]

Classification Report:
              precision    recall  f1-score   support

     0       0.89       0.94       0.91       4531
     1       0.78       0.65       0.71       1502

 accuracy      0.87      0.87      0.87      6033
 macro avg     0.83      0.79      0.81      6033
 weighted avg   0.86      0.87      0.86      6033

Accuracy: 0.87

```

Pada code ini melakukan pencarian hyperparameter (Grid Search) untuk empat model klasifikasi: Logistic Regression, Decision Tree, k-NN, dan XGBoost. Untuk setiap model, parameter yang relevan ditentukan, kemudian dilakukan pencarian hyperparameter menggunakan `GridSearchCV` dengan validasi silang (cross-validation) dan pengukuran akurasi. Setelah menemukan kombinasi parameter terbaik, model terbaik dievaluasi menggunakan matriks kebingungan, laporan klasifikasi, dan akurasi pada data uji. Hasil terbaik dari setiap model disimpan dalam dictionary `best_models`.

C. KESIMPULAN

1) Regression Model

PIPELINE			HYPERPARAMETER TUNING	
Model	MSE	Model	MSE	Model
Decision Tree	113.438	Decision Tree	113.438	Decision Tree
Polynomial	189.5494	Polynomial	189.5494	Polynomial
XGBoost	86.5028	XGBoost	86.5028	XGBoost

Dapat disimpulkan bahwa pada regression model yang bagus terdapat pada **XGBoost**.

2) Classification Model

	PIPELINE	HYPERPARAMETER TUNING
Model	Accuracy	Accuracy
KNN	0.83	0.83
Decision Tree	0.85	0.85
XGBoost	0.86	0.87
Logistic Regression	0.85	0.85

Dapat disimpulkan bahwa pada regression model yang bagus terdapat pada **XGBoost**.