

# GROUP 6 DATA MINING

CLASSIFICATION  
OF MBTI & SALARY  
SCALE

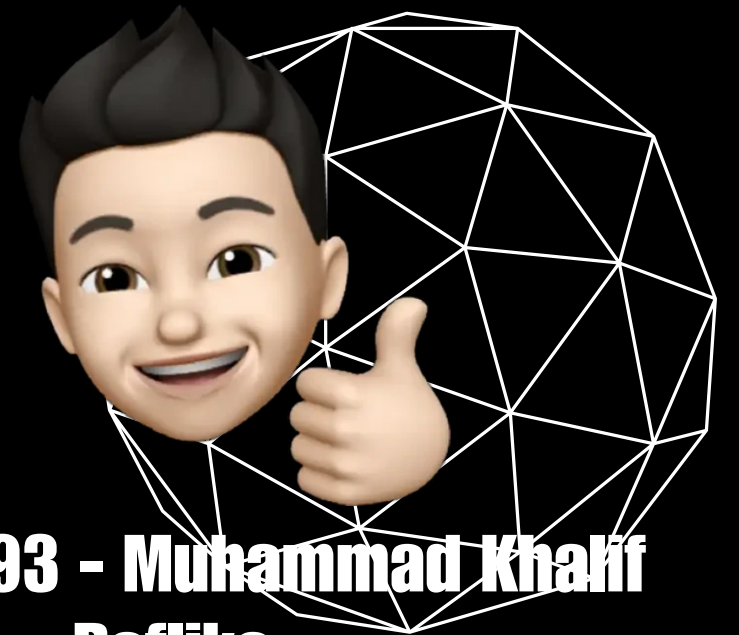
## OUR MEMBERS



**2602130445 - Najla Khalishah**



**2602195880 - Muhammad Denny  
Hardiyanto**



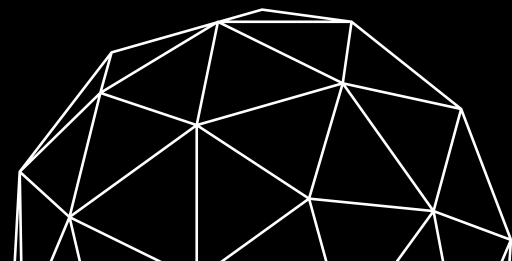
**2602197993 - Muhammad Khalif  
Raflika**



**2602185974 - Muhammad Tsaqif  
Samarta**



**2602187203 - Muhammad Rizqi  
Anugerah**




# PROJECT BACKGROUND



Proyek ini berfokus pada pengklasifikasian tipe kepribadian MBTI (Myers-Briggs Type Indicator) dan menghubungkannya dengan skala gaji

## Project Goals

- Menggunakan teknik klasifikasi untuk memahami hubungan antara MBTI, skala gaji, dan attribute lainnya seperti pendidikan
  - Memberikan wawasan bagi perusahaan untuk pengelolaan karyawan berdasarkan kepribadian mereka.
- 

## Relevansi MBTI dan Karir:

- Tipe MBTI sering digunakan untuk menilai kesesuaian pekerjaan dengan kepribadian.
- Memahami hubungan ini dapat membantu perusahaan dalam perencanaan karir dan manajemen karyawan.

## Kontribusi Proyek:

- Memberikan wawasan bagi individu untuk memahami pengaruh kepribadian terhadap potensi karir mereka.
- Membantu perusahaan menentukan strategi pengembangan SDM berbasis data.

# DATASET

## **Dataset MBTI:**

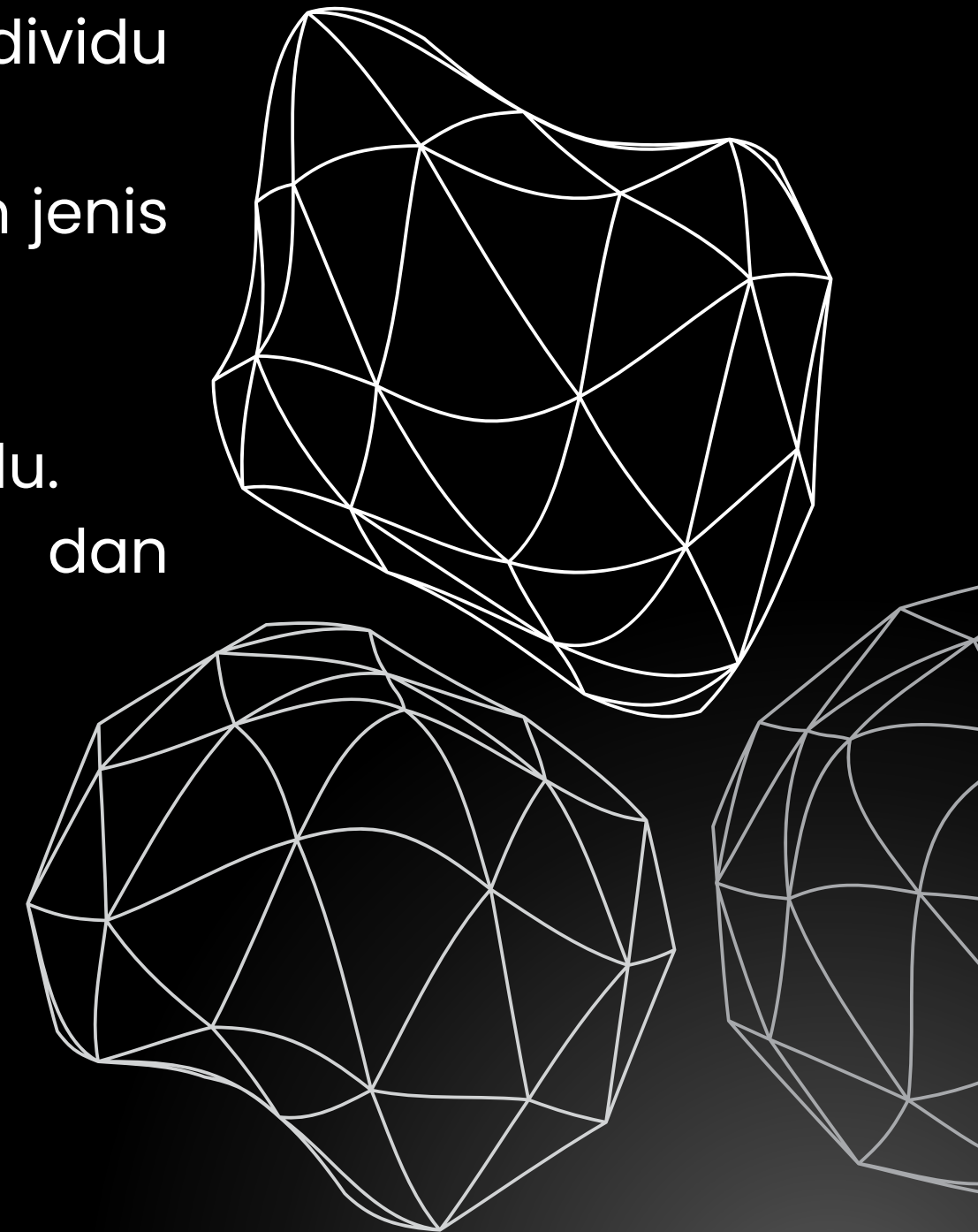
- Berisi informasi kepribadian berdasarkan MBTI dan karakteristik individu seperti usia, jenis kelamin, dan skor dimensi kepribadian.
- Digunakan untuk mengidentifikasi hubungan antara kepribadian dan jenis pekerjaan atau gaji.

## **Dataset Job Salary:**

- Berisi informasi tentang jenis pekerjaan dan gaji yang diterima individu.
- Digunakan untuk menentukan hubungan antara kepribadian dan pendapatan.

## **Ukuran Dataset:**

- Dataset MBTI: 1000 baris data dengan 10 kolom
- Dataset Job Salary: 200 baris data dengan 4 kolom





# DATASET

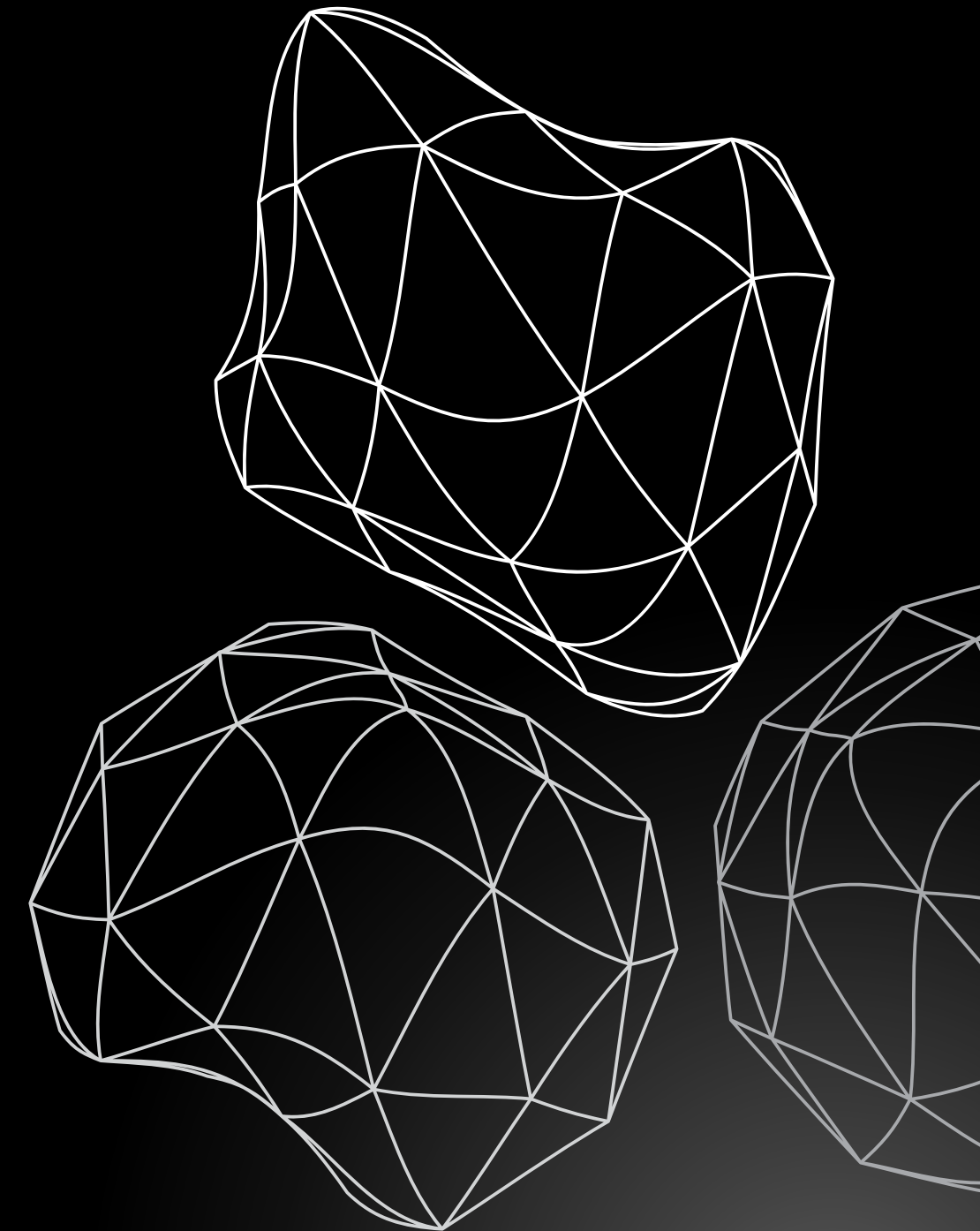
## Atribute dalam dataset

### Dataset 1: classification-mbti-data

Atributte	Description
participant_id	ID unik untuk setiap participant
Introversion Score	Presentase Introversion
Sensing Score	Presentase Sensing
Thinking Score	Presentase Thinking
Judging Score	Presentase Judging

### Dataset 2: classification-demography-data

Atributte	Description
participant_id	ID unik untuk setiap participant
Age	Umur participant
Gender	Jenis Kelamin (male/female)
Education	Pendidikan terakhir participant
Interest	Minat atau bidang ketertarikan participant
Personality	Tipe kepribadian MBTI



# DATASET

## Contoh datanya

Dataset 1: classification-mbti-data

participant_id	Introversion Score	Sensing Score	Thinking Score	Judging Score
----------------	--------------------	---------------	----------------	---------------

Dataset 2: classification-demography-data

participant_id	Age	Gender	Education	Interest	Personality
1	21	Female	1	Arts	ENTP
2	24	Female	1	Unknown	INTP



# READ DATASET

1. Mengimpor library yang dibutuhkan dan input dataset yang dibutuhkan

```
[ ] #Import necessary packages
import numpy as np
import pandas as pd
```

```
[ ] from google.colab import drive
drive.mount('/content/drive')
```

```
[ ] classification_mbti_data = '/content/drive/MyDrive/Python Classification/classification-mbti-data.csv'
classification_demography_data = '/content/drive/MyDrive/Python Classification/classification-demography-data.csv'
```

```
# Membaca dataset
classification_mbti_data = pd.read_csv(classification_mbti_data)
classification_demography_data = pd.read_csv(classification_demography_data)

# Menampilkan beberapa baris awal dari masing-masing dataset
print("MBTI Dataset:")
print(classification_mbti_data.head())

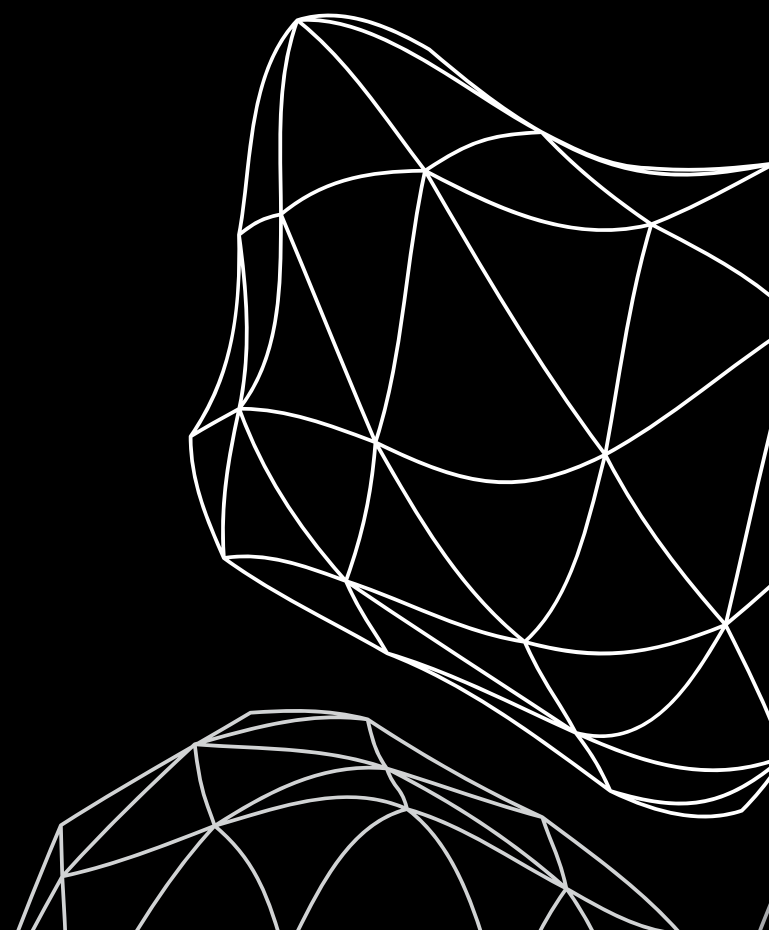
print("\nTrain-Test Dataset:")
print(classification_demography_data.head())
```

```
↗ MBTI Dataset:
  participant_id  Introversion Score  Sensing Score  Thinking Score  \
0              1          5.89208         2.144395         7.32363
1              2          7.02910         6.469302         4.16472
2              3          5.46525         4.179244         2.82487
3              4          3.59804         6.189259         5.31347
4              5          1.06869         7.143507         3.84411
```

```
      Judging Score
0          5.462224
1          5.454442
2          5.080477
3          3.677984
4          6.347241
```

```
Train-Test Dataset:
  participant_id  Age  Gender  Education  Interest  Personality
0              1   21  Female          1      Arts      ENTP
1              2   26  Female          1    Others      ESFP
2              3   30   Male          0    Sports      ENFJ
3              4   31  Female          0    Others      ISFP
4              5   33  Female          0    Sports      ISFJ
```

2. import database yang akan diproses



# READ DATASET

3. ambil data merge dan hitung jumlah minimum value dari dataset tersebut

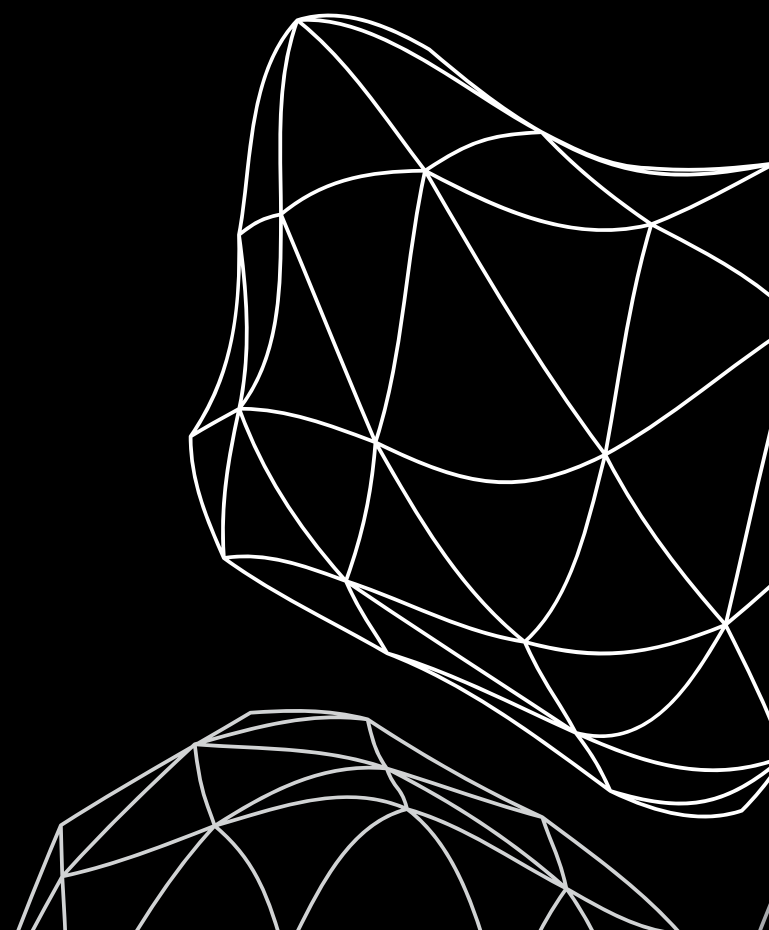
```
[ ] # Merge kedua dataset berdasarkan participant_id
    # merged_data = pd.merge(classification_mbti_data, classification_demography_data, on="participant_id")
    merged_data = pd.read_csv('/content/drive/MyDrive/Python Classification/data (2).csv')

[ ] personality_counts = merged_data['Personality'].value_counts()
    min_count = personality_counts.min()
    print(f"Jumlah data minimum per personality: {min_count}")

→ Jumlah data minimum per personality: 262
```

4. masukkan data kedalam dataframe

```
[ ] dfC = pd.DataFrame(classification_mbti_data)
    dfDC = pd.DataFrame(classification_demography_data)
```





# DATA CLEANING

## 5. mengecek jumlah null

```
# Mengecek jumlah nilai null pada setiap kolom  
merged_data.isnull().sum()
```

	0
Age	0
Gender	0
Education	0
Introversion Score	0
Sensing Score	0
Thinking Score	0
Judging Score	0
Interest	0
Personality	0

## 6. mengecek apakah ada nilai null

```
[ ] # Mengecek apakah ada nilai null pada seluruh dataset  
merged_data.isnull().values.any()
```

```
False
```

## 7. mengecek nilai 0

```
[ ] # Mengecek jumlah nilai 0 pada setiap kolom  
(merged_data == 0).sum()
```

	0
Age	0
Gender	0
Education	75408
Introversion Score	0
Sensing Score	8
Thinking Score	0
Judging Score	338
Interest	0
Personality	0

## 8. mengecek nilai 0

```
[ ] # Mengecek apakah ada nilai 0 pada seluruh dataset  
(merged_data == 0).any().any()
```

```
True
```

# DATA CLEANING

```
[ ] # Kolom yang ingin dicek
columns_to_check = ['Introversion Score', 'Sensing Score', 'Thinking Score', 'Judging Score']

# Mengecek jumlah nilai 0 pada kolom yang dipilih
for column in columns_to_check:
    zero_count = (merged_data[column] == 0).sum()
    print(f"Jumlah nilai 0 pada kolom '{column}': {zero_count}")

# Mengecek apakah ada nilai 0 pada kolom yang dipilih
has_zero_values = (merged_data[columns_to_check] == 0).any().any()
print(f"Apakah ada nilai 0 pada kolom yang dipilih: {has_zero_values}")
```

```
➞ Jumlah nilai 0 pada kolom 'Introversion Score': 0
Jumlah nilai 0 pada kolom 'Sensing Score': 8
Jumlah nilai 0 pada kolom 'Thinking Score': 0
Jumlah nilai 0 pada kolom 'Judging Score': 338
Apakah ada nilai 0 pada kolom yang dipilih: True
```

```
[ ] for column in columns_to_check:
    # Mengambil nilai minimum yang bukan 0
    min_value = merged_data[merged_data[column] != 0][column].min()

    # Mengganti nilai 0 dengan nilai minimum
    merged_data.loc[merged_data[column] == 0, column] = min_value
```


1. Menentukan kolom yang ingin dicek
2. Mengecek jumlah nilai 0 pada kolom yang dipilih
3. Mengecek apakah ada nilai 0 pada kolom yang dipilih

4. validasi dengan mengambil data yang bukan 0, dan dimasukkan ke table baru

# DATA CLEANING

```
[ ] # Mengecek jumlah baris duplikat
num_duplicates = merged_data.duplicated().sum()
print(f"Jumlah baris duplikat: {num_duplicates}")

# Menampilkan baris duplikat (opsional)
duplicate_rows = merged_data[merged_data.duplicated()]
print("\nBaris duplikat:")
print(duplicate_rows)
```

 Jumlah baris duplikat: 0

Baris duplikat:  
Empty DataFrame  
Columns: [Age, Gender, Education, Introversion Score, Sensing Score, Thinking Score, Judging Score, Interest,  
Index: []

## 1. Mengecek data duplikat

## 2. menghapus data yang duplikat

```
[ ] # Menghapus baris duplikat dan menyimpan hasilnya di DataFrame baru
merged_data_no_duplicates = merged_data.drop_duplicates()

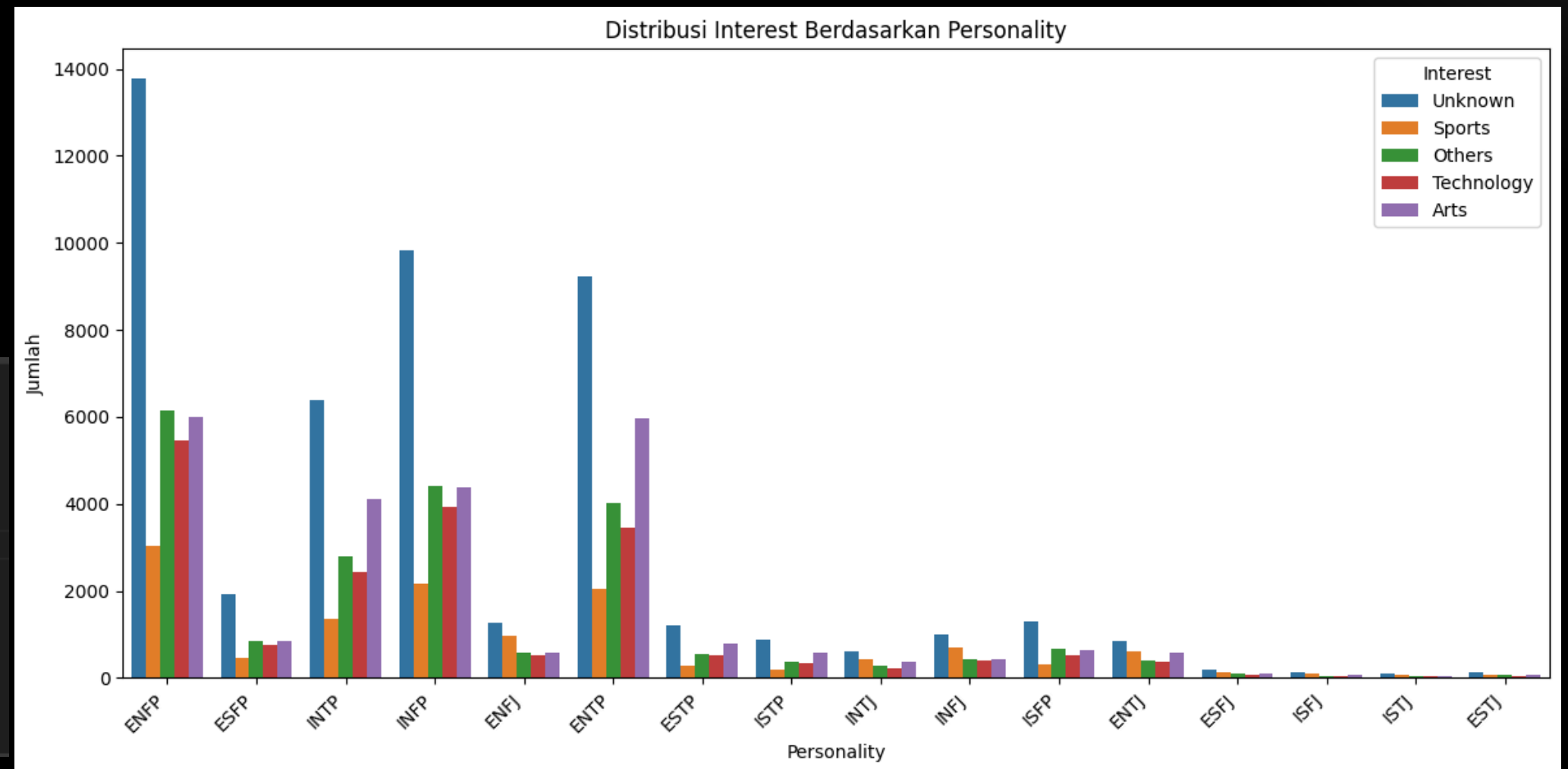
# Atau, Anda dapat menghapus baris duplikat secara inplace:
merged_data.drop_duplicates(inplace=True)
```

# VISUALISASI

## 1. Distribusi Interest berdasarkan personality

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

plt.figure(figsize=(12, 6))
sns.countplot(x='Personality', hue='Interest', data=merged_data)
plt.title('Distribusi Interest Berdasarkan Personality')
plt.xlabel('Personality')
plt.ylabel('Jumlah')
plt.xticks(rotation=45, ha='right') # Rotasi label sumbu x agar mudah dibaca
plt.legend(title='Interest')
plt.tight_layout() # Menyesuaikan layout agar tidak terpotong
plt.show()
```



# PRE-PROCESSING

## 1. Cek data 'Unknown'

```
[ ] # Cek jumlah data unknown
    unknown_count = merged_data['Interest'].value_counts().get('Unknown', 0)
    total_count = len(merged_data)
    print(f"Unknown Count: {unknown_count}, Total Count: {total_count}")
```

```
→ Unknown Count: 48835, Total Count: 128061
```

## 2. Split data into inputing & predicting sets

```
# Split data into imputing and predicting sets
imputing_data = merged_data[merged_data['Interest'] != 'Unknown']
predicting_data = merged_data[merged_data['Interest'] == 'Unknown']
```



# PRE-PROCESSING

## 3. Prepare features and target for imputation

```
# Prepare features and target for imputation
X_impute = imputing_data.drop(columns=['Interest'])
y_impute = imputing_data['Interest']
```

## 4. Encode categorical features in X\_impute

```
# Create a LabelEncoder object
encoder = LabelEncoder()

# Apply label encoding to categorical features in X_impute
for column in X_impute.select_dtypes(include=['object']).columns:
    X_impute[column] = encoder.fit_transform(X_impute[column])
```

# PRE-PROCESSING

5. Prepare features and target for prediction

```
# Prepare features for prediction
X_predict = predicting_data.drop(columns=['Interest'])
```

6. Encode categorical features in X\_predict

```
# Apply label encoding to categorical features in X_predict
for column in X_predict.select_dtypes(include=['object']).columns:
    X_predict[column] = encoder.fit_transform(X_predict[column])
```

# CLASSIFICATION - RF

7. Membuat RandomForestClassifier dengan random\_state=42 untuk reproduktifitas. Dan latih model menggunakan X\_impute dan y\_impute untuk mempelajari pola dalam memprediksi 'Minat'.

```
# Create and fit the RandomForestClassifier
rf_imputer = RandomForestClassifier(random_state=42)
rf_imputer.fit(X_impute, y_impute)
```

8. dengan model yang sudah di-*train* untuk memprediksi 'Interest' untuk subset predicting\_data. Mengganti 'Unknown' dengan nilai yang diprediksi

```
# Predict missing 'Interest' values
predicted_interest = rf_imputer.predict(X_predict)
```

```
# Impute predicted values back into the original DataFrame
merged_data.loc[merged_data['Interest'] == 'Unknown', 'Interest'] = predicted_interest
```

# CLASSIFICATION - RF

## 9. Hitung jumlah dan proporsi unknown

```
# Hitung jumlah dan proporsi Unknown
total_data = len(merged_data)
unknown_count = merged_data['Interest'].value_counts().get('Unknown', 0)
unknown_percentage = (unknown_count / total_data) * 100

print(f"Unknown Count: {unknown_count}")
print(f"Percentage of Unknown: {unknown_percentage:.2f}%")
```

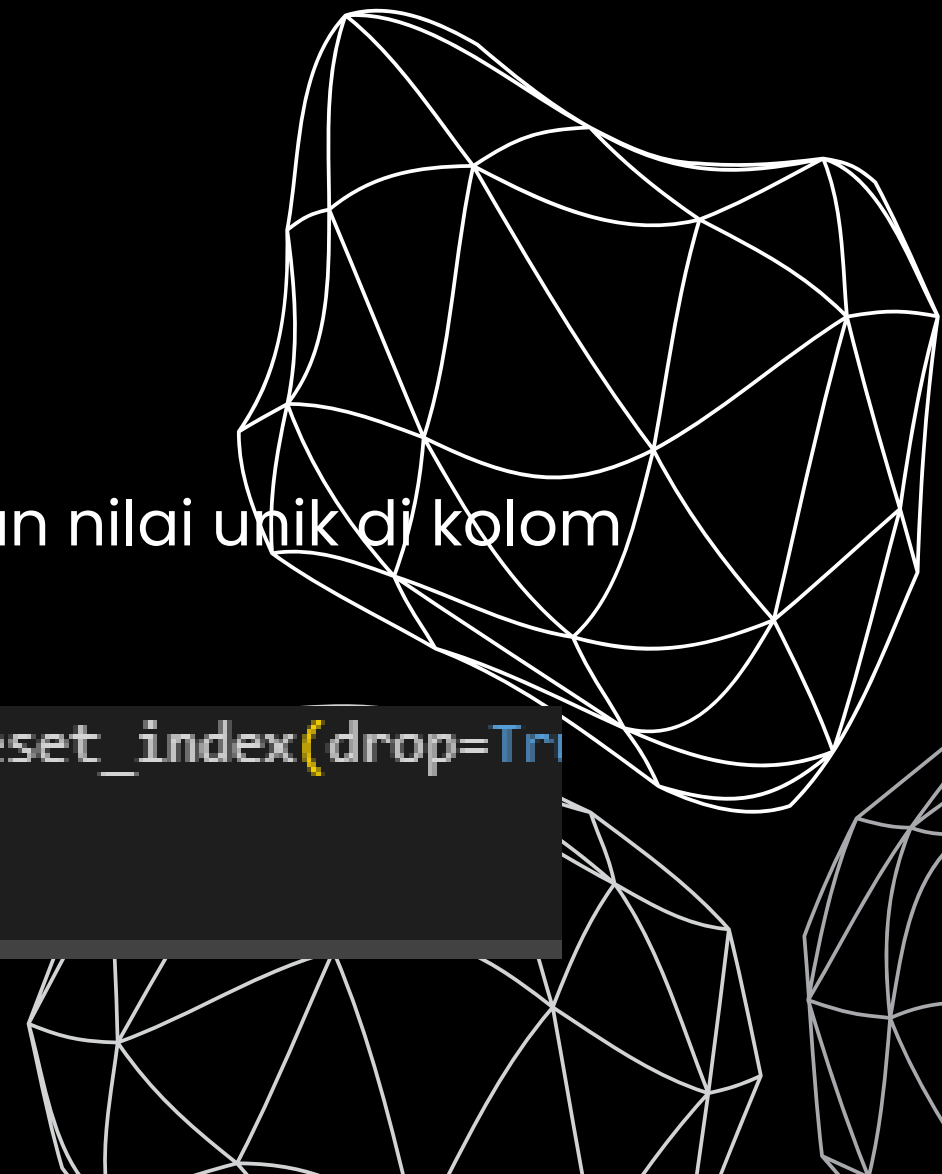
```
Unknown Count: 0
Percentage of Unknown: 0.00%
```

## 10. mengganti 'Interest' menjadi 'Unknown'

```
[ ] merged_data = merged_data[merged_data['Interest'] != 'Unknown']
```

## 11. Proses balancing dengan mengelompokkan data dalam merged\_data berdasarkan nilai unik di kolom 'Personality'.

```
merged_data = merged_data.groupby('Personality').apply(lambda x: x.sample(200)).reset_index(drop=True)
print(f"Jumlah data setelah balancing: {len(merged_data)}")
merged_data['Personality'].value_counts() # Untuk verifikasi
```



# CLASSIFICATION - RF

## 12. cek detail dari merged\_data

```
merged_data = merged_data.head(1000)

merged_data.info()
merged_data.dtypes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype  
---  -
 0   Age                   1000 non-null  float64
 1   Gender                1000 non-null  object  
 2   Education              1000 non-null  int64   
 3   Introversion Score    1000 non-null  float64
 4   Sensing Score         1000 non-null  float64
 5   Thinking Score        1000 non-null  float64
 6   Judging Score         1000 non-null  float64
 7   Interest              1000 non-null  object  
 8   Personality           1000 non-null  object  
dtypes: float64(5), int64(1), object(3)
memory usage: 70.4+ KB
```

0	
Age	float64
Gender	object
Education	int64
Introversion Score	float64
Sensing Score	float64
Thinking Score	float64
Judging Score	float64
Interest	object
Personality	object

## 13. input nilai unik kedalam unique\_interest

```
[ ] unique_interests = merged_data['Interest'].unique()
print(unique_interests)
```

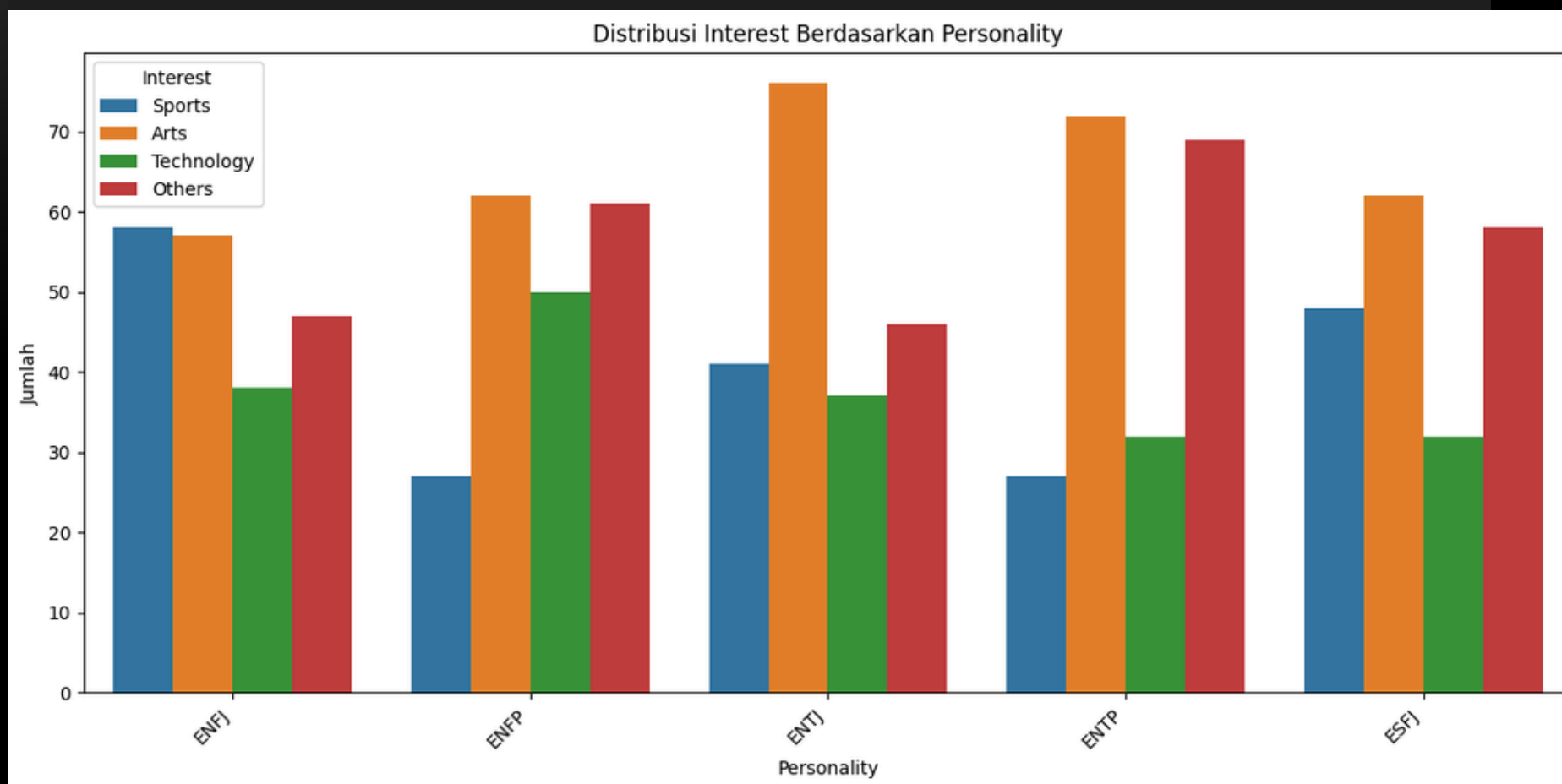
```
→ ['Sports' 'Arts' 'Technology' 'Others']
```



# VISUALISASI

## Distribusi Interest berdasarkan Personality

```
plt.figure(figsize=(12, 6))
sns.countplot(x='Personality', hue='Interest', data=merged_data)
plt.title('Distribusi Interest Berdasarkan Personality')
plt.xlabel('Personality')
plt.ylabel('Jumlah')
plt.xticks(rotation=45, ha='right') # Rotasi label sumbu x agar mudah dibaca
plt.legend(title='Interest')
plt.tight_layout() # Menyesuaikan layout agar tidak terpotong
plt.show()
```



# CLASSIFICATION - RF

13. Membandingkan value 'Personality' yang sudah di prediksi dan nilai yang sebenarnya

```
from tabulate import tabulate
import pandas as pd

# Asumsikan 'encoder' adalah objek LabelEncoder Anda yang sudah dilatih
personality_labels = encoder.classes_

# Buat DataFrame untuk pemetaan
personality_mapping = pd.DataFrame({
    'Personality (Encoded)': range(len(personality_labels)),
    'Personality (Label)': personality_labels
})

# Generate tabel markdown
table_md = tabulate(personality_mapping, headers='keys', tablefmt='pipe')

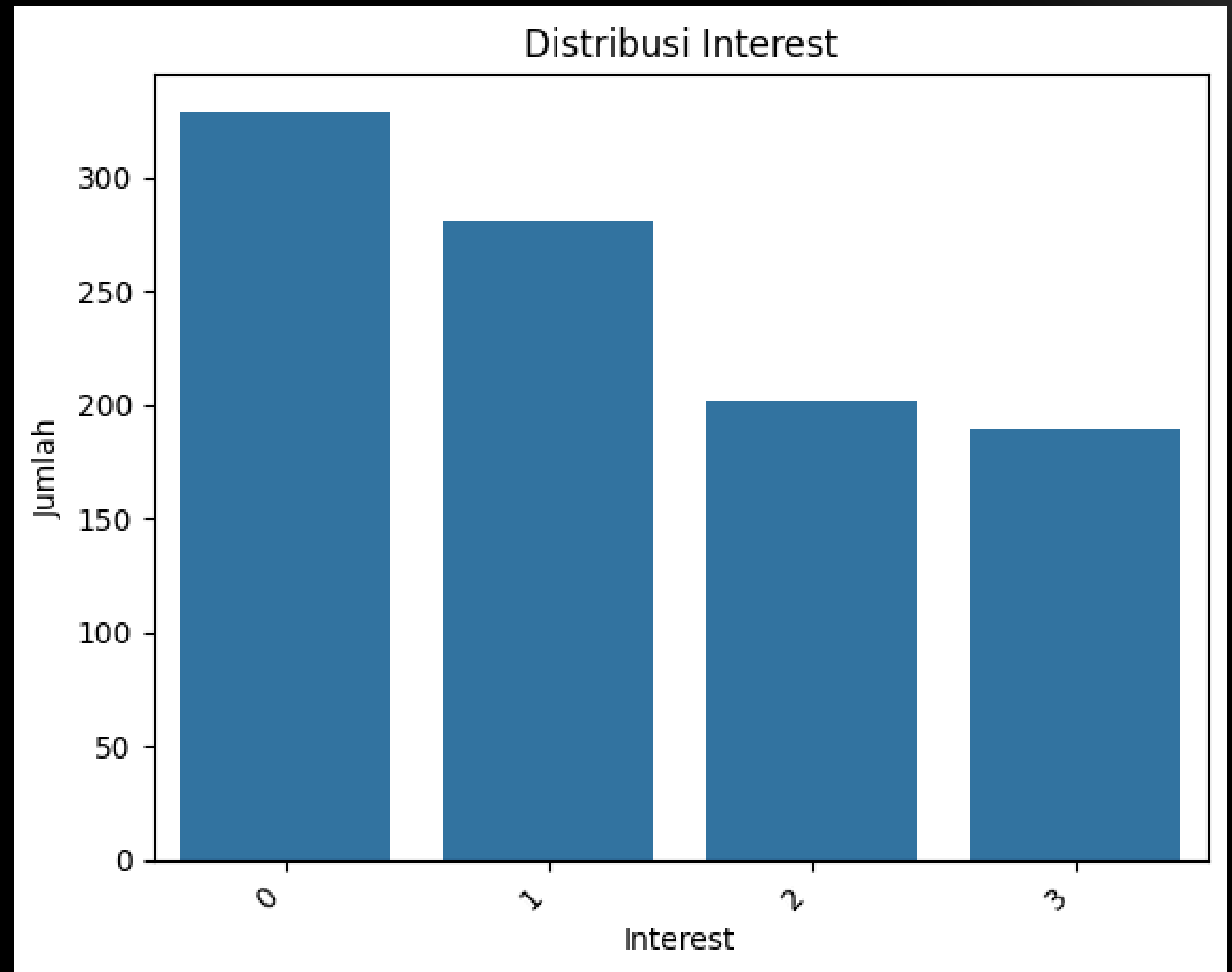
# Print tabel markdown
print(table_md)
```

# VISUALISASI HASIL - RF

## Distribusi Interest

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

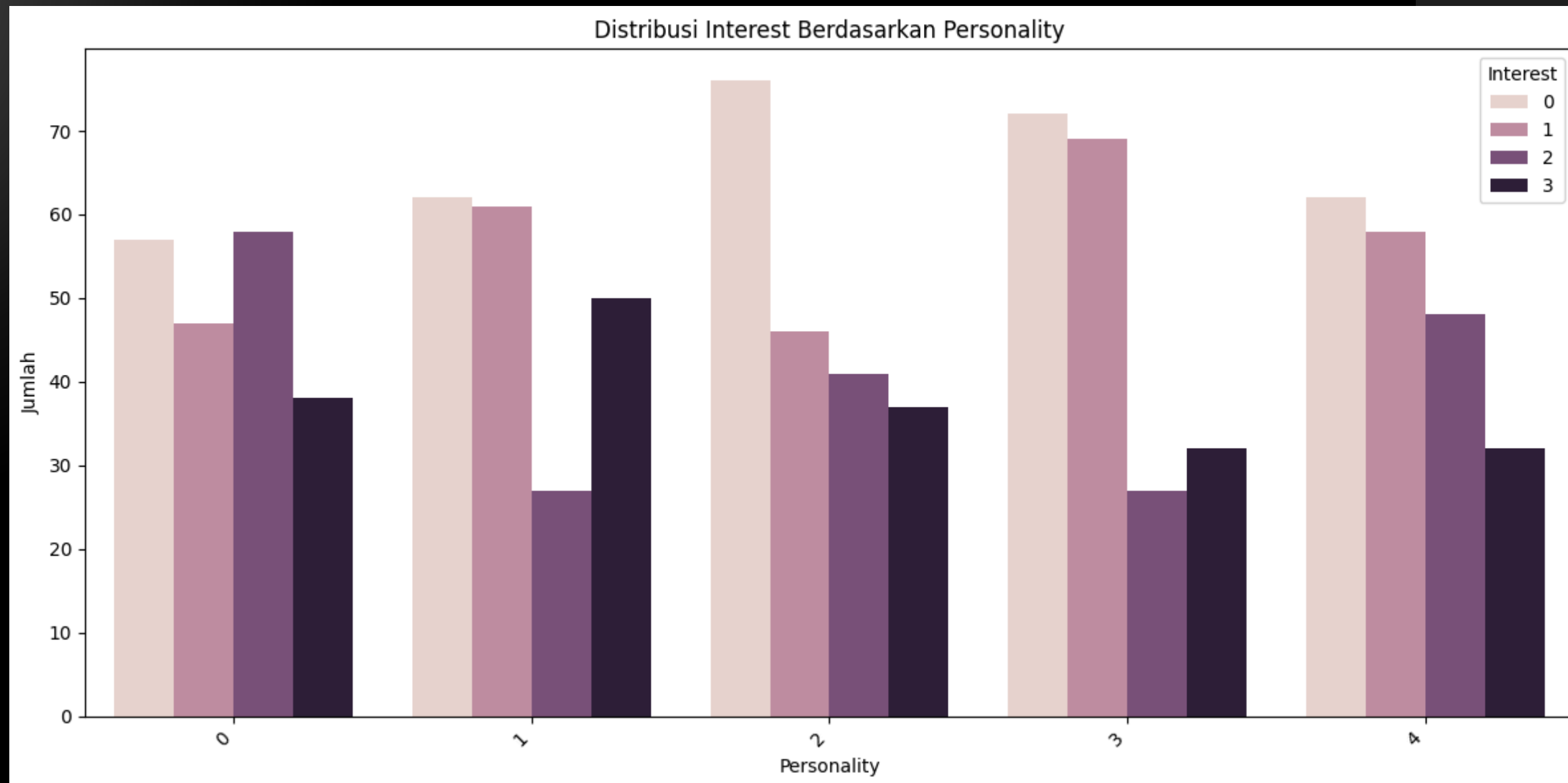
# Visualisasi distribusi kelas target (Interest)
sns.countplot(x='Interest', data=merged_data)
plt.title('Distribusi Interest')
plt.xlabel('Interest')
plt.ylabel('Jumlah')
plt.xticks(rotation=45, ha='right')
plt.show()
```



# VISUALISASI HASIL - RF

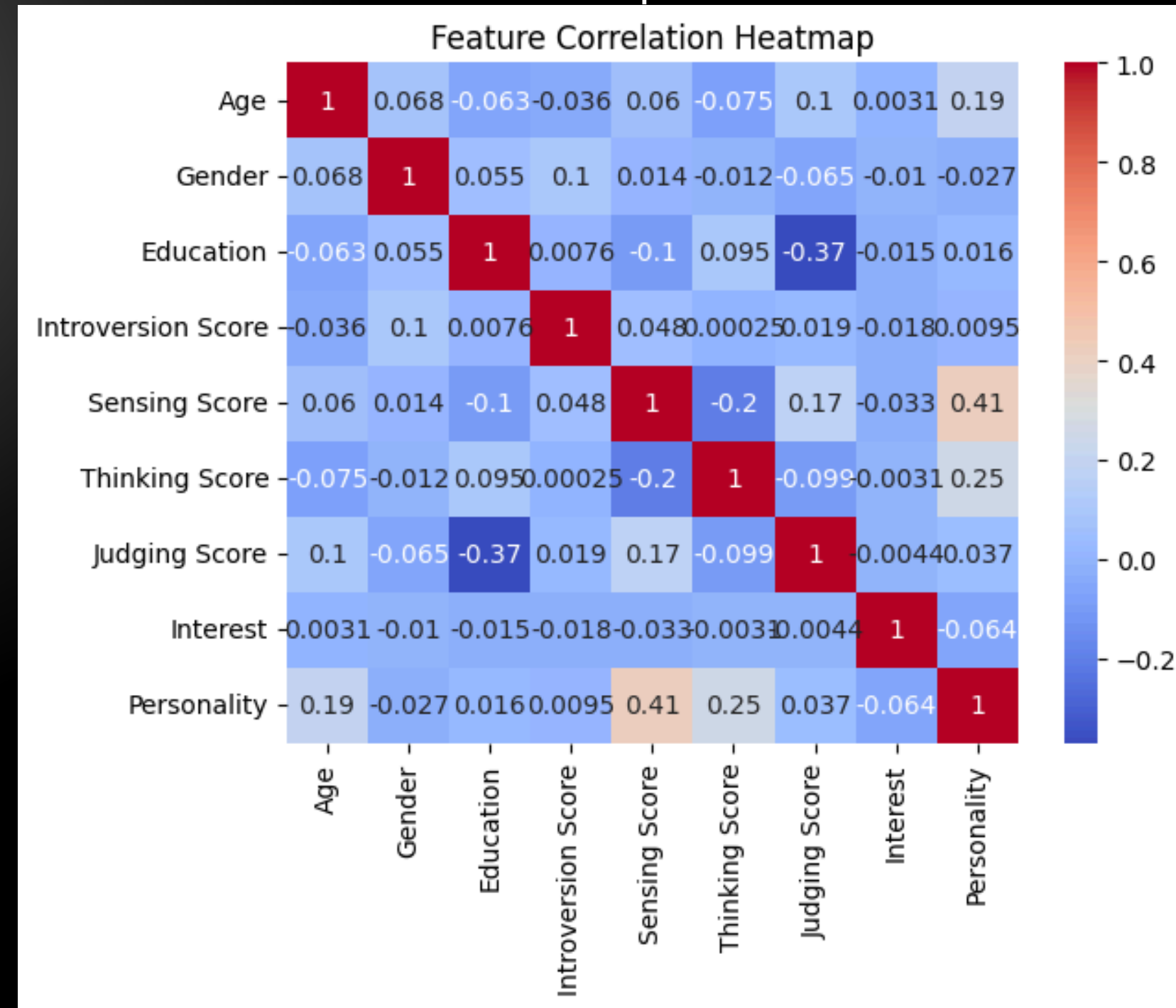
## Distribusi Interest Berdasarkan Personality

```
plt.figure(figsize=(12, 6))
sns.countplot(x='Personality', hue='Interest', data=merged_data)
plt.title('Distribusi Interest Berdasarkan Personality')
plt.xlabel('Personality')
plt.ylabel('Jumlah')
plt.xticks(rotation=45, ha='right') # Rotasi label sumbu x agar mudah dibaca
plt.legend(title='Interest')
plt.tight_layout() # Menyesuaikan layout agar tidak terpotong
plt.show()
```



# VISUALISASI HASIL - RF

Feature Correlation Heatmap



```
correlation_matrix = merged_data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```



# CLASSIFICATION - KNN

1. Split data untuk data yang akan dilatih dan dites

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

2. Normalization

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

3. Latih model KNN dengan  $n=6$ , dan buat prediksi data testing

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Latih model KNN
knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train, y_train)

# Prediksi data testing
y_pred_knn = knn.predict(X_test)
```

# CLASSIFICATION - KNN

4. pilih 5 fitur terbaik dengan  $k = 35$  dan  $k = 5$  untuk memprediksi variabel target ( $y$ ),

```
from sklearn.feature_selection import SelectKBest, f_classif

# Pilih 5 fitur terbaik
selector = SelectKBest(score_func=f_classif, k=35)
X_selected = selector.fit_transform(X, y)

# Tampilkan skor fitur
feature_scores = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
print(feature_scores.sort_values(by='Score', ascending=False))
```

	Feature	Score
	Thinking Score	650.820718
	Judging Score	280.997428
	Education	140.380980
	Sensing Score	131.440264
	Age	18.548709
	Gender	3.681595
	Interest	2.087063
	Introversion Score	0.211376

```
selector = SelectKBest(score_func=f_classif, k=5)
X_selected = selector.fit_transform(X, y)

feature_scores = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})
print(feature_scores.sort_values(by='Score', ascending=False))
```

	Feature	Score
2	Thinking Score	650.820718
3	Judging Score	280.997428
6	Education	140.380980
1	Sensing Score	131.440264
4	Age	18.548709
5	Gender	3.681595
7	Interest	2.087063
0	Introversion Score	0.211376

# CLASSIFICATION - KNN

5. mencari nilai optimal k dengan mengecek akurasi satu persatu

```
k_values = range(1, 50) # Rentang nilai n yang ingin diuji
accuracies = []
```

```
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
```

```
# Menampilkan akurasi untuk setiap nilai k
for k, accuracy in zip(k_values, accuracies):
    print(f"k={k}, accuracy={accuracy}")
```

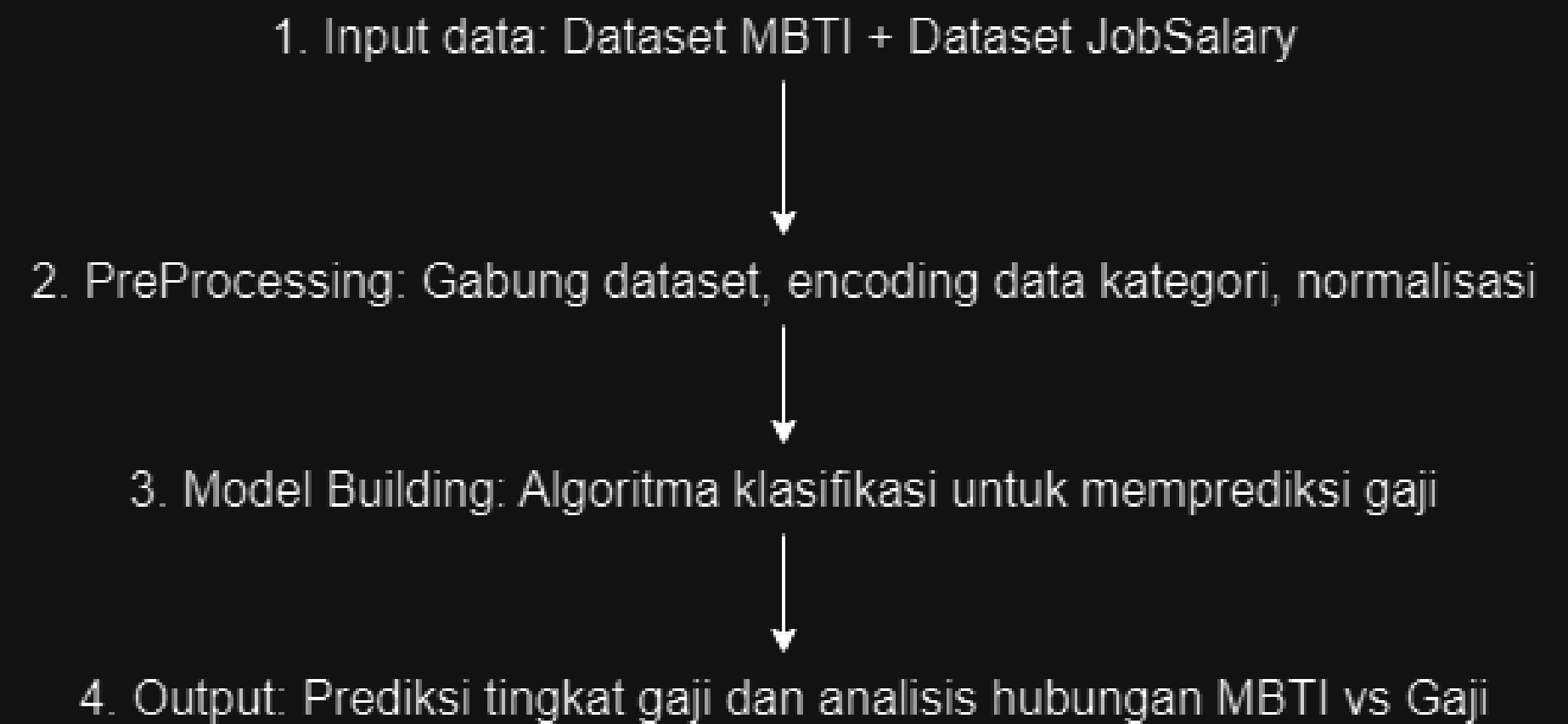
```
optimal_k = k_values[accuracies.index(max(accuracies))]
print(f"Nilai k optimal: {optimal_k}")
```

```
Nilai k optimal: 28
```

# SISTEM YANG DIBANGUN

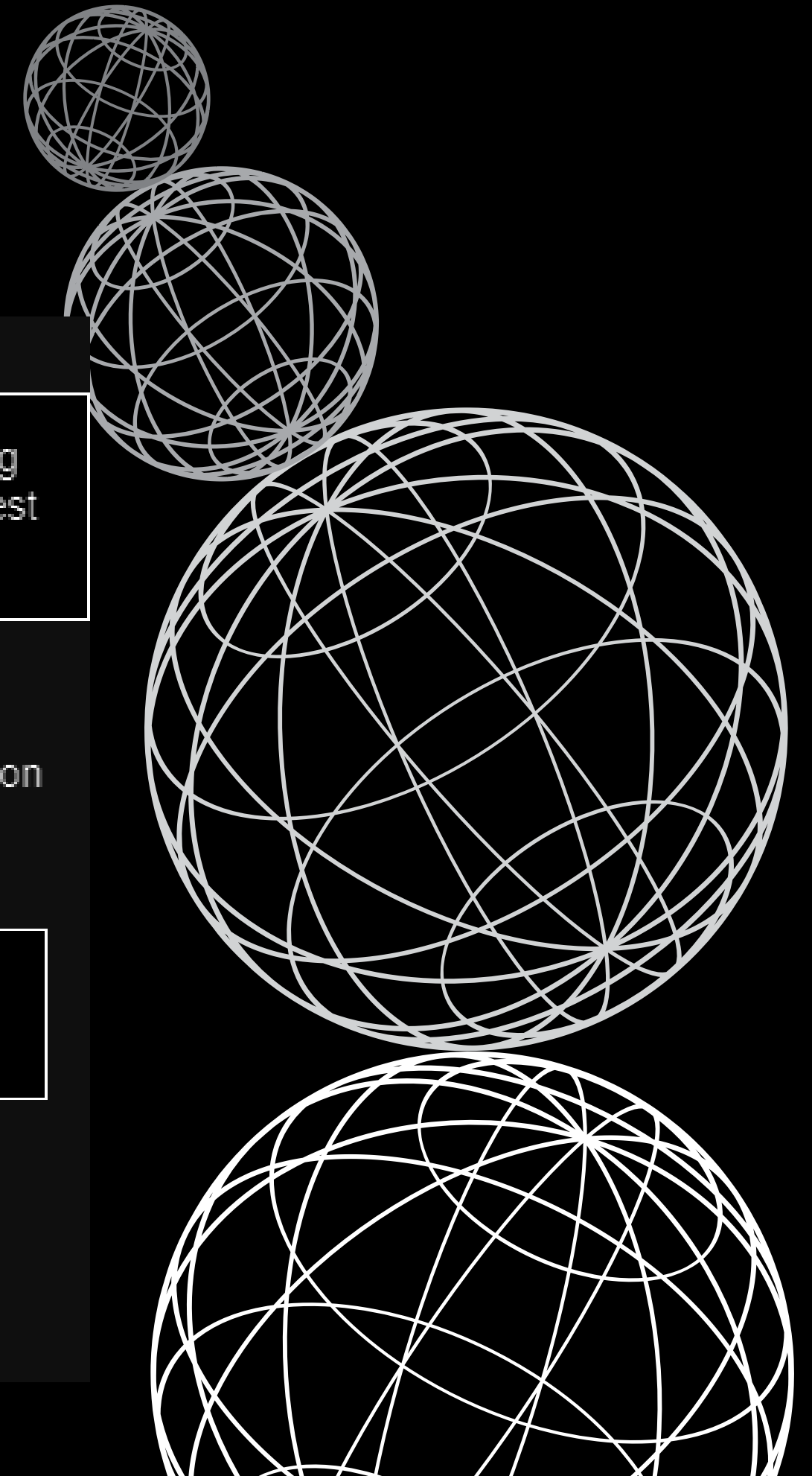
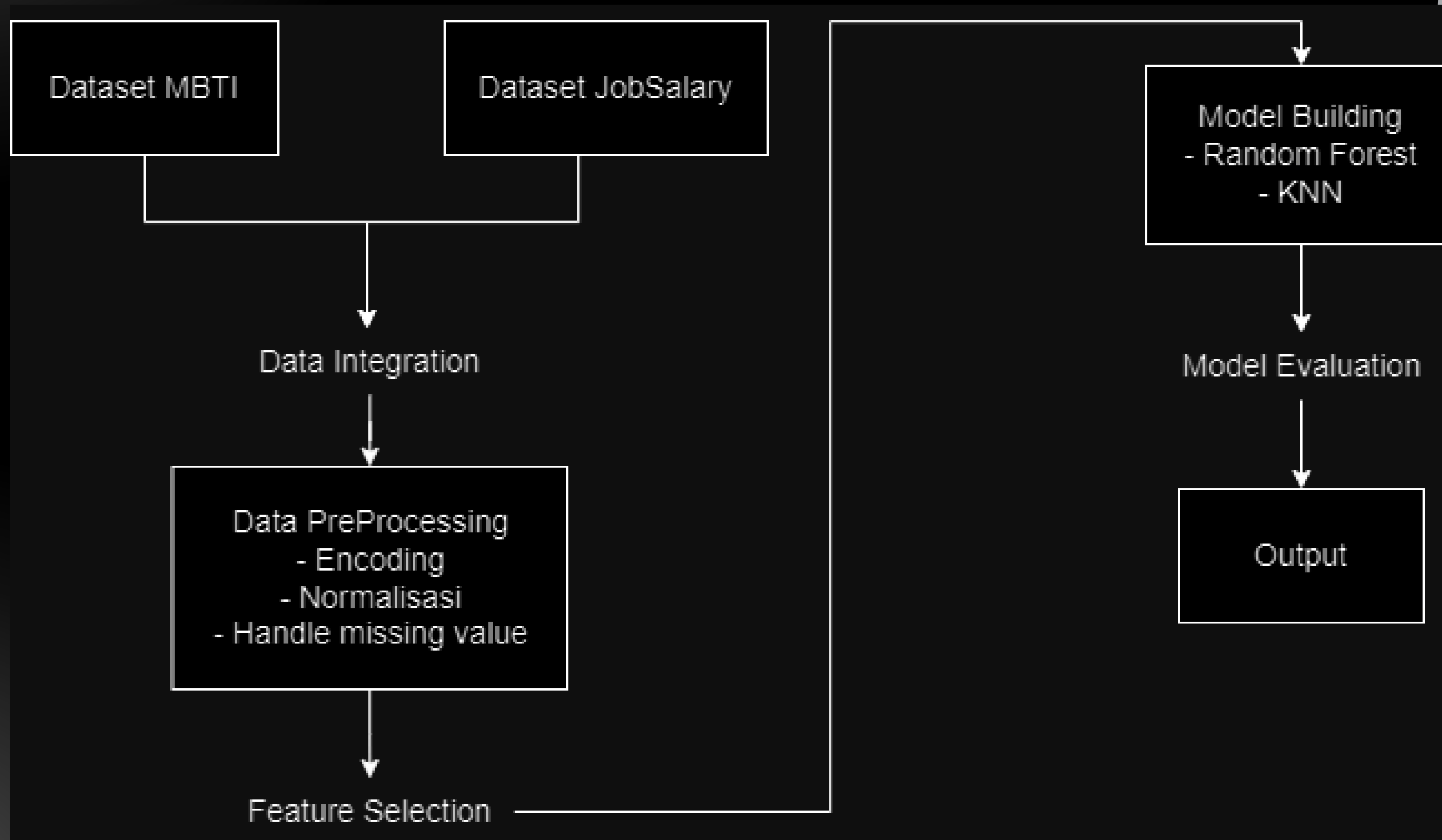
## Sistem yang dibangun bertujuan untuk:

1. Menggabungkan informasi kepribadian individu (dari dataset MBTI) dengan data pekerjaan dan gaji (dari dataset Job Salary).
2. Menganalisis hubungan antara tipe kepribadian MBTI dan gaji.
3. Menggunakan algoritma klasifikasi untuk memprediksi tingkat gaji berdasarkan karakteristik individu (usia, gender, kepribadian, dll.).



# SISTEM YANG DIBANGUN

## Diagram alur sistem





# SISTEM YANG DIBANGUN

## Tools yang digunakan

Platform:

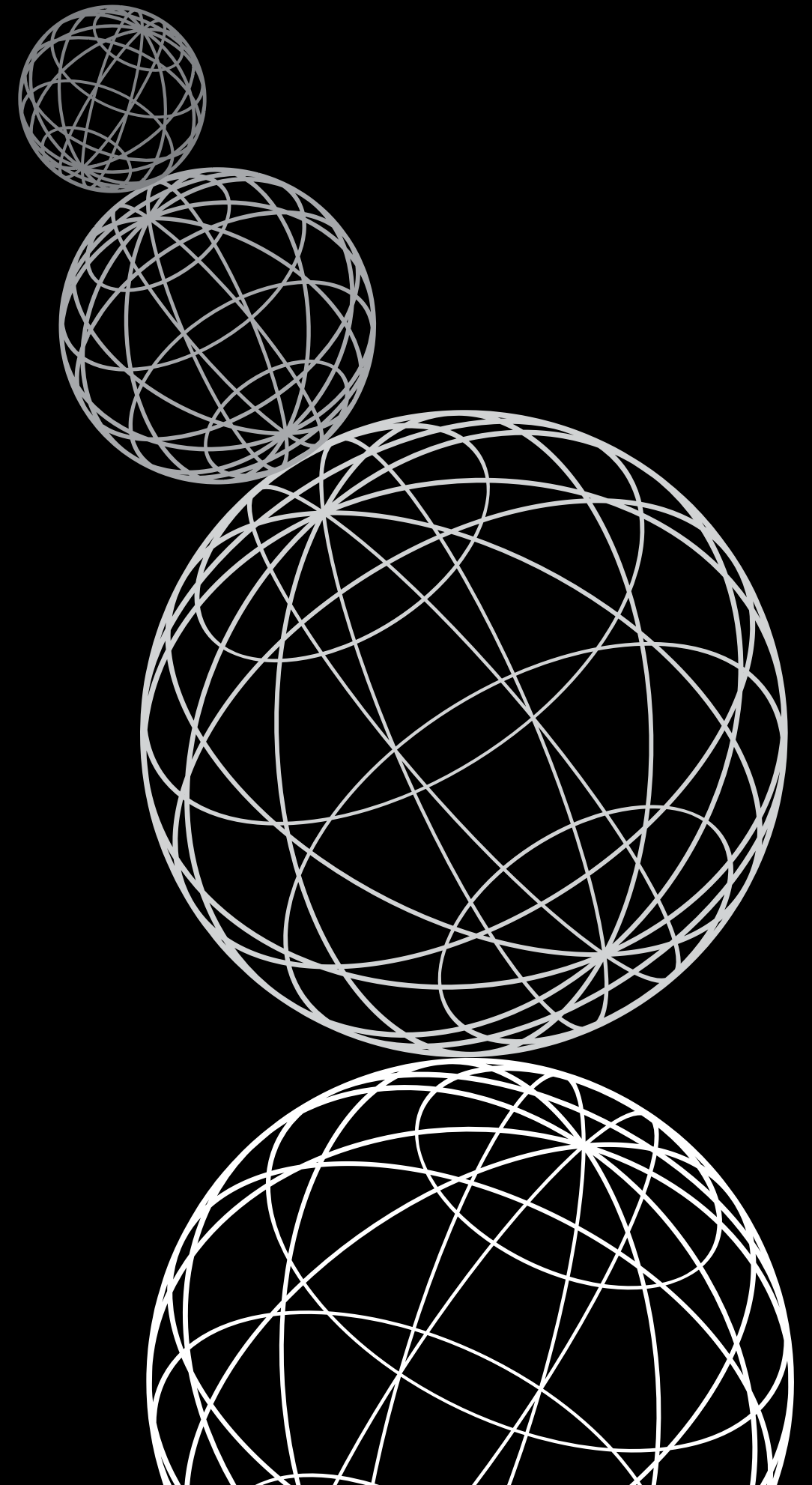
- Python (Jupyter Notebook atau Google Colab)

Library yang Digunakan:

- Pandas: Untuk pengolahan data
- Scikit-learn: Untuk implementasi algoritma klasifikasi (Random Forest, Logistic Regression)
- Matplotlib dan Seaborn: Untuk visualisasi data

Proses Automasi:

- Seluruh pipeline mulai dari preprocessing hingga evaluasi dilakukan menggunakan script Python



# Metode

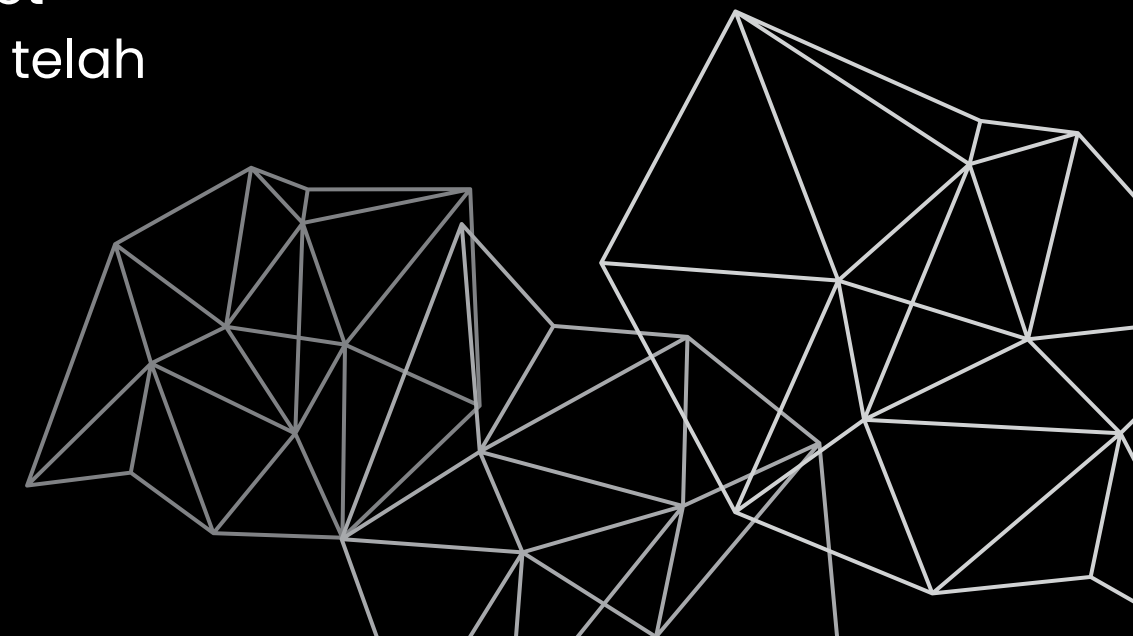
Algoritma yang digunakan

## **K-Nearest Neighbors (KNN)**

Algoritma KNN dipilih karena kesederhanaannya dan kemampuannya yang baik dalam menangani tugas klasifikasi pada dataset dengan fitur numerik, seperti data gaji dan pekerjaan yang telah diencode. Sebelum diaplikasikan, data distandardisasi menggunakan StandardScaler untuk memastikan performa optimal KNN.

## **Random Forest**

Algoritma Random Forest dipilih karena kemampuannya yang unggul dalam mengurangi risiko overfitting dengan menggabungkan banyak pohon keputusan. Metode ini efektif untuk dataset dengan distribusi data yang tidak merata serta kombinasi fitur numerik dan kategorikal yang telah diencode.



# RESULT KNN

## K-Nearest Neighbors (KNN)

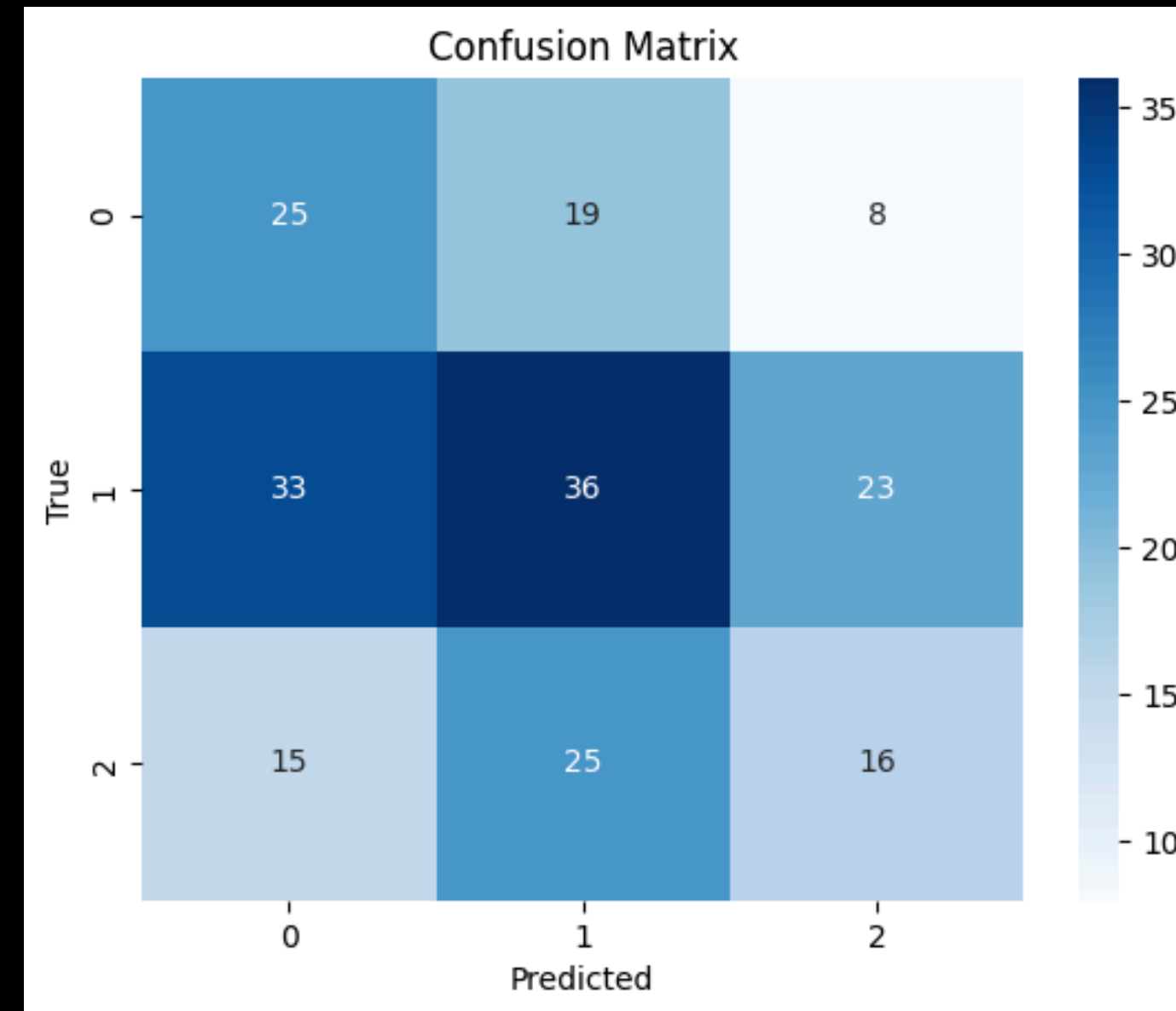
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive")
Accuracy: 0.48
```

	feature	importance	p_value
4	Job_encoded	0.13150	0.465347
2	Thinking Score	0.05895	0.455446
1	Sensing Score	0.05110	0.495050
6	Interest_Sports	0.02510	0.435644
3	Judging Score	0.02505	0.504950
7	Interest_Technology	0.01745	0.495050
5	Interest_Arts	0.01015	0.445545
0	Introversion Score	0.00340	0.514851

```
[0.0034 0.0511 0.05895 0.02505 0.1315 0.01015 0.0251 0.01745]
```

Features used in training: Index(['Introversion Score', 'Sensing Score', 'Thinking Score', 'Judging Score', 'Job\_encoded', 'Interest\_Arts', 'Interest\_Sports', 'Interest\_Technology'], dtype='object')

Hasil Confiusion Matrix

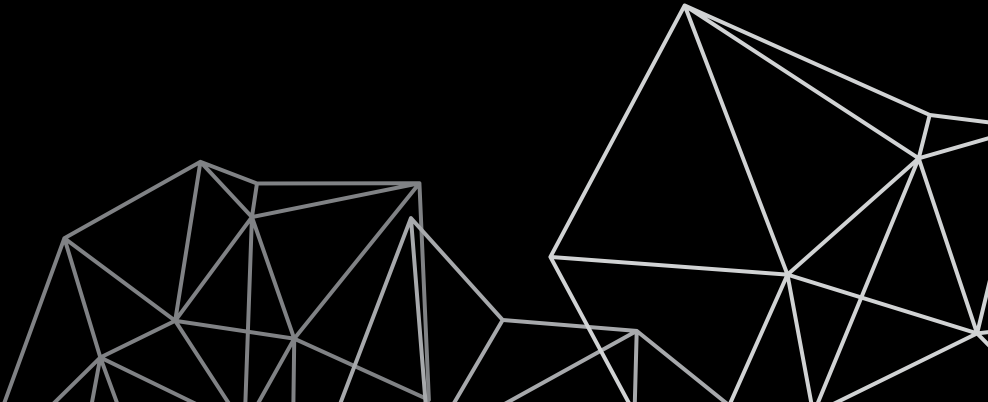


Hasil akurasi dari kasus ini adalah 0.46

# RESULT KNN

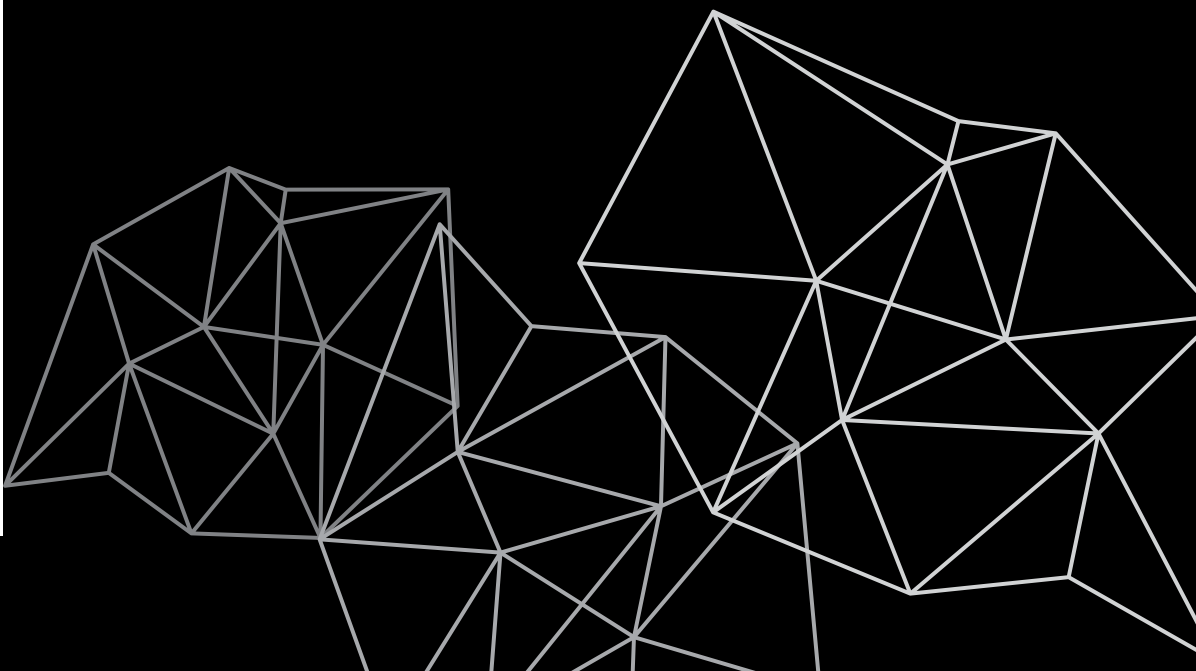
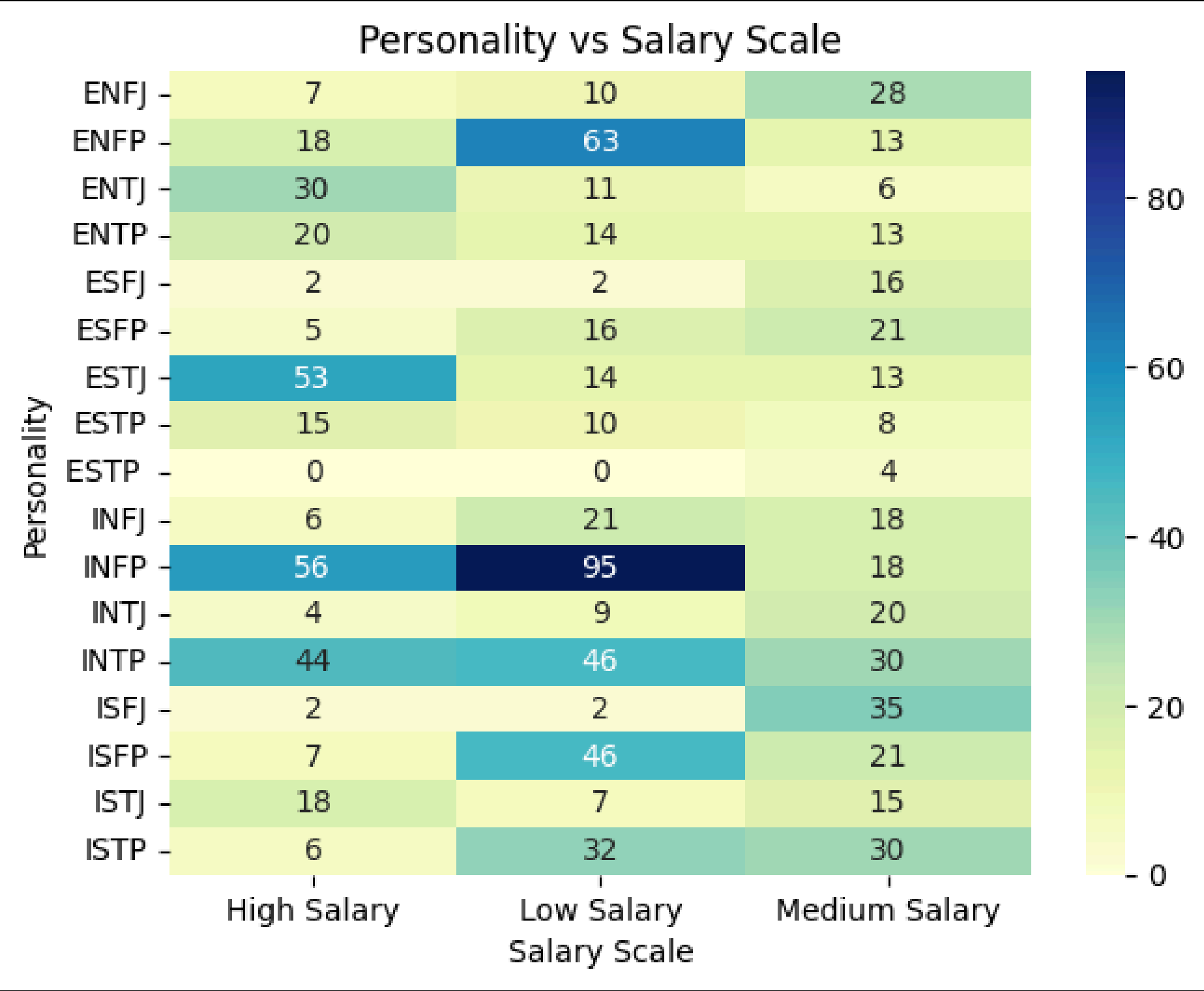
Prediksi Data Testing

	Actual	Predicted
521	Low Salary	High Salary
737	Low Salary	Medium Salary
740	Low Salary	High Salary
660	Medium Salary	Low Salary
411	Low Salary	Medium Salary
..	...	...
408	Low Salary	High Salary
332	Medium Salary	High Salary
208	Medium Salary	Low Salary
613	Medium Salary	Low Salary
78	Low Salary	High Salary
[200 rows x 2 columns]		



# RESULT KNN

visualisation heatmap

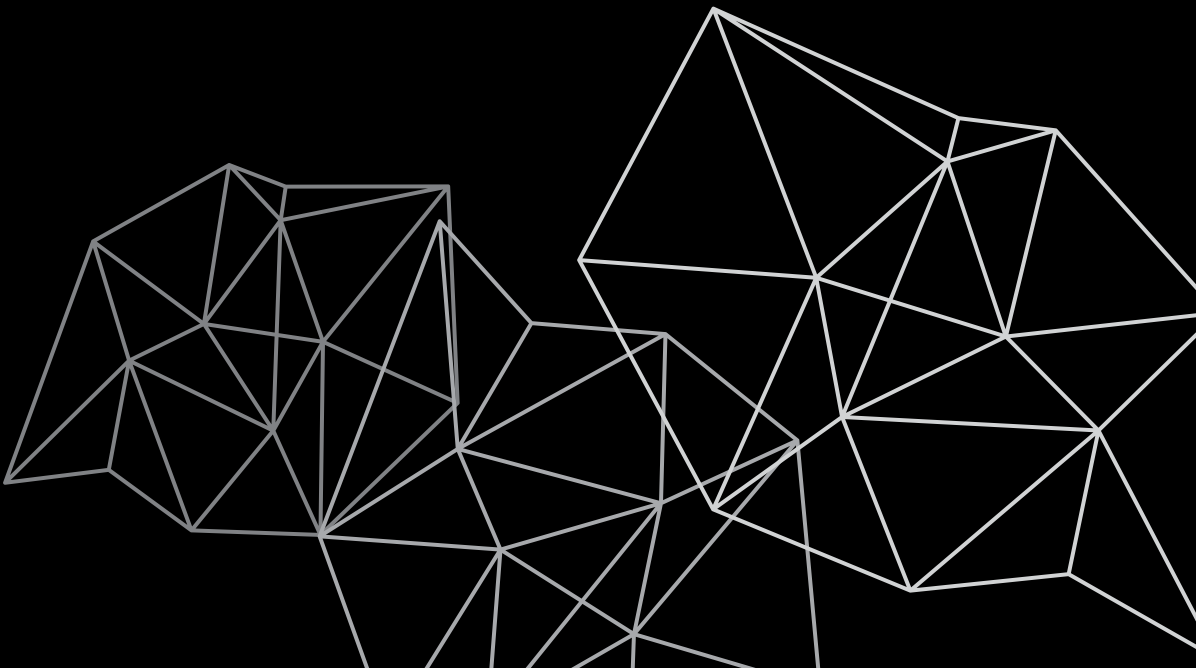


# RANDOM FOREST

## Random Forest

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Accuracy: 0.94
feature importance p_value
4 Job_encoded 0.57405 0.514851
2 Thinking Score 0.02330 0.524752
1 Sensing Score 0.01875 0.544554
3 Judging Score 0.01225 0.455446
7 Interest_Technology 0.00440 0.643564
6 Interest_Sports 0.00125 0.425743
0 Introversion Score 0.00075 0.415842
5 Interest_Arts -0.00565 0.702970
[ 0.00075 0.01875 0.0233 0.01225 0.57405 -0.00565 0.00125 0.0044 ]
Features used in training: Index(['Introversion Score', 'Sensing Score', 'Thinking Score',
    'Judging Score', 'Job_encoded', 'Interest_Arts', 'Interest_Sports',
    'Interest_Technology'],
    dtype='object')
Features used in permutation importance: ['Introversion Score', 'Sensing Score', 'Thinking Score', 'Judging Score', 'Job_encoded', 'Interest_Arts', 'Interest_Sports', 'Interest_Technology']
Shape of X test: (200, 8)
```

Model Random Forest mencapai akurasi 0.94 menunjukkan performa yang sangat baik dalam memprediksi data uji.



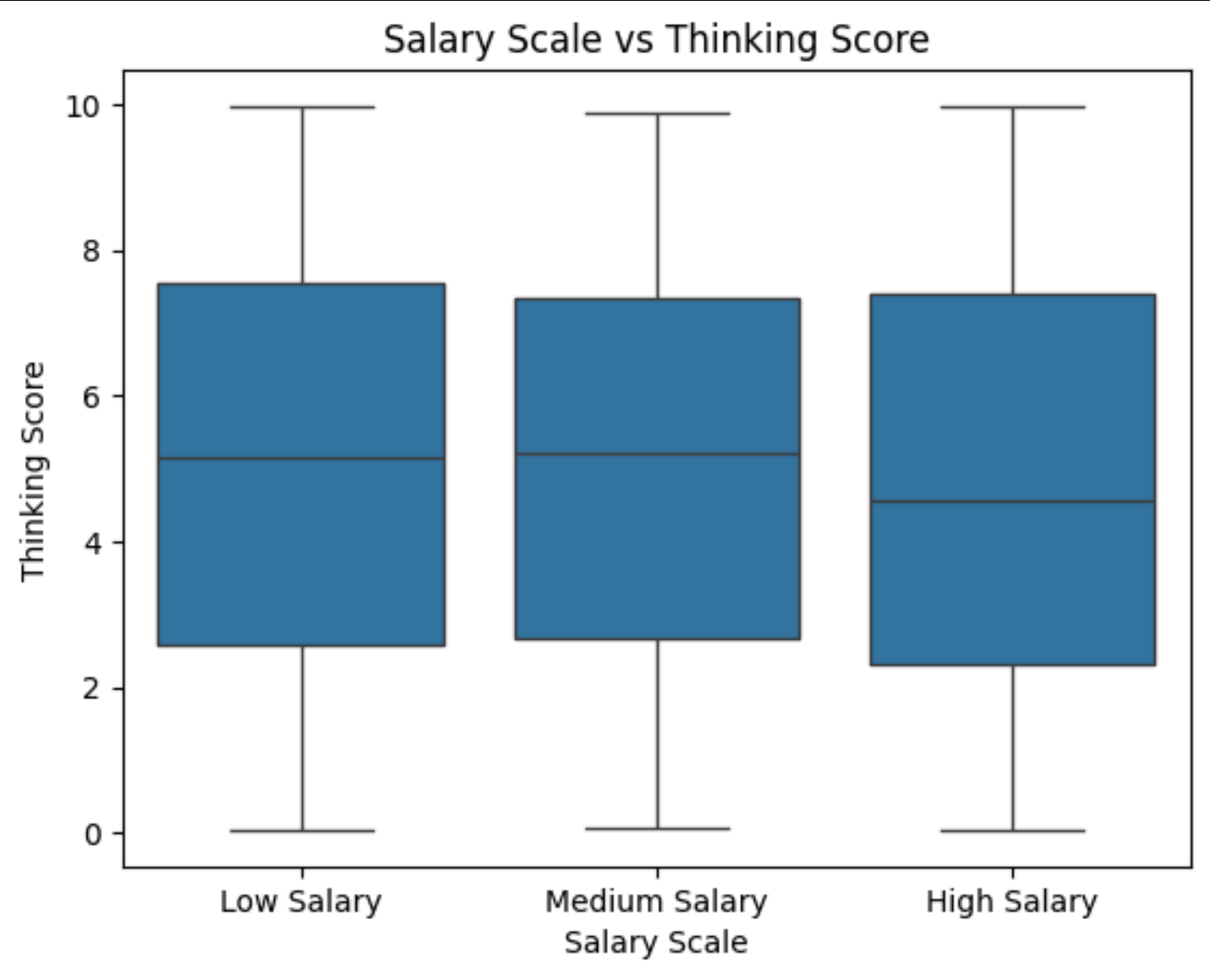


# RANDOM FOREST

## Prediksi Data Testing

	Actual	Predicted
521	Low Salary	Low Salary
737	Low Salary	Medium Salary
740	Low Salary	Low Salary
660	Medium Salary	Low Salary
411	Low Salary	Low Salary
..	...	...
408	Low Salary	Low Salary
332	Medium Salary	Low Salary
208	Medium Salary	Medium Salary
613	Medium Salary	Medium Salary
78	Low Salary	Low Salary
[200 rows x 2 columns]		

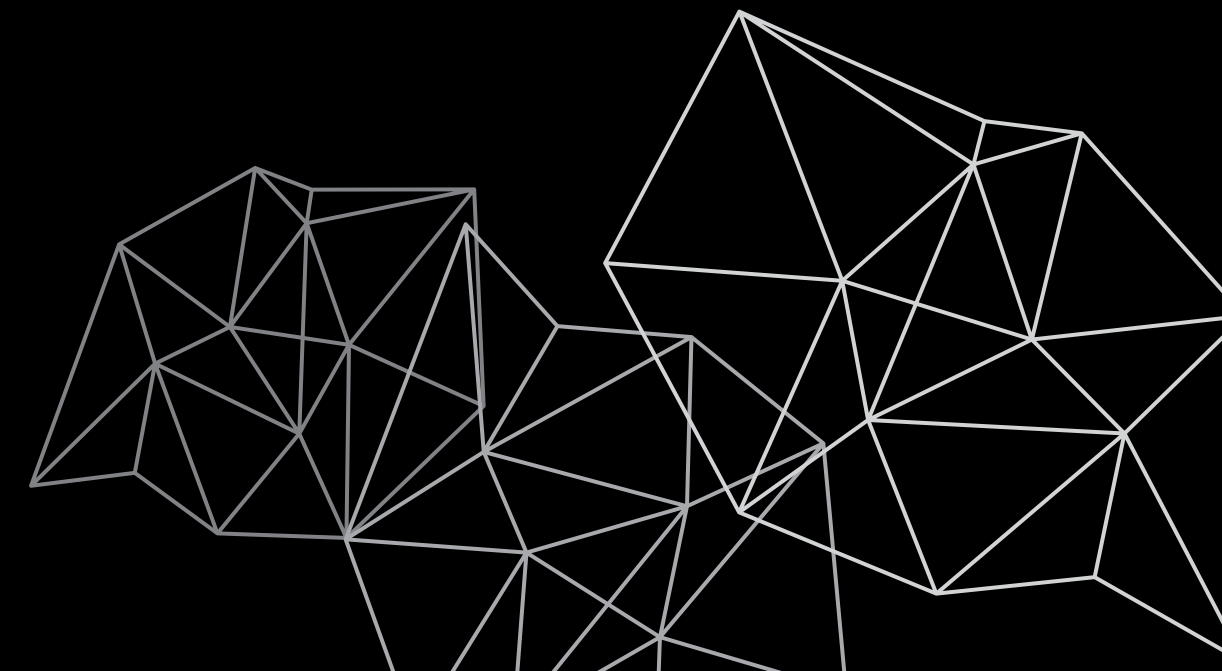
hasil dari tabel ini menunjukkan prediksi label (Predicted) dibandingkan dengan label aktual (Actual) untuk data uji.





# KESIMPULAN

Dalam penelitian ini, ditemukan bahwa terdapat hubungan signifikan antara tipe kepribadian, pekerjaan, dan skala gaji. Model Random Forest menunjukkan hasil prediksi yang lebih akurat dibandingkan dengan K-Nearest Neighbors (KNN). Selain itu, tipe kepribadian tertentu cenderung mempengaruhi skala gaji dan jenis pekerjaan yang dipilih oleh individu.



The background is a dark gradient with white, wavy, wireframe-like lines that create a sense of depth and movement. These lines are concentrated in the corners, framing the central text.

**THANK YOU**