

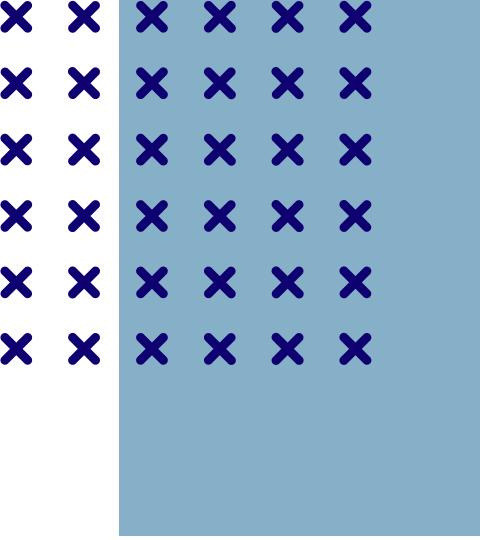


# FINAL PROJECT

**Optimalisasi Klasifikasi *Sleep Disorder*  
Menggunakan Metode *Decision Tree*,  
*Random Forest*, *K-Nearest Neighbor*  
(KNN), dan *Support Vector Machine* (SVM)**

Dipresentasikan oleh kelompok 9





# OUR TEAM



**Yola Darma Dhaifulla**

5003211023



**Salma Zaura Baraza**

5003211151



# TABLE OF CONTENT

01	PENDAHULUAN	04	HASIL DAN PEMBAHASAN
02	TINJAUAN PUSTAKA	05	KESIMPULAN DAN SARAN
03	METODOLOGI	06	LAMPIRAN



# PENDAHULUAN

# Latar Belakang

- 1** Insomnia dan sleep apnea mempengaruhi kualitas tidur dan kesehatan seseorang.
- 2** Penggunaan machine learning untuk mengklasifikasikan gangguan tidur dengan efisien.
- 3** Dilakukan untuk membangun model prediktif berdasarkan variabel seperti durasi tidur, kualitas tidur, tingkat stres, dan aktivitas fisik.

# Tujuan dan Manfaat

- 1** Mengidentifikasi gangguan tidur dengan metode Decision Tree, Random Forest, K-Nearest Neighbor (KNN), dan Support Vector Machine (SVM).
- 2** Menerapkan ilmu statistika, memberikan informasi tentang gangguan tidur kepada masyarakat, serta menjadi referensi bagi peneliti yang menggunakan metode serupa.
- 3** Menggunakan algoritma-algoritma tersebut untuk analisis faktor-faktor utama yang mempengaruhi gangguan tidur.

# TINJAUAN PUSTAKA

## Sleep Disorder

Gangguan tidur merupakan sekelompok kondisi yang ditandai dengan adanya gangguan dalam kuantitas, kualitas, atau durasi tidur. Faktor-faktor yang dapat mempengaruhi gangguan tidur meliputi kualitas tidur, durasi tidur, tingkat stres, pekerjaan, tingkat aktivitas, dan faktor lainnya. Dalam data ini, gangguan tidur terdiri dari insomnia, sleep apnea, dan tidak ada gangguan tidur

## Decision Tree

Decision Tree merupakan salah satu algoritma *supervised learning* yang melakukan prediksi menggunakan struktur pohon. Decision tree terbuat dari tiga node yaitu *leaf node*, lalu *root node* yang merupakan titik awal dari suatu *decision tree*, dan yang terakhir adalah simpul perantara atau *internal node* yang berhubungan dengan suatu pengujian.

## **Random Forest**

Metode random forest merupakan pendekatan yang dapat meningkatkan akurasi dengan melakukan pemilihan simpul anak secara acak untuk setiap node. Metode ini digunakan untuk membangun pohon keputusan yang terdiri dari root node, internal node, dan leaf node dengan memilih atribut dan data secara acak sesuai aturan yang ditetapkan. Pohon keputusan dimulai dengan menghitung nilai entropy untuk menentukan tingkat ketidakmurnian atribut dan nilai information gain.

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

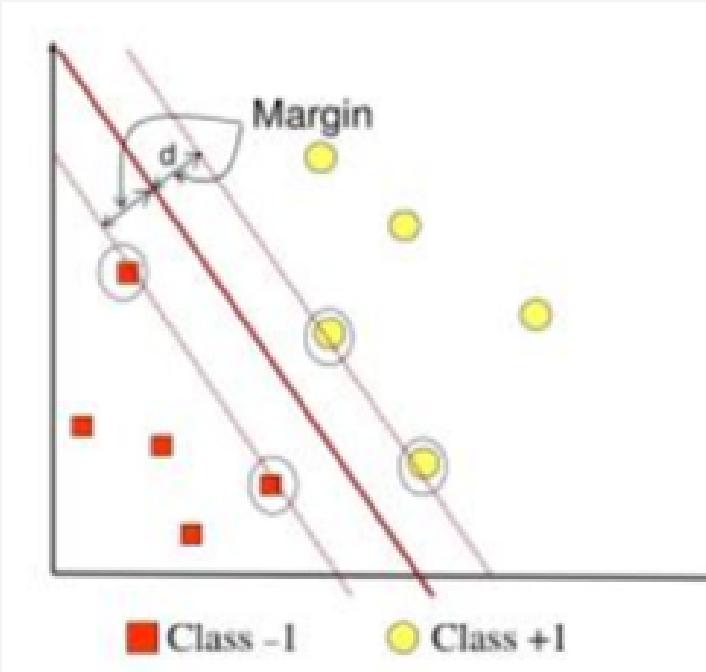
K-Nearest Neighbor (KNN) merupakan metode klasifikasi terhadap objek berdasarkan data pembelajaran (neighbor) yang jaraknya paling dekat dengan objek tersebut. Dekat atau jauhnya neighbor biasanya dihitung menggunakan jarak Euclidean. Metode KNN terbagi menjadi dua fase: pembelajaran (training) dan klasifikasi atau pengujian (testing). Pada fase pembelajaran, algoritma ini hanya menyimpan vektor-vektor fitur dan klasifikasi dari data pelatihan. Pada fase klasifikasi, fitur-fitur yang sama dihitung untuk data yang akan diuji (dengan klasifikasi yang tidak diketahui). Jarak antara vektor baru ini dan seluruh vektor data pelatihan dihitung, kemudian diambil sejumlah  $k$  neighbor terdekat. Perhitungan jarak tetangga menggunakan algoritma Euclidean sebagaimana ditunjukkan pada persamaan berikut.

$$euc = \sqrt{(a_1 - b_1)^2 + \dots + (a_n - b_n)^2}$$

dimana  $a=a_1,a_2,\dots,a_n$  dan  $b=b_1,b_2,\dots,b_n$  mewakili nilai atribut dari 2 record. untuk atribut dengan nilai kategori.

# Support Vector Machine (SVM)

Support Vector Machine (SVM) adalah metode pembelajaran supervised yang pertama kali diperkenalkan oleh Vapnik pada tahun 1995 dan telah terbukti sukses dalam melakukan prediksi, baik untuk kasus regresi maupun klasifikasi.



Tujuan utama:

Membangun Optimal Separating Hyperplane yang berfungsi sebagai pemisah optimal antara dua kelas pada input space.

$$x_i w + b \geq +1, y_i = +1 \quad \text{untuk kelas positif}$$
$$x_i w + b \leq -1, y_i = -1 \quad \text{untuk kelas negatif}$$

Hyperplane optimum dapat dicari dengan nilai margin maksimum dapat dirumuskan menjadi masalah optimasi konstrain ke dalam formula lagrange

$$L_{pri}(x, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i [y_i(x_i^T w + b) - 1]$$

# **Support Vector Machine (SVM)**

Apabila data training tidak dapat dipisahkan dengan linier, sehingga memerlukan solusi untuk memetakan input ke ruang yang berdimensi lebih tinggi. Fungsi kernel mampu menyelesaikan permasalahan SVM non-linier

- Linier

$$K(x_i, x_j) = x_i^T x_j$$

- Polinomial

$$K(x_i, x_j) = (x_i^T x_j + 1)^d$$

- Radial Basis Function

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

# Confusion Matrix

Confusion matrix diartikan sebagai pengukuran performa pada machine learning dengan output berupa dua kelas atau lebih.

Confusion matrix	Classification	
	Positive (+)	Negative (-)
Positive (+)	True Positive	False Negative
Negative (-)	False Positive	True Negative

Performa machine learning yang bagus atau tidak dari confusion matrix dengan melakukan perhitungan *accuracy*, *precision*, *recall*, dan *f1-score*.

$$\text{Accuracy} = \frac{TP+TN}{(TP+FP+FN+TN)}$$

$$\text{Recall} = \frac{TP}{(TP+FN)} \times 100\%$$

$$\text{Precision} = \frac{TP}{(TP+FP)} \times 100\%$$

$$F1 - score = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

## ROC dan AUC

Receiver Operating Characteristic (ROC) memberikan visualisasi hubungan antara *False Positive* pada sumbu x dan *True Positive* pada sumbu y. Oleh karena itu, ROC pada dasarnya adalah visualisasi dari *confusion matrix*, yang mencakup semua *confusion matrix* yang mungkin dengan menggunakan threshold dari 0 hingga 1 untuk semua nilai *False Positive* dan *True Positive*. Area Under Curve (AUC) adalah nilai luas di bawah kurva ROC. Nilai AUC biasanya digunakan untuk membandingkan berbagai model, dengan model terbaik adalah yang memiliki nilai AUC tertinggi.

# METODOLOGI

# Sumber Data

- Data yang digunakan dalam penelitian ini merupakan data sekunder berjudul "**Sleep Health and Lifestyle Dataset**" yang bersumber dari platform Kaggle.
- Terdapat **375 observasi** dan **13 variabel** yang mencakup berbagai variabel terkait tidur dan kebiasaan sehari-hari.

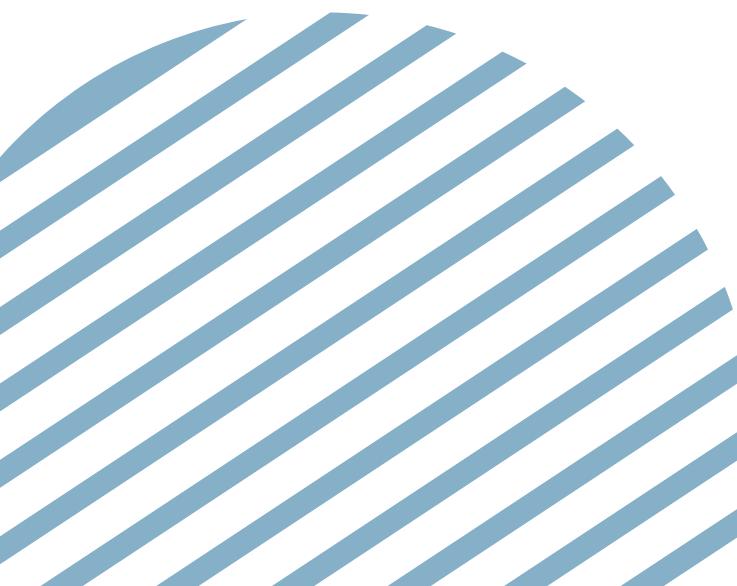
## Variabel Penelitian

Variabel	Keterangan	Deskripsi	Skala
Y	Sleep Disorder	0 : Insomnia (seseorang yang mengalami kesulitan untuk tertidur atau tetap tertidur). 1 : None (seseorang yang tidak mengalami gangguan tidur). 2 : Sleep Apnea (seseorang yang mengalami jeda pernapasan saat tidur).	Nominal
X <sub>1</sub>	Age	Usia dalam tahun	Rasio
X <sub>2</sub>	Occupation	Pekerjaan atau profesi	Nominal
X <sub>3</sub>	Sleep Duration	Jumlah jam tidur per hari	Rasio
X <sub>4</sub>	Quality of Sleep	Kualitas tidur (1-10)	Ordinal
X <sub>5</sub>	Physical Activity Level	Jumlah menit seseorang melakukan aktivitas fisik	Rasio
X <sub>6</sub>	BMI Category	Kategori BMI seseorang	Nominal
X <sub>7</sub>	Heart Rate	Detak jantung per menit	Rasio
X <sub>8</sub>	Daily Steps	Jumlah langkah per hari	Rasio
X <sub>9</sub>	BP Low	Pengukuran tekanan diastolik	Rasio



# Struktur Data

Y	X <sub>1</sub>	X <sub>2</sub>	X <sub>3</sub>	X <sub>4</sub>	X <sub>5</sub>	X <sub>6</sub>	X <sub>7</sub>	X <sub>8</sub>	X <sub>9</sub>
2	29	1	6,0	6	30	0	70	8000	80
2	30	4	6,4	5	35	1	78	4100	86
0	30	4	6,4	5	35	1	78	4100	86
2	31	1	7,7	7	75	0	70	8000	80
0	33	1	6,0	6	30	0	72	5000	80
...	...	...	...	...	...	...	...	...	...
1	31	4	7,9	8	75	1	69	6800	76
1	37	0	7,2	8	60	0	68	7000	75
1	53	2	8,5	9	30	0	65	5000	80
1	32	1	6,2	6	30	0	72	5000	80

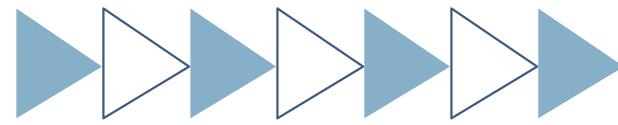


## Langkah Analisis

1. Merumuskan masalah dan tujuan penelitian
2. Mengumpulkan data penelitian
3. Melakukan eksplorasi dan visualisasi data
4. Melakukan preprocessing data yang mencakup pengecekan data duplikat, missing value, dan outliers.
5. Melakukan feature selection menggunakan metode yang sesuai dengan model yang akan dibentuk
6. Melakukan klasifikasi dengan menggunakan metode decision tree
7. Melakukan klasifikasi dengan menggunakan metode random forest
8. Melakukan klasifikasi dengan menggunakan metode K-Nearest Neighbors (KNN)
9. Melakukan klasifikasi dengan menggunakan metode support vector machine (SVM)
10. Melakukan analisis hasil dari seluruh model
11. Membandingkan nilai akurasi dari seluruh model
12. Memilih model terbaik

# **HASIL DAN PEMBAHASAN**

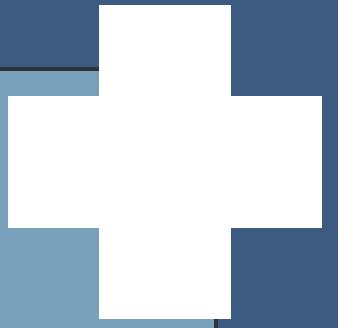
# Statistika Deskriptif



## Statistika Deskriptif

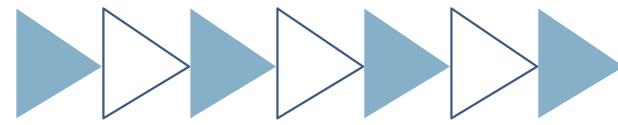
	Age	Sleep Duration	Quality of Sleep	Physical Activity Level	Stress Level	Heart Rate	Daily Steps
Count	374	374	374	374	374	374	374
Mean	42,18	7,13	7,31	59,17	5,39	70,17	6816,84
Std	8,67	0,80	1,20	20,83	1,77	4,14	1617,92
Min	27,0	5,8	4,0	30,0	3,0	65,0	3000
25%	35,25	6,4	6,0	45,0	4,0	68,0	5600
50%	43,0	7,2	7,0	60,0	5,0	70,0	7000
75%	50,0	7,8	8,0	75,0	7,0	72,0	8000
Max	59,0	8,5	9,0	90,0	8,0	86,0	10000

Berdasarkan tabel dapat dilihat bahwa **rata-rata variabel age seseorang yaitu 42 tahun** dengan **usia termuda adalah 27 tahun** dan **tertua adalah 59 tahun**. Variabel **sleep duration** yang paling singkat pada seseorang yaitu **5,8 jam** dan **terlama yaitu 8,5 jam** dengan **rata-rata durasi tidur selama 7,1 jam**. Variabel **quality of sleep** memiliki nilai dengan rating terendah sebesar **4** dan tertinggi sebesar **9** dengan **rata-rata 7,3**. Variabel **physical activity level** memiliki jumlah menit terendah pada seseorang yaitu minimal **30 menit** dan tertinggi **90 menit** dengan **rata-rata tingkat aktivitas fisik seseorang 59 menit**. Variabel **stress level** seseorang dengan rating nilai terendah sebesar **3** dan tertinggi **8** dengan **rata-rata tingkat stress 5,4**. Variabel **heart rate** terendah seseorang yaitu **65 bpm** dan tertinggi **86 bpm** dengan **rata-rata denyut jantung seseorang dalam detak per menit yaitu 70,2 bpm**. Kemudian untuk variabel **daily steps** memiliki jumlah terendah sebesar **3000 langkah** dan tertinggi sebesar **10000 langkah** dengan **rata-rata jumlah langkah yang dilakukan seseorang per hari sebesar 6816 langkah**.

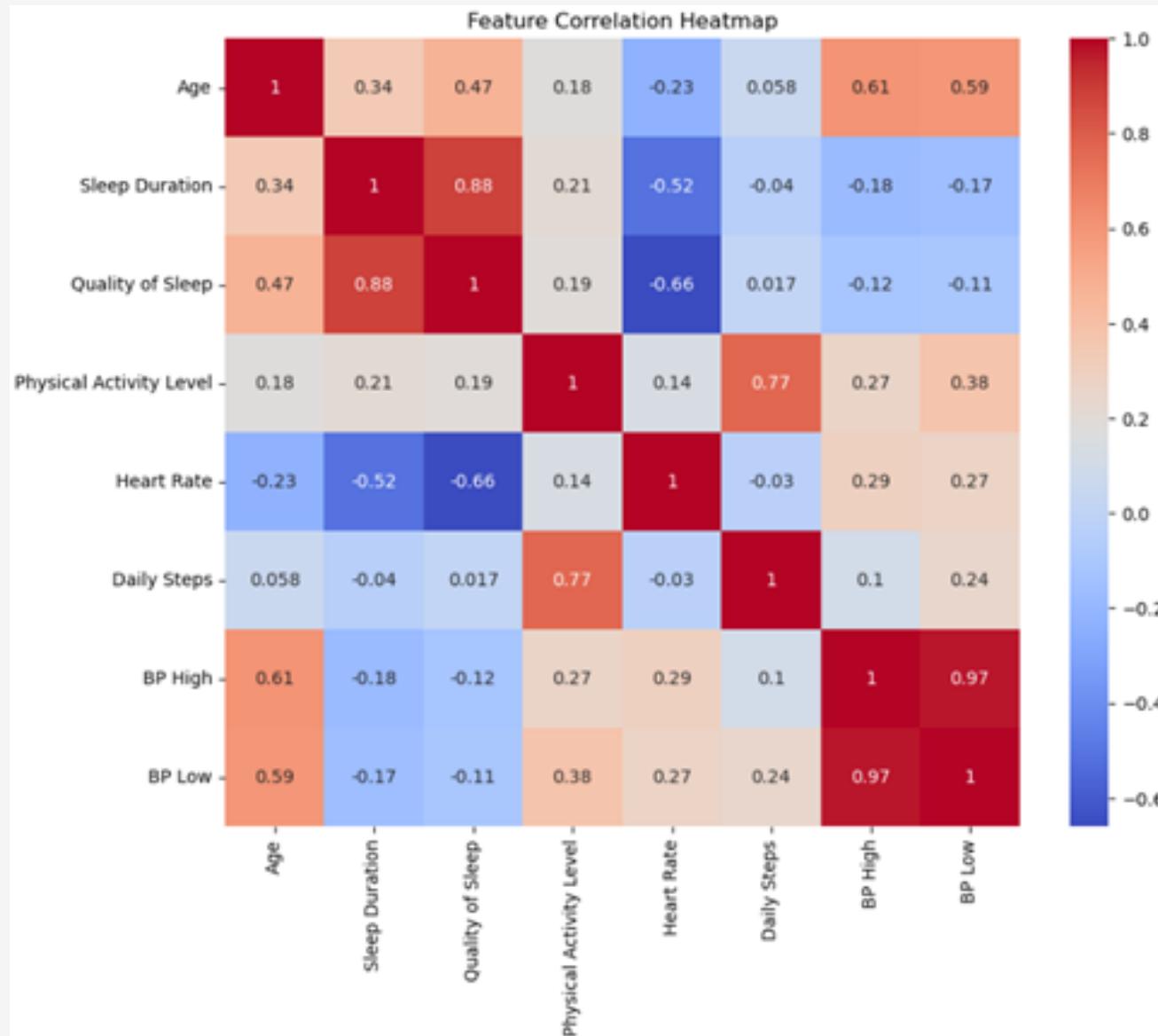


# Visualisasi dan Eksplorasi Data



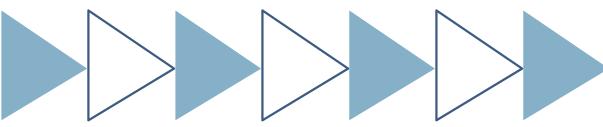


## Correlation Matrix



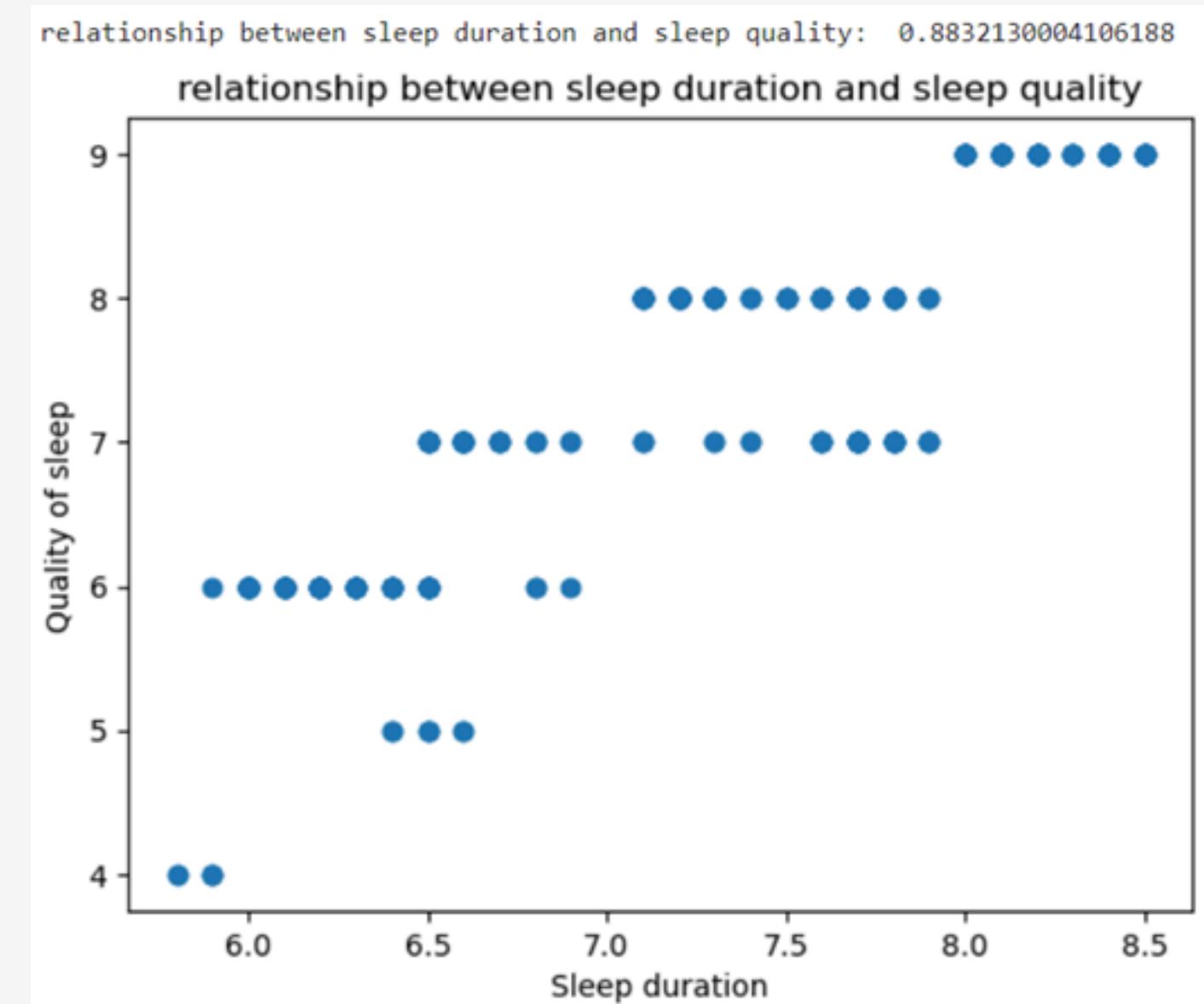
Variabel yang memiliki **hubungan yang cukup kuat** adalah sebagai berikut.

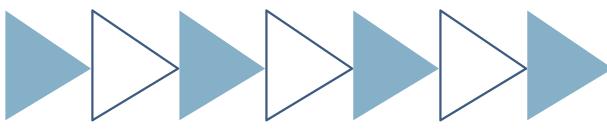
1. Variabel **quality of sleep** dan **sleep duration** dimana memiliki **nilai korelasi sebesar 0,88** yang artinya kedua variabel tersebut memiliki **hubungan positif**. Semakin lama seseorang tidur maka cenderung memiliki kualitas tidur yang lebih baik.
2. Variabel **quality of sleep** dan **heart rate** dimana memiliki **nilai korelasi sebesar -0,52** yang artinya kedua variabel tersebut memiliki **hubungan negatif**. Semakin lama seseorang tidur maka cenderung memiliki detak jantung yang rendah.



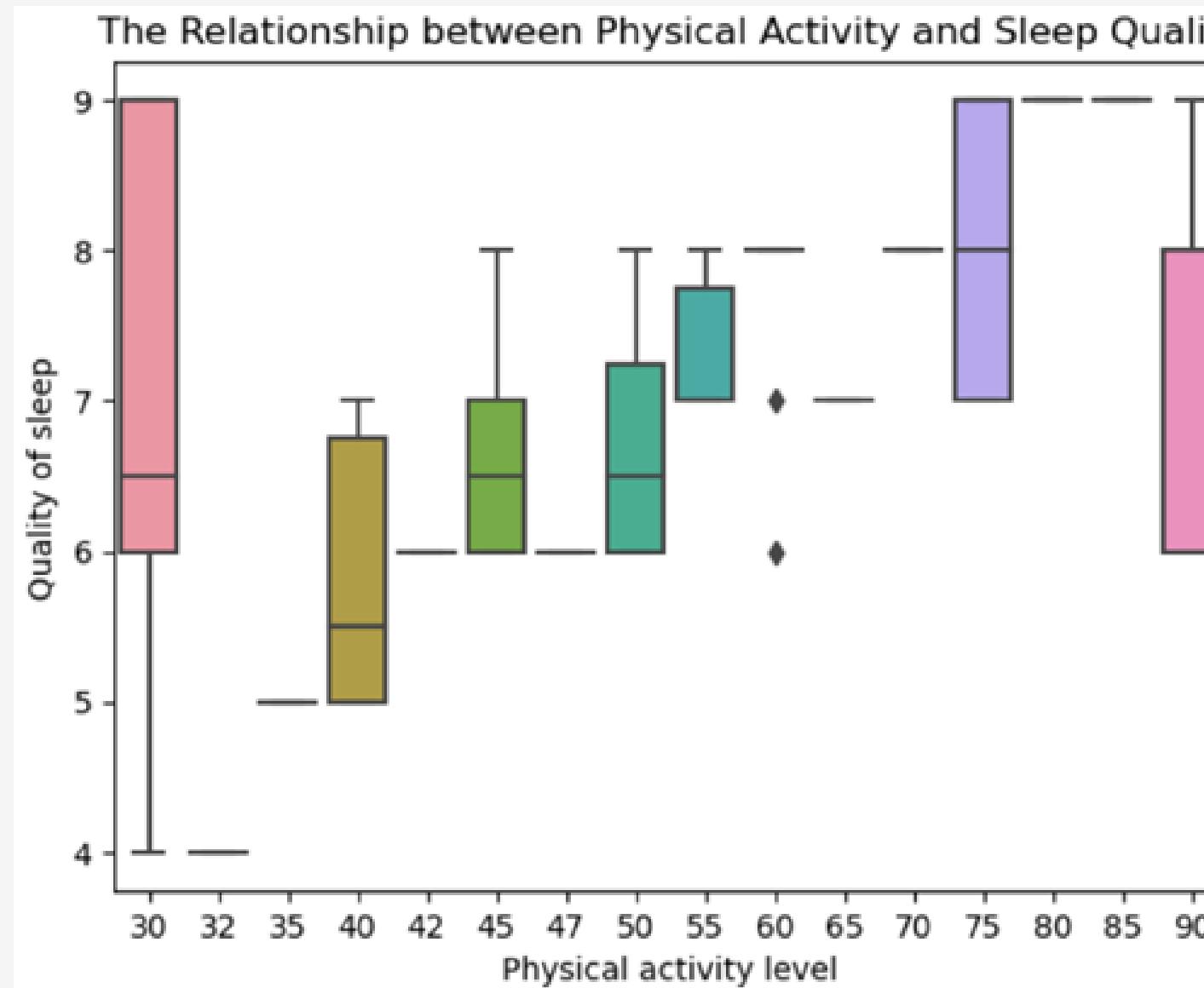
## Korelasi antara Variabel Sleep Duration dan Sleep Quality

Berdasarkan output korelasi pada Gambar disamping terdapat scatter plot yang menunjukkan **hubungan positif yang kuat** antara *sleep duration* dan *sleep quality* yang ditunjukkan oleh **nilai korelasi sebesar 0,88**, yang artinya bahwa orang yang **tidur lebih lama cenderung memiliki kualitas tidur yang lebih baik**. Durasi tidur cukup penting untuk kesehatan dan kesejahteraan secara keseluruhan.

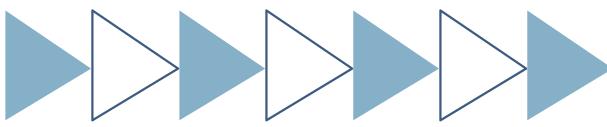




## Korelasi antara Variabel Physical Activity dan Sleep Quality

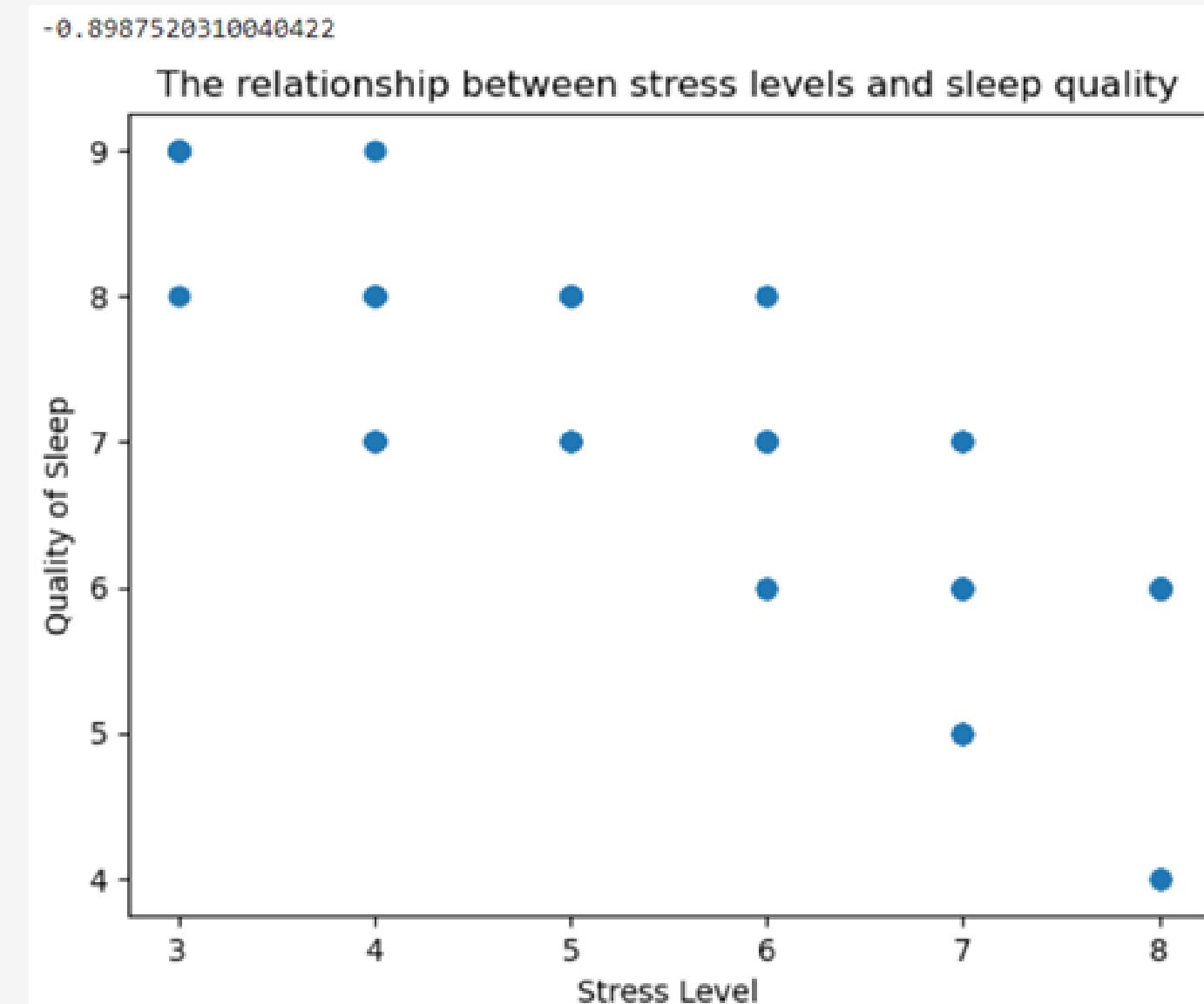


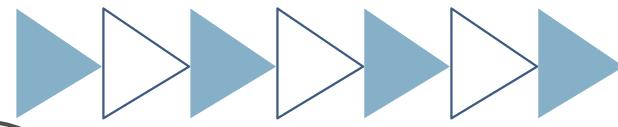
Hasil korelasi menunjukkan bahwa terdapat **hubungan positif yang kuat** antara aktivitas fisik dan kualitas tidur. Hal ini berarti bahwa **orang yang lebih aktif secara fisik cenderung memiliki kualitas tidur yang lebih baik**. Oleh karena itu, penting untuk meningkatkan aktivitas fisik Anda untuk mendapatkan tidur yang lebih nyenyak dan berkualitas.



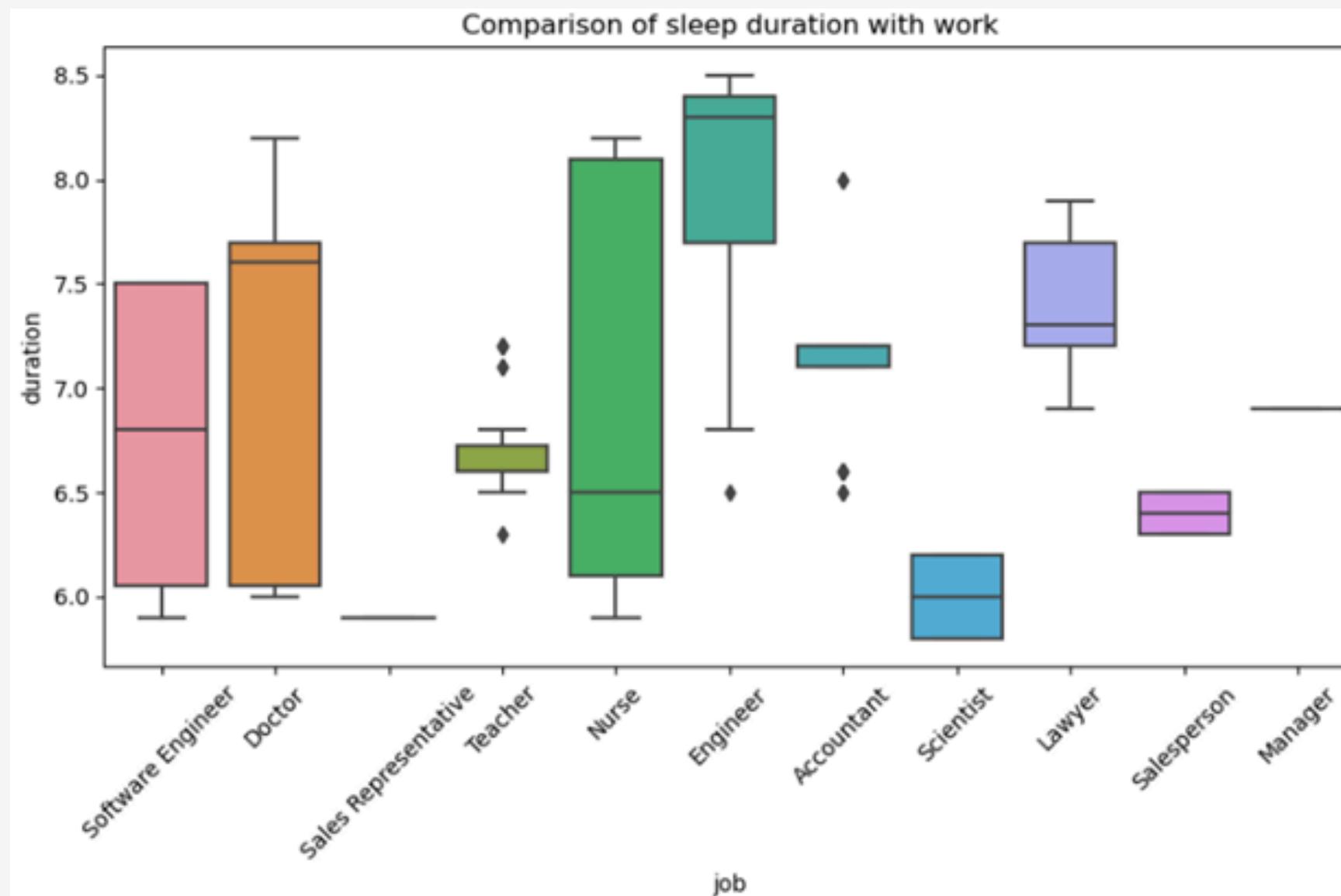
## Korelasi antara Variabel Stress Level dan Sleep Quality

Berdasarkan output korelasi pada Gambar disamping terdapat scatter plot yang menunjukkan **hubungan negatif yang kuat** antara stress level dan sleep quality yang ditunjukkan oleh **nilai korelasi sebesar -0,90**, yang artinya bahwa **seseorang yang mengalami tingkat stres yang lebih tinggi kemungkinan besar akan mengalami kualitas tidur yang lebih rendah.**

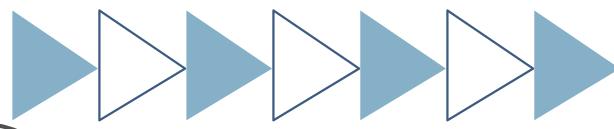




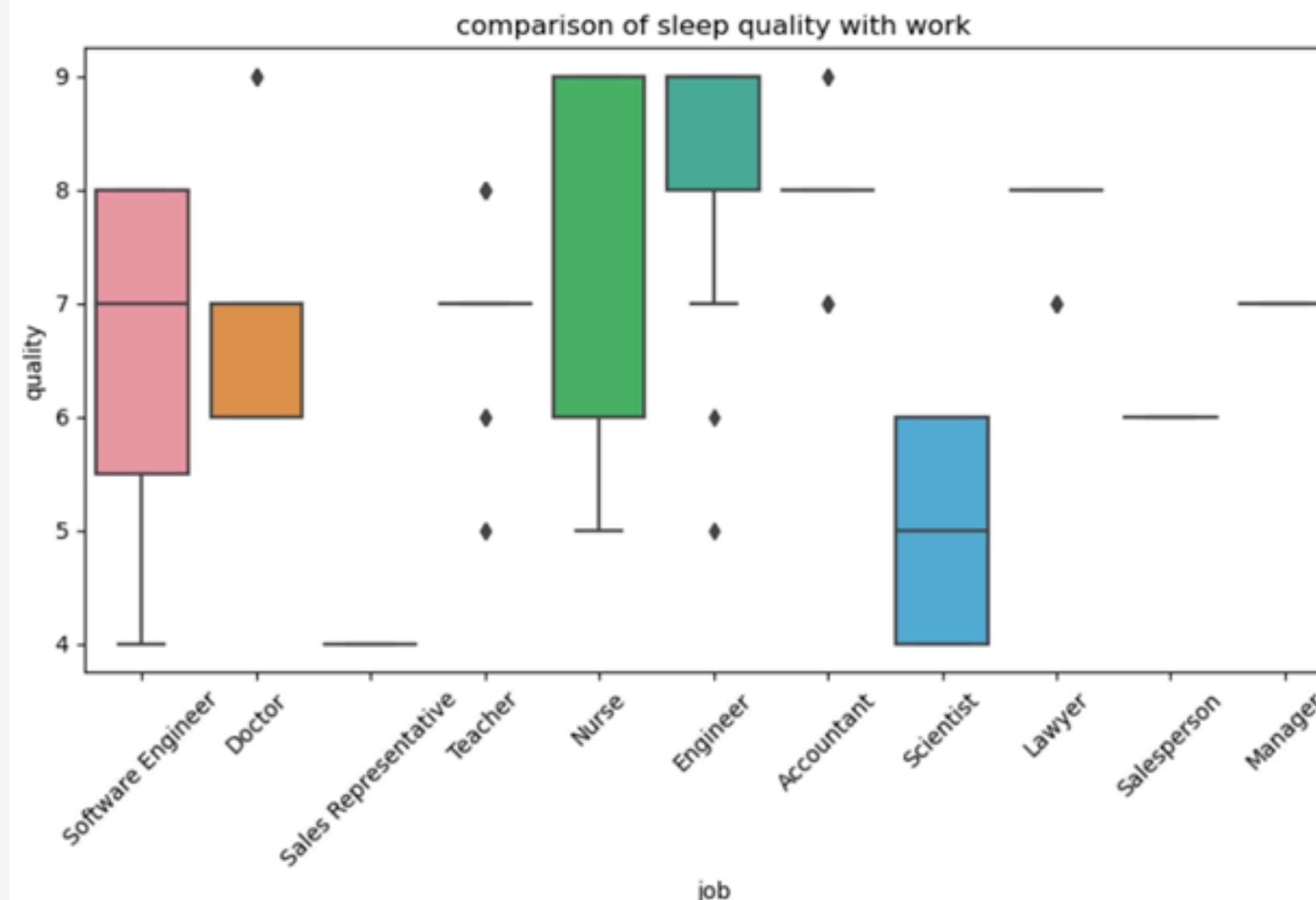
## Perbandingan antara Variabel Occupation dan Sleep Duration



Berdasarkan output perbandingan pada gambar disamping terdapat boxplot yang menunjukkan bahwa terdapat **variasi yang signifikan dalam distribusi durasi tidur berdasarkan jenis pekerjaan**. Secara umum, durasi tidur rata-rata untuk setiap pekerjaan cenderung lebih rendah daripada durasi tidur rata-rata secara keseluruhan. **Variabilitas durasi tidur antar pekerjaan juga cukup besar**, seperti yang tercermin dari interquartile range yang bervariasi di setiap boxplot. Beberapa pekerjaan memiliki nilai outlier, menunjukkan adanya kasus ekstrem dengan durasi tidur yang jauh lebih rendah atau lebih tinggi dibandingkan dengan pekerjaan lainnya. boxplot menunjukkan bahwa beberapa pekerjaan seperti Engineer, Accountant, Scientist, Lawyer, dan Salesperson memiliki durasi tidur rata-rata yang lebih tinggi dibandingkan dengan pekerjaan lainnya seperti Software Engineer, Doctor, Sales Representative, Teacher, dan Nurse.

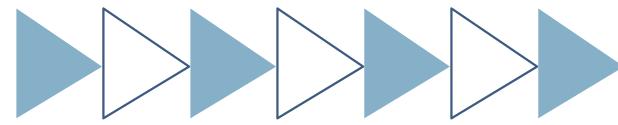


## Perbandingan antara Variabel Occupation dan Sleep Quality



Berdasarkan output perbandingan pada gambar disamping terdapat boxplot yang menunjukkan bahwa **kualitas tidur rata-rata untuk setiap jenis pekerjaan umumnya lebih rendah dibandingkan dengan kualitas tidur rata-rata secara keseluruhan**. Variabilitas kualitas tidur antar pekerjaan juga bervariasi, seperti yang tercermin dari ukuran interquartile range yang berbeda-beda di setiap boxplot. Secara keseluruhan, boxplot juga menggambarkan bahwa beberapa pekerjaan seperti Engineer, Accountant, Scientist, Lawyer, dan Salesperson cenderung memiliki kualitas tidur rata-rata yang lebih tinggi dibandingkan dengan pekerjaan seperti Software Engineer, Doctor, Sales Representative, Teacher, dan Nurse.

# PRE-PROCESSING



## Missing Value

```
Person ID          0  
Gender            0  
Age               0  
Occupation        0  
Sleep Duration    0  
Quality of Sleep   0  
Physical Activity Level 0  
Stress Level      0  
BMI Category      0  
Heart Rate         0  
Daily Steps        0  
Sleep Disorder     219  
BP High            0  
BP Low             0  
dtype: int64
```

**Tidak terdapat missing value** dalam data, kategori orang yang tidak mengalami gangguan tidur yang tercantum pada data adalah "**None**". Sehingga dia terbaca sebagai Missing Value, jadi perlu dilakukan **imputasi** dengan mengganti nilai '**NaN**' menjadi '**No Sleep Disorder**'.

## Data Duplicated

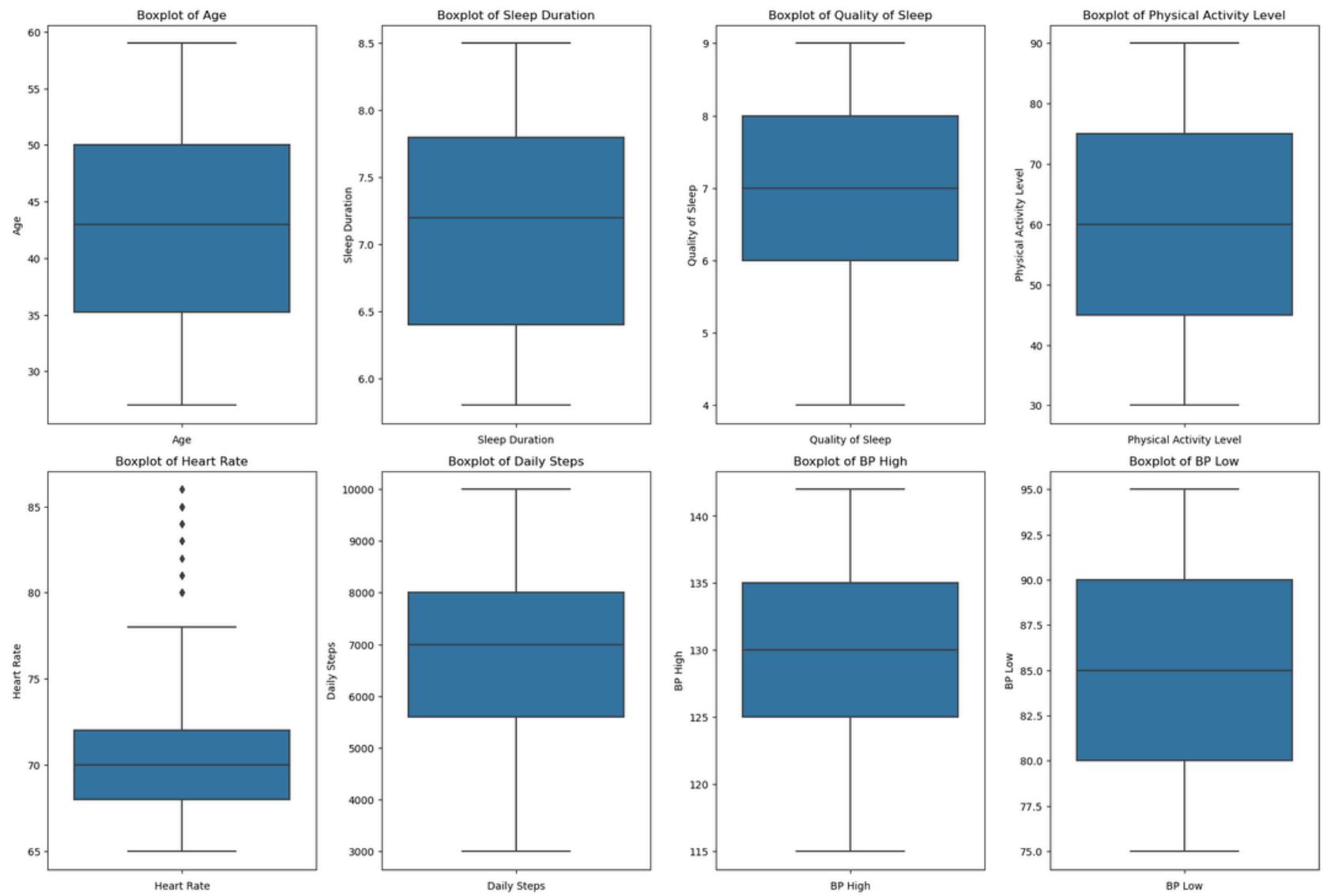
```
# Check for duplicates across all columns  
duplicated = df.duplicated()  
  
# Print the number of duplicated instances  
print("Number of duplicated instances:", duplicated.sum())  
  
# Print the duplicated instances  
df[duplicated]
```

```
Number of duplicated instances: 0
```

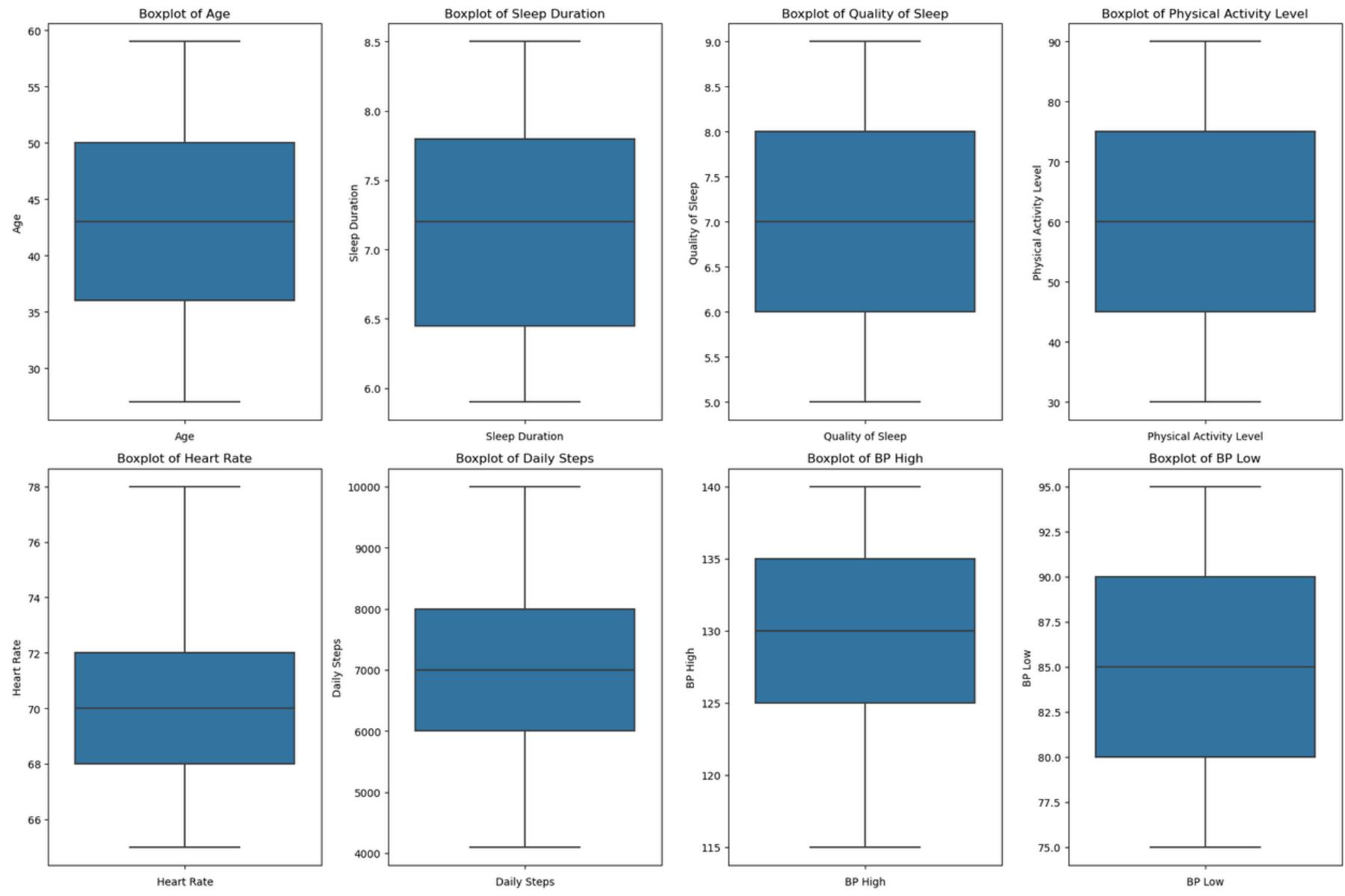
**Pengecekan duplikasi** perlu dilakukan supaya tidak terdapat kesalahan dalam analisis data. Berdasarkan pengecekan pada dataset **tidak terdapat duplikasi data**.

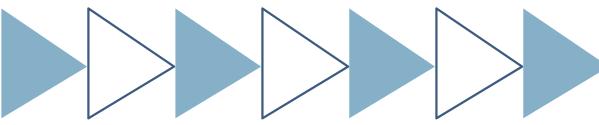
# Outlier

**Outlier** adalah data yang jauh berbeda dari pola umum atau tren dalam kumpulan data. Berdasarkan hasil boxplot dibawah, menunjukkan **adanya outlier** pada variabel **Heart Rate** sehingga perlu diatasi dengan menghilangkan data outlier.



# After Removal





## Imbalance Data

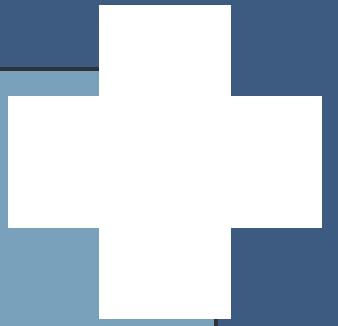
**Imbalance data** adalah kondisi di mana **jumlah sampel atau observasi** dalam setiap kelas atau kategori dalam dataset **tidak seimbang secara signifikan**.

```
Sleep Disorder  
No Sleep Disorder    219  
Insomnia             71  
Sleep Apnea          69  
Name: count, dtype: int64
```

Ketidakseimbangan ini dapat **mempengaruhi performa model** karena algoritma lebih cenderung untuk memprediksi kelas mayoritas dan kurang memperhatikan kelas minoritas. Oleh karena itu untuk mengatasinya, peneliti menggunakan **metode oversampling**.

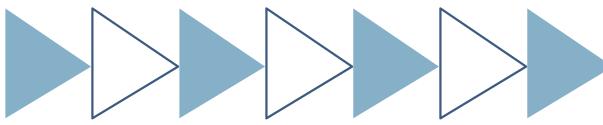
## After Handling

```
Sleep Disorder  
Insomnia              71  
No Sleep Disorder     70  
Sleep Apnea            69  
Name: count, dtype: int64
```



# UJI MULTIKOLINEARITAS

X X X X X X X X X  
X X X X X X X X X  
V V V V V V V V V



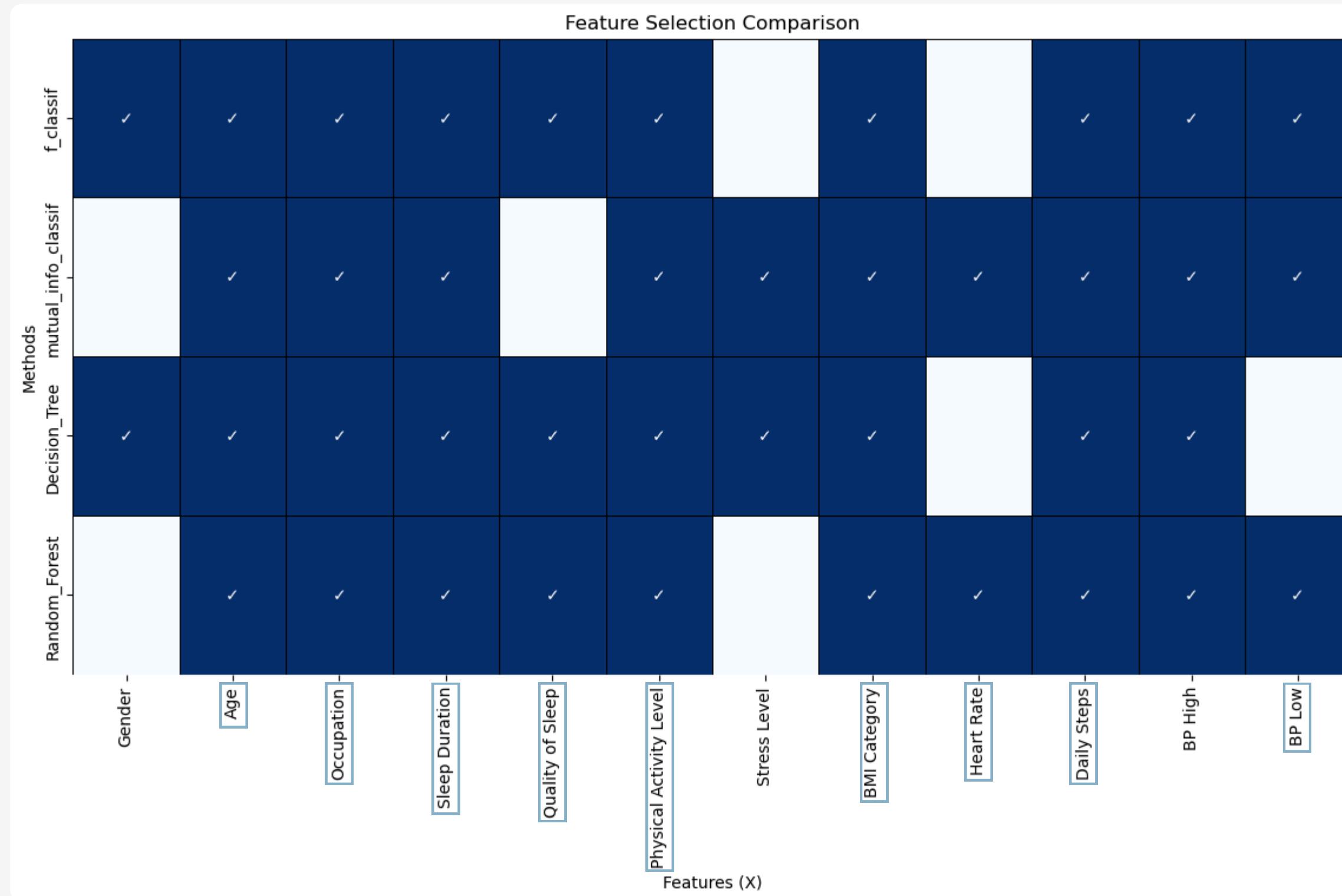
Pemeriksaan multikolinearitas data bertujuan untuk mengidentifikasi **ada atau tidaknya korelasi antar variabel prediktornya**, Salah satu kriteria untuk mendeteksi adanya multikolinearitas adalah **nilai VIF** dari masing-masing variabel prediktor, Berikut ini adalah nilai VIF dari semua variabel prediktor.

Variabel	Nilai VIF
X <sub>1</sub>	68,35
X <sub>4</sub>	11,17
X <sub>5</sub>	8,18
X <sub>7</sub>	6,23
X <sub>8</sub>	8,15
X <sub>9</sub>	68,62

Berdasarkan nilai VIF dari masing-masing variabel di atas, dapat dilihat terdapat 3 **variabel prediktor** memiliki **nilai VIF lebih dari 10**, sehingga dapat disimpulkan bahwa **terdapat multikolinearitas pada data**. Untuk mengatasi hal tersebut, akan dilakukan **feature selection** pada data menggunakan beberapa metode. Setelah dilakukan feature selection didapati bahwa **tidak terdapat multikolinearitas pada data**.

	feature	VIF
0	const	3127.352317
1	Age	5.801546
2	Occupation	3.604587
3	Quality of Sleep	7.718719
4	Physical Activity Level	7.423364
5	BMI Category	6.299802
6	Heart Rate	3.892019
7	Daily Steps	5.743925
8	BP Low	5.268164

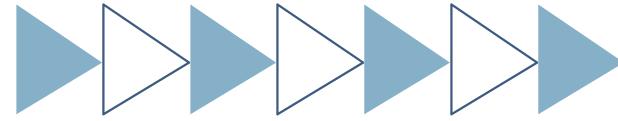
# ***FEATURE SELECTION***



**Terpilih 9 variabel** sebagai prediktor dengan minimal **empat metode** yang relevan dengan **sembilan** variabel tersebut.



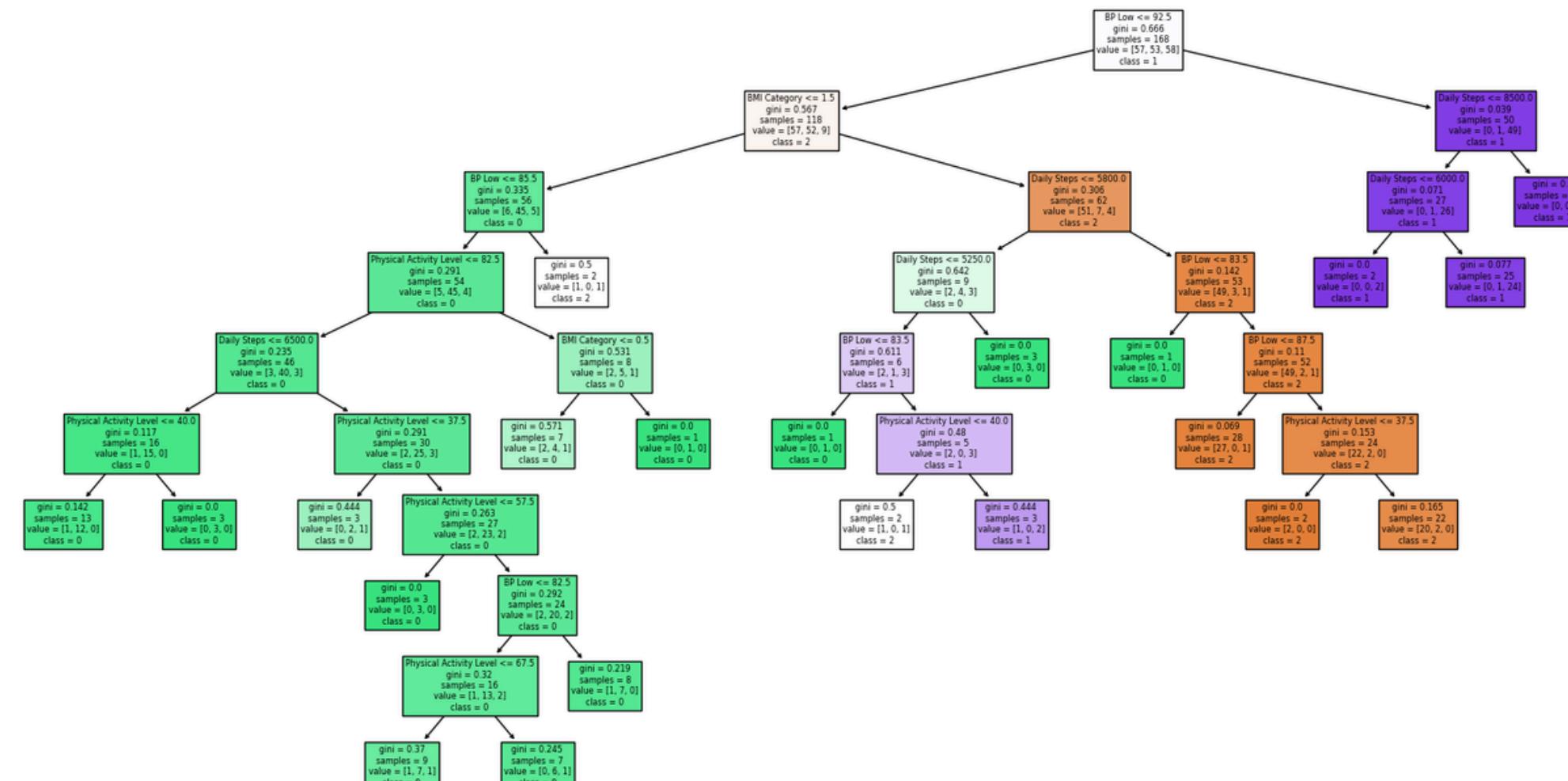
# *DECISION TREE*

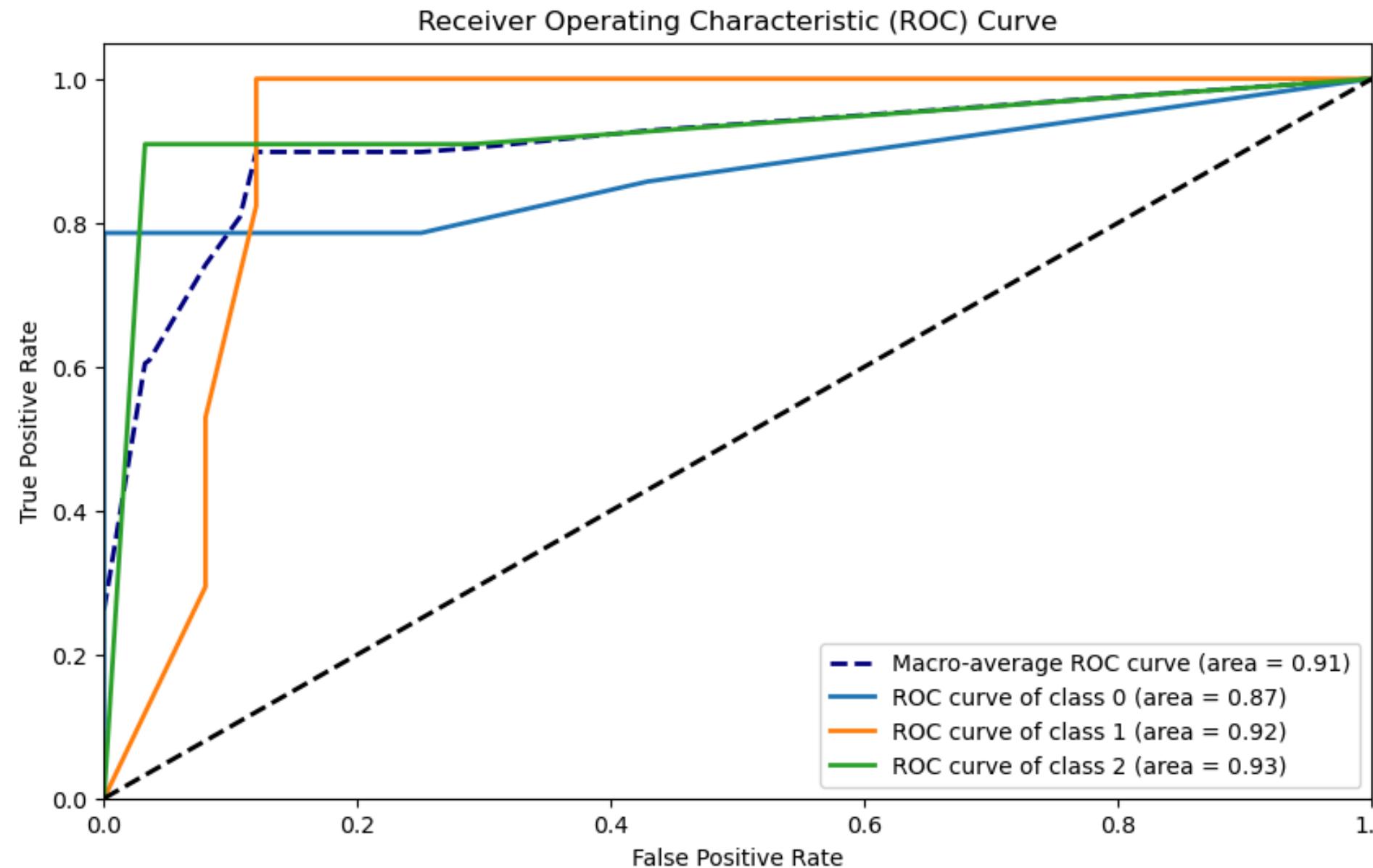
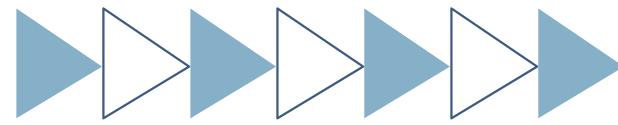


Pada proses klasifikasi **metode Decision Tree** akan dilakukan beberapa tahapan yaitu inisialisasi model, menampilkan fitur penting dalam model, melakukan hyperparameter tuning dengan Grid Search, hingga menampilkan kurva ROC pada masing-masing kategori data variabel respon. Dimana pada **tunning parameter** didapatkan parameter terbaik model sebagai berikut.

Parameter	Hasil
Criterion	Gini
Maximum depth	40
Minimum samples leaf	3
Minimum samples split	4

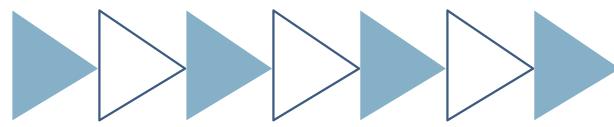
Melalui tahapan tersebut, didapatkan **hasil decision tree** pada Gambar berikut.





Nilai **Macro-Averaged AUC** untuk metode Decision Tree adalah **0,91**, yang menandakan bahwa model ini **sangat baik dalam membedakan antara berbagai kelas secara keseluruhan**. AUC yang mendekati 1 menunjukkan kemampuan diskriminatif model yang sangat tinggi, memberikan keyakinan lebih pada penggunaan model ini untuk klasifikasi gangguan tidur pada seseorang dalam dataset yang dianalisis.

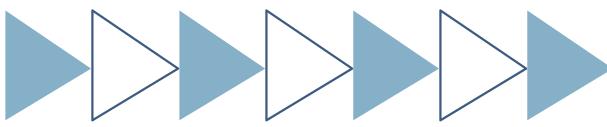
# *RANDOM FOREST*



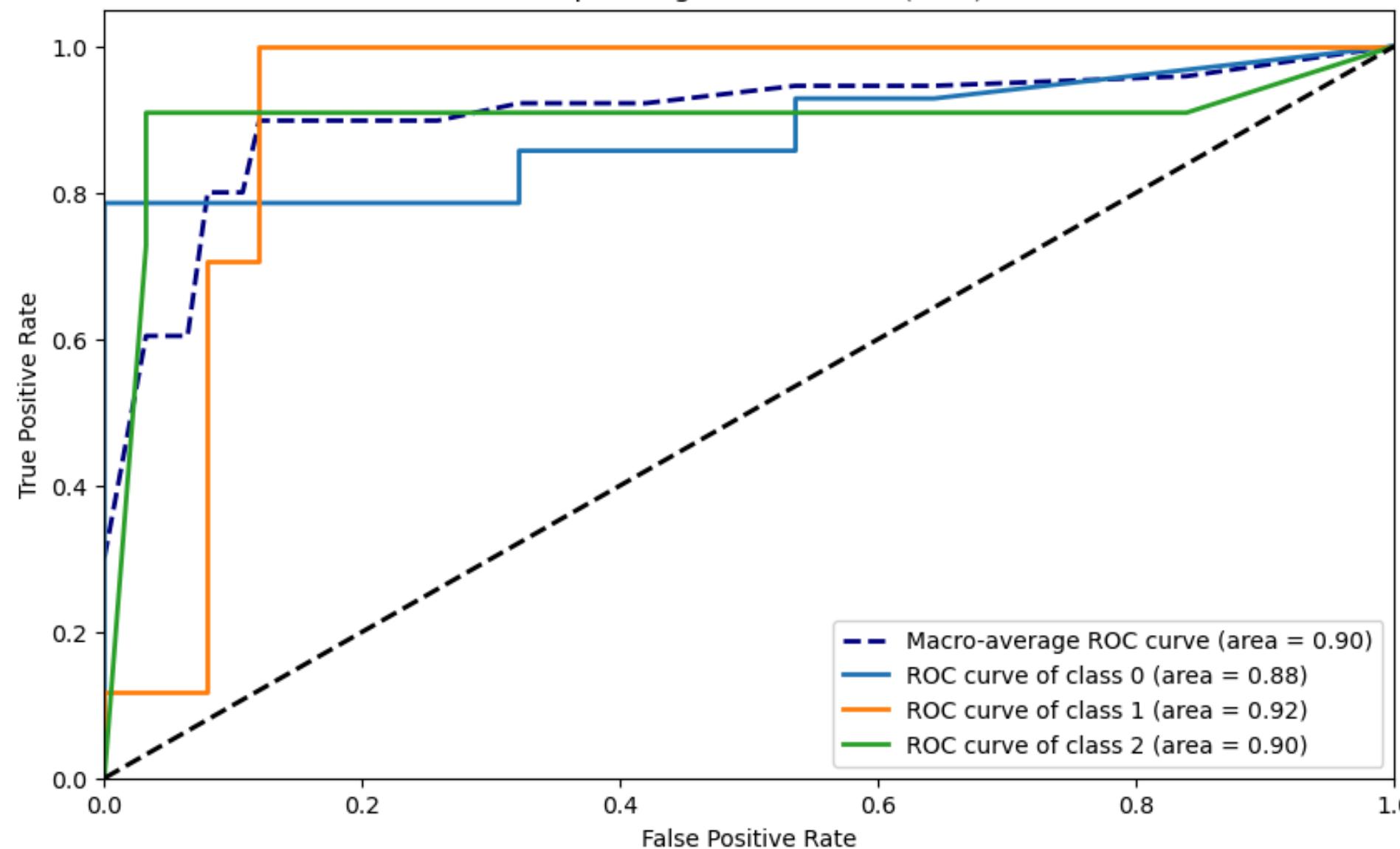
Pada proses klasifikasi **metode Random Forest** dilakukan uji coba kombinasi beberapa hyperparameter untuk memperoleh model dengan hyperparameter paling optimal. **Kombinasi hyperparameter** yang diuji coba yaitu jumlah pohon (**n\_estimators**) sebanyak **50, 100, dan 200**. Jumlah sampel minimum yang dibutuhkan untuk memecah node internal (**min\_samples\_split**) terdiri dari **2, 5, dan 10**. Serta, jumlah sampel minimum yang dibutuhkan untuk menjadi daun pada pohon (**min\_samples\_leaf**) terdiri dari **1, 2, dan 4**. Pemodelan random forest dengan menggunakan hasil feature importance pada random forest didapatkan parameter terbaik model sebagai pada Tabel berikut.

Parameter	Hasil
Maximum depth	40
Maximum features	Log2
Minimum samples leaf	1
Minimum samples split	5
n estimators	192

Tahapan selanjutnya adalah **meng evaluasi kebaikan model random forest** dengan 4 fitur sederhana berdasarkan **skor MDI dan parameter yang telah ditentukan** berdasarkan **hasil GridSearch menggunakan classification report** sebagaimana Tabel berikut.



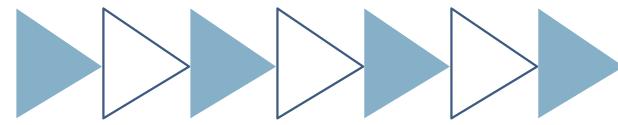
Receiver Operating Characteristic (ROC) Curve



Parameter	Precision	Recall	F1-Score	Support
0	0,98	0,79	0,88	14
1	0,85	0,98	0,92	17
2	0,91	0,91	0,91	11
Accuracy			0,90	42
Macro avg	0,92	0,90	0,90	42
Weighted avg	0,92	0,90	0,90	42

Nilai **Macro-Averaged AUC** untuk **metode Random Forest** adalah **0,90**, yang menandakan bahwa model ini **sangat baik dalam membedakan antara berbagai kelas secara keseluruhan**. AUC yang mendekati 1 menunjukkan **kemampuan diskriminatif model yang sangat tinggi**, memberikan keyakinan lebih pada penggunaan model ini untuk klasifikasi gangguan tidur pada seseorang dalam dataset yang dianalisis.

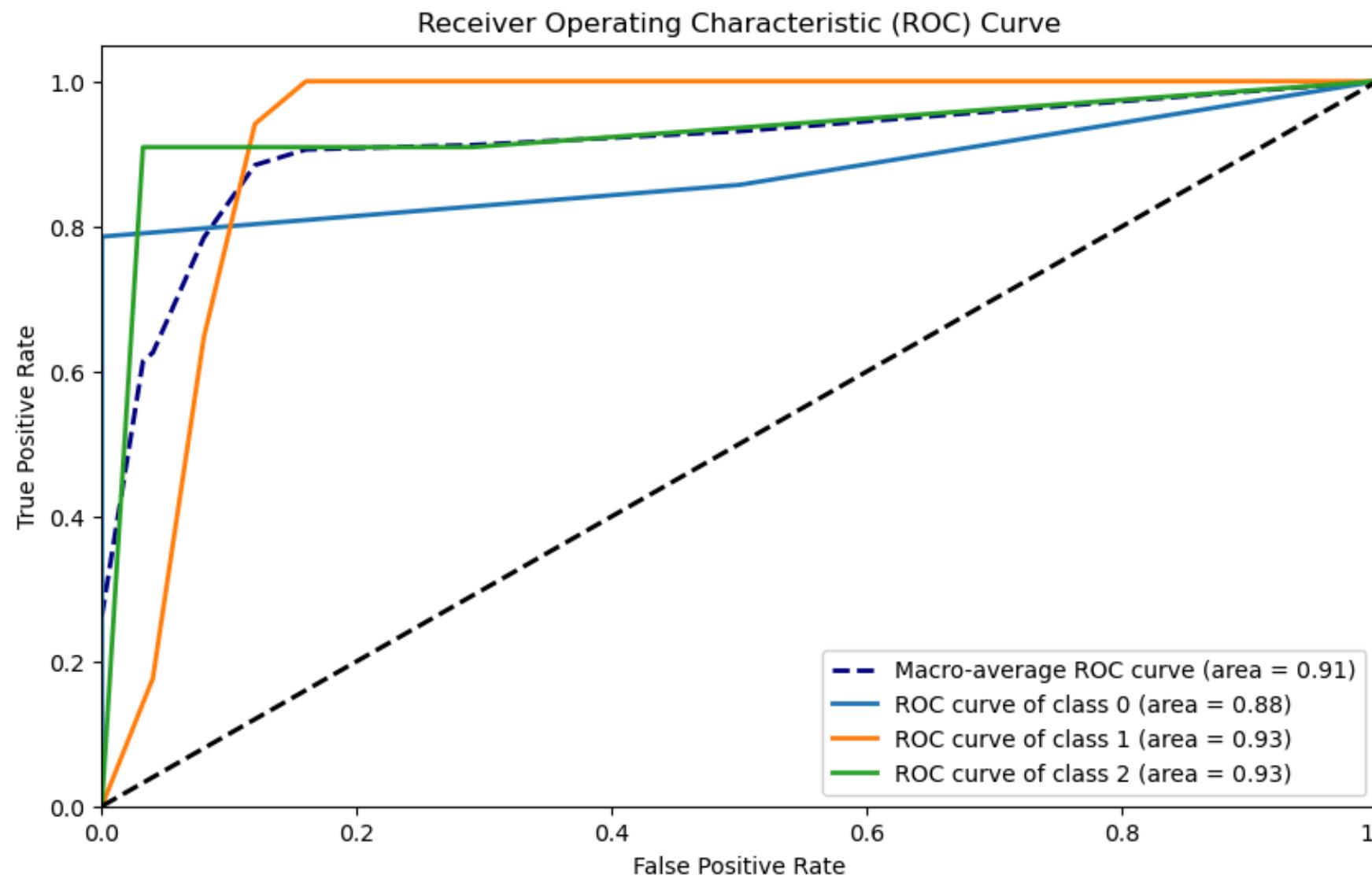
# **K-NEAREST NEIGHBORS (KNN)**



Pemodelan dengan **metode K-Nearest Neighbors** dilakukan dengan inisialisasi dan pelatihan model, prediksi dan evaluasi, confusion matrix dan classification report, serta hyperparameter tuning dengan Grid Search. Dengan tahapan tersebut, telah didapatkan **parameter terbaik dengan k sebesar 5**. Dengan nilai k tersebut, didapatkan nilai akurasi pada data train dan test sebagai berikut.

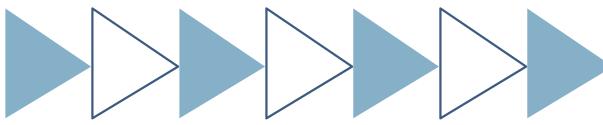
Data	Akurasi
Train	89,29%
Test	90,48%

Melalui tahapan tersebut, didapatkan **kurva ROC dan AUC** sebagai berikut



Nilai **Macro-Averaged AUC** untuk metode KNN adalah **0,91**, yang menandakan bahwa model ini **sangat baik dalam membedakan antara berbagai kelas secara keseluruhan**. AUC yang mendekati 1 menunjukkan kemampuan diskriminatif model yang sangat tinggi, memberikan keyakinan lebih pada penggunaan model ini untuk klasifikasi gangguan tidur pada seseorang dalam dataset yang dianalisis.

# **SUPPORT VECTOR MACHINE (SVM)**

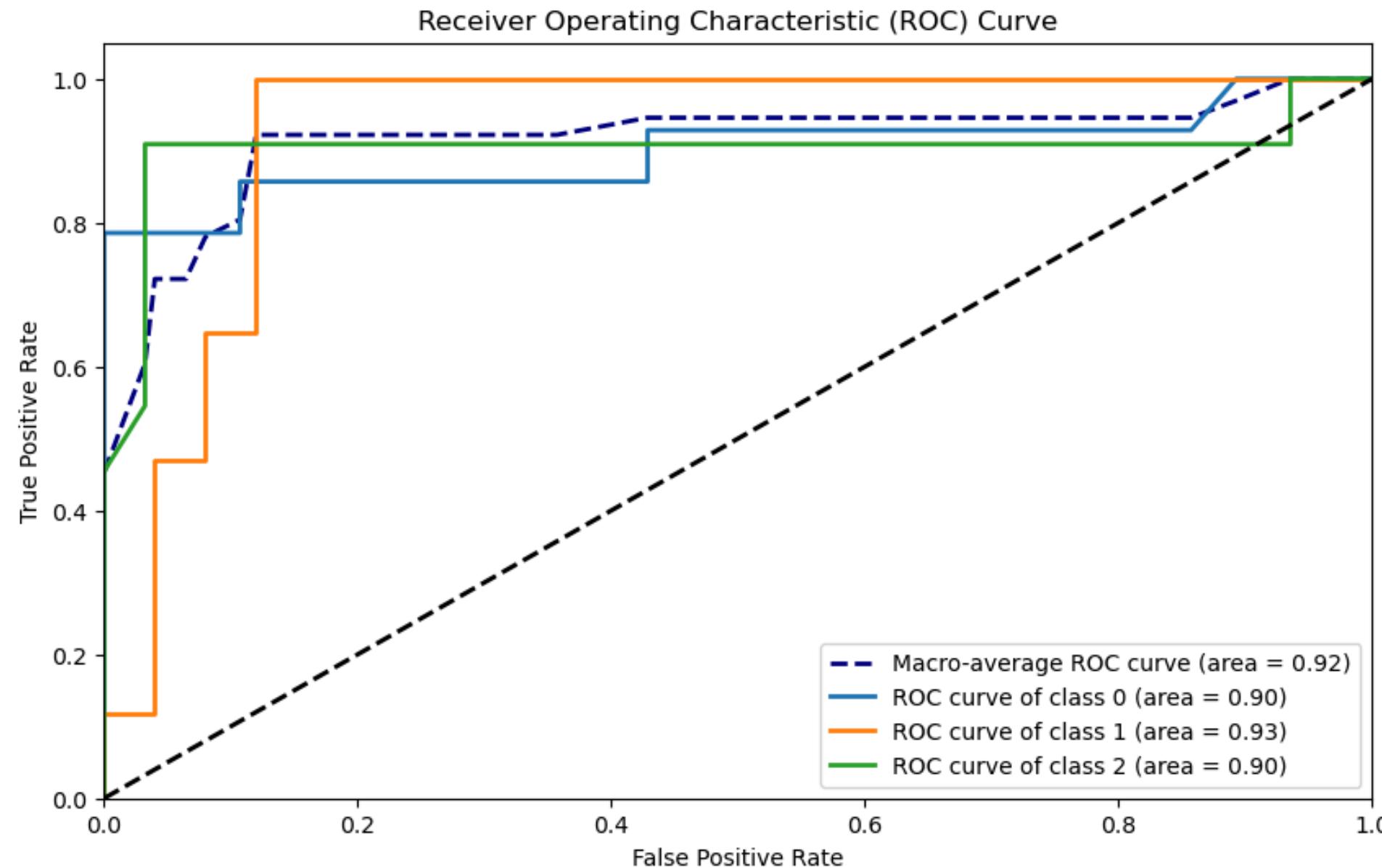
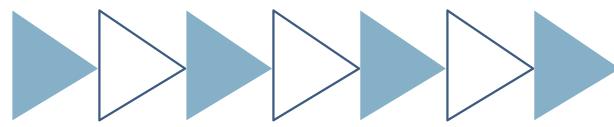


Pemodelan dengan **metode Support Vector Machine (SVM)** dilakukan dengan memeriksa pada dua jenis kernel yaitu **kernel linear dan RBF**. Kemudian dilakukan tuning parameter menggunakan GridSearchCV untuk mencari kombinasi hyperparameter terbaik (C, gamma, kernel) untuk model SVM dengan kernel 'rbf'. Setelah dilakukan uji coba, **paramater terbaik** yang didapatkan adalah sebagai berikut.

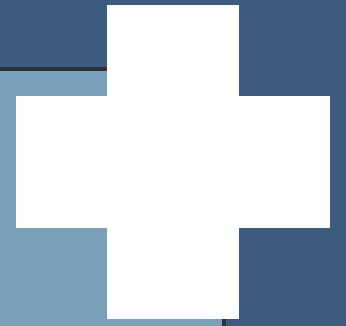
Parameter	Hasil
C	1
Gamma	auto
Kernel	rbf

Selain itu, juga dilakukan pemodelan menggunakan kernel linear. Dari kedua kernel tersebut, berikut merupakan **hasil perbandingan akurasi** antar keduanya.

Kernel	Akurasi
Linear	0,7619
rbf	0,9048



Nilai **Macro-Averaged AUC untuk metode SVM** adalah **0,92**, yang menandakan bahwa model ini **sangat baik dalam membedakan antara berbagai kelas secara keseluruhan**. AUC yang mendekati 1 menunjukkan kemampuan diskriminatif model yang sangat tinggi, memberikan keyakinan lebih pada penggunaan model ini untuk klasifikasi gangguan tidur pada seseorang dalam dataset yang dianalisis.

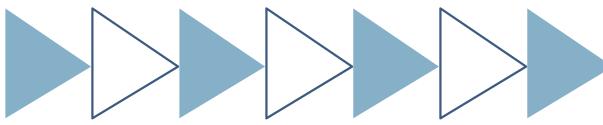


# HASIL PERBANDINGAN

X X X X X X X X X

X X X X X X X X X

< < < < < < < <



Setelah dilakukan pengujian satu per satu dari tiap algoritma kemudian hasil yang didapatkan **dibandingkan untuk menentukan algoritma yang cocok** untuk **mengklasifikasikan gangguan tidur seseorang**. Berdasarkan hasil analisis yang telah dilakukan, hasil perbandingan dari keempat metode algoritma yaitu sebagaimana pada Tabel berikut.

Metode	Akurasi	AUC
<i>Decision Tree</i>	0,9048	0,91
<i>Random Forest</i>	0,9048	0,90
<i>K-Nearest Neighbors</i>	0,8929	0,91
<i>Support Vector Machine</i>	0,9048	0,92

Terlihat bahwa **seluruh metode memiliki nilai akurasi dan AUC yang mirip** dengan **kategori sangat baik**. Namun, klasifikasi **metode SVM lebih tinggi** dibandingkan dengan hasil klasifikasi lainnya. Berdasarkan Tabel diatas, metode SVM dengan parameter **C sebesar 1, gamma adalah auto**, dan dengan **kernel rbf** mampu mengklasifikasikan gangguan tidur seseorang dengan skor **AUC sebesar 0,92** yang masuk dalam kategori **excellent classification**.

# KESIMPULAN DAN SARAN

# Kesimpulan

- 1 Berdasarkan hasil analisis, **metode Decision Tree, Random Forest, K-Nearest Neighbors (KNN), dan Support Vector Machine (SVM)** mampu mengklasifikasikan gangguan tidur dengan **kategori "excellent classification"**. Decision Tree dan Random Forest mencapai akurasi 0,905 dengan AUC 0,90 dan 0,91 secara berturut-turut, sementara KNN mencapai akurasi 0,893 dengan AUC 0,91, dan **SVM mencapai akurasi 0,905 dengan AUC tertinggi 0,92**.
- 2 **Performa tertinggi** dalam mengklasifikasikan gangguan tidur dicapai oleh **metode Support Vector Machine (SVM)**, yang setelah tahap latih dan tuning parameter, memberikan nilai **AUC 0,92** yang lebih baik dibandingkan dengan metode Decision Tree, Random Forest, dan K-Nearest Neighbors (KNN).

## Saran

Disarankan agar peneliti mempertimbangkan penggunaan **algoritma machine learning tambahan** seperti AdaBoost atau LightGBM untuk hasil yang lebih baik atau lebih efisien. Penggunaan **teknik penyeimbangan data lain** seperti Random Under Sampling atau ADASYN dapat meningkatkan kinerja model pada dataset tidak seimbang. **Perluasan pencarian hyperparameter dan penggunaan teknik optimasi** seperti Grid Search atau Random Search direkomendasikan untuk parameter yang lebih optimal. **Validasi menggunakan dataset yang berbeda** atau lebih besar penting untuk memastikan generalisasi model. **Teknik visualisasi hasil** seperti t-SNE atau PCA dapat memberikan wawasan lebih jelas tentang distribusi data dan kinerja model, berguna untuk analisis lanjutan dan aplikasi praktis.



# LAMPIRAN

× × × × × × × × ×

× × × × × × × × ×

✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓

## Import Data

```
[1]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import scipy.stats as stats
from sklearn.preprocessing import LabelEncoder
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

[2]: df = pd.read_csv('Sleep_health_dataset.csv')
df

[3]: df.shape

[3]: (374, 13)
```

## Eksplorasi dan Visualisasi Data

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

[5]: df[['BP High', 'BP Low']] = df['Blood Pressure'].str.split('/', expand=True)
df.drop('Blood Pressure', axis=1, inplace=True)
df.head(3)

[6]: categoric = ['Gender', 'Occupation', 'BMI Category', 'Sleep Disorder']
numeric = df[['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level', 'Heart Rate', 'Daily Steps', 'BP High', 'BP Low']]

[7]: print(type(numeric))
<class 'pandas.core.frame.DataFrame'>

[8]: # Create the correlation matrix
corr_matrix = numeric.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Feature Correlation Heatmap')
plt.show()
```

```
# Relationship between sleep duration and sleep quality
korelasi = df['Sleep Duration'].corr(df['Quality of Sleep'])
print('relationship between sleep duration and sleep quality: ', korelasi)

plt.scatter(df['Sleep Duration'], df['Quality of Sleep'])
plt.xlabel('Sleep duration')
plt.ylabel('Quality of sleep')
plt.title('relationship between sleep duration and sleep quality')
plt.show()

# Relationship between Physical Activity and Sleep Quality
sns.boxplot(x='Physical Activity Level', y='Quality of Sleep', data=df)
plt.xlabel('Physical activity level')
plt.ylabel('Quality of sleep')
plt.title('The Relationship between Physical Activity and Sleep Quality')
plt.show()

# Relationship between stress levels and sleep quality
stres = df['Stress Level']
kualitas_tdr = df['Quality of Sleep']

kor = stres.corr(kualitas_tdr)
print(kor)

plt.scatter(stres, kualitas_tdr)
plt.xlabel('Stress Level')
plt.ylabel('Quality of Sleep')
plt.title('The relationship between stress levels and sleep quality')
plt.show()

# Occupation and Sleep Duration
plt.figure(figsize=(10, 5))
sns.boxplot(x='Occupation', y='Sleep Duration', data=df)
plt.title('Comparison of sleep duration with work')
plt.xlabel('job')
plt.ylabel('duration')
plt.xticks(rotation=45)
plt.show()
```

```

# Occupation and Sleep Quality
plt.figure(figsize=(10, 5))
sns.boxplot(x='Occupation', y='Quality of Sleep', data=df)
plt.title('comparison of sleep quality with work')
plt.xlabel('job')
plt.ylabel('quality')
plt.xticks(rotation=45)
plt.show()

```

## PRE-PROCESSING

```

df.info()

df.describe()

# Check for duplicates across all columns
duplicated = df.duplicated()

# Print the number of duplicated instances
print("Number of duplicated instances:", duplicated.sum())

# Print the duplicated instances
df[duplicated]

# Checking for missing values
df.isnull().sum()

df['Sleep Disorder'] = df['Sleep Disorder'].fillna('No Sleep Disorder')
df.isna().sum()
df

df['Sleep Disorder'].unique()

array(['No Sleep Disorder', 'Sleep Apnea', 'Insomnia'], dtype=object)

# Checking for missing values
df.isnull().sum()

```

```

# Mengubah kolom BP High dan BP Low dari object menjadi integer
df['BP High'] = df['BP High'].astype(int)
df['BP Low'] = df['BP Low'].astype(int)

# Checking Outlier
numerical_variables = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level', 'Heart Rate', 'Daily Steps', 'BP High', 'BP Low']

fig, axs = plt.subplots(nrows=len(numerical_variables)//4 + 1, ncols=4, figsize=(18, 6*(len(numerical_variables)//4 + 1)))

for i, var in enumerate(numerical_variables):
    sns.boxplot(y=df[var], ax=axs[i//4, i%4])
    axs[i//4, i%4].set_xlabel(var)
    axs[i//4, i%4].set_title(f'Boxplot of {var}')

# Menghapus subplot yang tidak terpakai
for j in range(i + 1, len(axs.flatten())):
    fig.delaxes(axs.flatten()[j])

# Handling Outlier
variables = ['Heart Rate']

df_clean = df.copy()

for var in variables:
    if var != 'concave points_worst':
        Q1 = df[var].quantile(0.25)
        Q3 = df[var].quantile(0.75)
        IQR = Q3 - Q1
        df_clean = df_clean[((df_clean[var] < (Q1 - 1.5 * IQR)) | (df_clean[var] > (Q3 + 1.5 * IQR)))]

#Checking Outlier after Removal
variables = ['Age', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level',
            'Heart Rate', 'Daily Steps', 'BP High', 'BP Low']

num_cols = 4
num_rows = (len(variables) + num_cols - 1) // num_cols

fig, axs = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(18, 6 * num_rows))
axs = axs.flatten()

# Plot the boxplots
for i, var in enumerate(variables):
    sns.boxplot(y=df_clean[var], ax=axs[i])
    axs[i].set_xlabel(var)
    axs[i].set_title(f'Boxplot of {var}')

# Remove any empty subplots
for j in range(i+1, len(axs)):
    fig.delaxes(axs[j])

plt.tight_layout()
plt.show()

#Mengganti type data variabel "Sleep Duration" menjadi object
df_clean['Sleep Duration'] = df_clean['Sleep Duration'].astype('object')
df_clean.info()

```

```

# Handling Imbalance Data
from sklearn.utils import resample

No_sleep = df_clean[df_clean['Sleep Disorder'] == 'No Sleep Disorder']
insom_apnea = df_clean[(df_clean['Sleep Disorder'] == 'Insomnia') | (df_clean['Sleep Disorder'] == 'Sleep Apnea')]
minority_samp = resample(No_sleep, replace=True, n_samples=int(0.5*len(insom_apnea)))

df_balanced = pd.concat([insom_apnea, minority_samp])
print(df_balanced['Sleep Disorder'].value_counts())

```

---

## Uji Multikolinearitas

```

# Menentukan variabel independen
X = df_balanced.drop(['Sleep Disorder'],axis=1)

# Select only numeric columns
X_numeric = X.select_dtypes(include=['float', 'int'])

# Menambahkan konstanta (intersep) ke dalam model
X_numeric = sm.add_constant(X_numeric)

# Menghitung VIF untuk setiap variabel
vif_data = pd.DataFrame()
vif_data["feature"] = X_numeric.columns
vif_data["VIF"] = [variance_inflation_factor(X_numeric.values, i) for i in range(X_numeric.shape[1])]

print(vif_data)

```

---

## FEATURE SELECTION

```

from sklearn.datasets import load_breast_cancer, fetch_california_housing
from sklearn.feature_selection import SelectKBest, f_classif, mutual_info_classif, f_regression, mutual_info_regression, RFE
from sklearn.tree import DecisionTreeClassifier,DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.linear_model import LogisticRegression, LinearRegression

df_balanced.columns

```

---

```

Index(['Person ID', 'Gender', 'Age', 'Occupation', 'Sleep Duration',
       'Quality of Sleep', 'Physical Activity Level', 'Stress Level',
       'BMI Category', 'Heart Rate', 'Daily Steps', 'Sleep Disorder',
       'BP High', 'BP Low'],
      dtype='object')

```

```

df_balanced.drop(['Person ID'],axis=1,inplace=True)
df_balanced

```

---

```

#Ubah ke dalam bentuk numerik
kategori = ['Gender','Occupation','BMI Category','Sleep Disorder']
encoded_df = LabelEncoder()

for feature in kategori:
    if feature in df.columns.values:
        df_balanced[feature] = encoded_df.fit_transform(df_balanced[feature])

df_balanced.head()

```

---

# 1. SelectKBest with f\_classif

```

selector_f = SelectKBest(score_func=f_classif, k=10)
X_f = selector_f.fit_transform(X, y)
selected_features_f = X.columns[selector_f.get_support(indices=True)].tolist()

df_f = pd.DataFrame(X_f, columns=selected_features_f)
df_f['Sleep Disorder'] = y.reset_index(drop=True)
df_f

```

---

# 2. SelectKBest with mutual\_info\_classif

```

selector_mutual_info = SelectKBest(score_func=mutual_info_classif, k=10)
X_mutual_info = selector_mutual_info.fit_transform(X, y)
selected_features_mutual_info = X.columns[selector_mutual_info.get_support(indices=True)].tolist()

df_mutual_info = pd.DataFrame(X_mutual_info, columns=selected_features_mutual_info)
df_mutual_info['Sleep Disorder'] = y.reset_index(drop=True)
df_mutual_info

```

---

# 3. Feature importance from Decision Tree

```

tree = DecisionTreeClassifier()
tree.fit(X, y)
importances_tree = tree.feature_importances_
indices_tree = importances_tree.argsort()[-10:][::-1]
selected_features_tree = X.columns[indices_tree].tolist()

X_tree = X.iloc[:, indices_tree]
df_tree = pd.DataFrame(X_tree, columns=selected_features_tree)
df_tree['Sleep Disorder'] = y
df_tree

```

```

# 4. Feature importance from Random Forest
forest = RandomForestClassifier()
forest.fit(X, y)
importances_rf = forest.feature_importances_
indices_rf = importances_rf.argsort()[-10:][::-1]
selected_features_rf = X.columns[indices_rf].tolist()

X_rf = X.iloc[:, indices_rf]
df_rf = pd.DataFrame(X_rf, columns=selected_features_rf)
df_rf['Sleep Disorder'] = y
df_rf

# All features
all_features = df_balanced.drop(['Sleep Disorder'], axis=1).columns.tolist()

# Initialize comparison DataFrame
comparison_df = pd.DataFrame(columns=all_features, index=['f_classif', 'mutual_info_classif', 'Decision_Tree', 'Random_Forest'])

# Mark selected features
comparison_df.loc['f_classif', selected_features_rf] = '✓'
comparison_df.loc['mutual_info_classif', selected_features_rf] = '✓'
comparison_df.loc['Decision_Tree', selected_features_rf] = '✓'
comparison_df.loc['Random_Forest', selected_features_rf] = '✓'

# Fill NaN with empty string
comparison_df = comparison_df.fillna('')

# Display the DataFrame
plt.figure(figsize=(14, 7))

# Draw a heatmap with the mask and correct aspect ratio
sns.heatmap(comparison_df.replace('✓', 1).replace('', 0), annot=comparison_df, fmt='', cmap='Blues', cbar=False, linewidths=.5, linecolor='black')

plt.title('Feature Selection Comparison')
plt.xlabel('Features (X)')
plt.ylabel('Methods')
plt.show()

df_new = df_balanced[['Age', 'Occupation', 'Sleep Duration', 'Quality of Sleep', 'Physical Activity Level', 'BMI Category', 'Heart Rate', 'Daily Steps', 'BP Low', 'Sleep Disorder']]
df_new

```

```

# Uji Multikolinearitas setelah feature selection
# Menentukan variabel independen
X = df_new.drop(['Sleep Disorder'], axis=1)

# Select only numeric columns
X_numeric = X.select_dtypes(include=['float', 'int'])

# Menambahkan konstanta (intersep) ke dalam model
X_numeric = sm.add_constant(X_numeric)

# Menghitung VIF untuk setiap variabel
vif_data = pd.DataFrame()
vif_data["feature"] = X_numeric.columns
vif_data["VIF"] = [variance_inflation_factor(X_numeric.values, i) for i in range(X_numeric.shape[1])]

print(vif_data)

# Variabel
X = df_new.drop(['Sleep Disorder'], axis=1)
y = df_new['Sleep Disorder']

```

## DECISION TREE

```

from sklearn import tree
from sklearn.datasets import load_iris, load_diabetes
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import accuracy_score, mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the model
tree_model = DecisionTreeClassifier()

# Train the model
tree_model.fit(X_train, y_train)

# Predictions
y_pred_tm = tree_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred_tm)
print(f"Accuracy: {accuracy:.2f}")

Accuracy: 0.90

# Define feature_names and target_names
feature_names = X.columns.tolist()
target_names = y.unique().astype(str).tolist()

# Plot the decision tree
plt.figure(figsize=(20, 10))
tree.plot_tree(tree_model, filled=True, feature_names=feature_names, class_names=target_names)
plt.show()

```

```

importances = tree_model.feature_importances_

for feature, importance in zip(feature_names, importances):
    print(f"{feature}: {importance:.4f}")

# Creating Labels and indices sorted by importance
indices = np.argsort(importances)[::-1]
sorted_features = [feature_names[i] for i in indices]
sorted_importances = importances[indices]

# Plotting
plt.figure(figsize=(10, 6))
plt.title('Feature Importances in Decision Tree Classifier')
plt.bar(range(len(importances)), sorted_importances, align='center')
plt.xticks(range(len(importances)), sorted_features, rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.show()

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

tree_model = DecisionTreeClassifier()
grid_search = GridSearchCV(tree_model, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

Best parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 2}
Best cross-validation score: 0.86

# Initialize the model
tree_model = DecisionTreeClassifier(criterion='gini', max_depth=20, min_samples_leaf=1, min_samples_split=10)

# Train the model
tree_model.fit(X_train, y_train)

DecisionTreeClassifier(max_depth=20, min_samples_split=10)

```

```

# Predictions
y_pred_tm = tree_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred_tm)
print(f"Accuracy: {accuracy:.4f}")

Accuracy: 0.9048

# Define feature_names and target_names
feature_names = X.columns.tolist()
target_names = y.unique().astype(str).tolist()

# Plot the decision tree
plt.figure(figsize=(20, 10))
tree.plot_tree(tree_model, filled=True, feature_names=feature_names, class_names=target_names)
plt.show()

importances = tree_model.feature_importances_

for feature, importance in zip(feature_names, importances):
    print(f"{feature}: {importance:.4f}")

# Creating Labels and indices sorted by importance
indices = np.argsort(importances)[::-1]
sorted_features = [feature_names[i] for i in indices]
sorted_importances = importances[indices]

# Plotting
plt.figure(figsize=(10, 6))
plt.title('Feature Importances in Decision Tree Classifier')
plt.bar(range(len(importances)), sorted_importances, align='center')
plt.xticks(range(len(importances)), sorted_features, rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.show()

from scipy.stats import randint
param_dist = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': randint(2, 11),
    'min_samples_leaf': randint(1, 5)
}

random_search = RandomizedSearchCV(tree_model, param_distributions=param_dist, n_iter=100, cv=3, scoring='accuracy', random_state=42)
random_search.fit(X_train, y_train)

```

```

print("Best parameters:", random_search.best_params_)
print("Best cross-validation score: {:.4f}".format(random_search.best_score_))

Best parameters: {'criterion': 'entropy', 'max_depth': 50, 'min_samples_leaf': 3, 'min_samples_split': 10}
Best cross-validation score: 0.8571

# Initialize the model
tree_model = DecisionTreeClassifier(criterion='gini', max_depth=30, min_samples_leaf=1, min_samples_split=9)

# Train the model
tree_model.fit(X_train, y_train)

DecisionTreeClassifier(max_depth=30, min_samples_split=9)

# Predictions
y_pred_tm = tree_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred_tm)
print(f"Accuracy: {accuracy:.4f}")

Accuracy: 0.9048

df_dect = df_new[['Sleep Disorder', 'BP Low', 'BMI Category', 'Daily Steps', 'Physical Activity Level']]
df_dect

# Variabel importances
Xd = df_dect.drop(['Sleep Disorder'], axis=1)
yd = df_dect['Sleep Disorder']

X_train, X_test, y_train, y_test = train_test_split(Xd, yd, test_size=0.2, random_state=42)

# Initialize the model
tree_model = DecisionTreeClassifier()

# Train the model
tree_model.fit(X_train, y_train)

DecisionTreeClassifier()

# Predictions
y_pred = tree_model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

Accuracy: 0.90

# Define feature_names and target_names
feature_names = Xd.columns.tolist()
target_names = yd.unique().astype(str).tolist()

# Plot the decision tree
plt.figure(figsize=(20, 10))
tree.plot_tree(tree_model, filled=True, feature_names=feature_names, class_names=target_names)
plt.show()

```

```

from scipy.stats import randint
param_dist = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': randint(2, 11),
    'min_samples_leaf': randint(1, 5)
}

random_search = RandomizedSearchCV(tree_model, param_distributions=param_dist, n_iter=100, cv=3, scoring='accuracy', random_state=42)
random_search.fit(X_train, y_train)

print("Best parameters:", random_search.best_params_)
print("Best cross-validation score: {:.4f}".format(random_search.best_score_))

Best parameters: {'criterion': 'gini', 'max_depth': 40, 'min_samples_leaf': 3, 'min_samples_split': 4}
Best cross-validation score: 0.8690

# Initialize the model
tree_model = DecisionTreeClassifier(criterion='gini', max_depth=40, min_samples_leaf=3, min_samples_split=4)

# Train the model
tree_model.fit(X_train, y_train)

DecisionTreeClassifier(max_depth=40, min_samples_leaf=3, min_samples_split=4)

# Predictions
y_pred_dt3 = tree_model.predict(X_test)

# Evaluate accuracy
accuracy_dt3 = accuracy_score(y_test, y_pred_dt3)
print(f"Accuracy: {accuracy_dt3:.4f}")

Accuracy: 0.9048

```

```

# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt
import numpy as np

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Decision Tree model with best parameters
tree_model = DecisionTreeClassifier(criterion='gini', max_depth=40, min_samples_leaf=3, min_samples_split=4)
ovr_tree_model = OneVsRestClassifier(tree_model)
y_score = ovr_tree_model.fit(X_train, y_train).predict_proba(X_test)

# Binarize the output
y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute macro-average ROC curve and ROC area
# Aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

# Interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

# Average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(figsize=(10, 6))
plt.plot(fpr["macro"], tpr["macro"], color='navy', linestyle='--', linewidth=2, label='Macro-average ROC curve (area = {0:0.2f})'.format(roc_auc["macro"]))

for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Print AUC scores
print(f'Macro-average AUC: {roc_auc["macro"]:.4f}')
for i in range(n_classes):
    print(f'AUC of class {i}: {roc_auc[i]:.4f}')

```

```

from scipy.stats import randint
param_dist = {
    'n_estimators': randint(175, 225),
    'max_features': ['log2'],
    'max_depth': [None],
    'min_samples_split': randint(3, 7),
    'min_samples_leaf': randint(1, 2)
}

model = RandomForestClassifier()
random_search = RandomizedSearchCV(model, param_distributions=param_dist, n_iter=100, cv=3, scoring='accuracy')
random_search.fit(X_train, y_train)
print("Best parameters:", random_search.best_params_)
print("Best cross-validation score: {:.4f}".format(random_search.best_score_))
predictions = random_search.best_estimator_.predict(X_test)
accuracy_rf1 = accuracy_score(y_test, predictions)
print(f'Test Accuracy: {accuracy_rf1:.4f}')

```

## RANDOM FOREST

```

from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.metrics import accuracy_score, mean_squared_error

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
((168, 9), (42, 9), (168,), (42,))

```

```

model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

```

Accuracy: 0.9048

```

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

```

```

model = RandomForestClassifier()
grid_search = GridSearchCV(model, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.4f}".format(grid_search.best_score_))
predictions = grid_search.best_estimator_.predict(X_test)
accuracy_rf1 = accuracy_score(y_test, predictions)
print(f'Test Accuracy: {accuracy_rf1:.4f}')

```

```
importances = random_search.best_estimator_.feature_importances_

```

```

# Creating labels and indices sorted by importance
indices = np.argsort(importances)[::-1]
sorted_features = [X.columns[i] for i in indices]
sorted_importances = importances[indices]

```

```

# Plotting
plt.figure(figsize=(10, 6))
plt.title('Feature Importances in Random Forest Classifier')
plt.bar(range(len(importances)), sorted_importances, align='center')
plt.xticks(range(len(importances)), sorted_features, rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.show()

```

```

df_ranf = df_new[['BP Low', 'Age', 'Occupation', 'Physical Activity Level', 'Sleep Disorder']]
df_ranf

# Variabel
Xrf = df_ranf.drop(['Sleep Disorder'],axis=1)
yrf = df_ranf['Sleep Disorder']

X_train, X_test, y_train, y_test = train_test_split(Xrf, yrf, test_size=0.2, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
((168, 4), (42, 4), (168,), (42,))

model = RandomForestClassifier()
model.fit(X_train,y_train)
y_pred_rf = model.predict(X_test)
accuracy_ranf = accuracy_score(y_test, y_pred_rf)
print(f"Accuracy: {accuracy_ranf:.4f}")

Accuracy: 0.8333

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

model = RandomForestClassifier()
grid_search = GridSearchCV(model, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.4f}".format(grid_search.best_score_))
predictions = grid_search.best_estimator_.predict(X_test)
accuracy_ranf2 = accuracy_score(y_test, predictions)
print(f"Test Accuracy: {accuracy_ranf2:.4f}")

```

```

from scipy.stats import randint
param_dist = {
    'n_estimators': randint(175, 225),
    'max_features': ['log2'],
    'max_depth': [None],
    'min_samples_split': randint(3, 7),
    'min_samples_leaf': randint(1, 2)
}

model = RandomForestClassifier()
random_search = RandomizedSearchCV(model, param_distributions=param_dist, n_iter=100, cv=3, scoring='accuracy')
random_search.fit(X_train, y_train)
print("Best parameters:", random_search.best_params_)
print("Best cross-validation score: {:.4f}".format(random_search.best_score_))
predictions = random_search.best_estimator_.predict(X_test)
accuracy_ranf3 = accuracy_score(y_test, predictions)
print(f"Test Accuracy: {accuracy_ranf3:.4f}")

# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import roc_curve, auc, accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Assuming you have a dataset X and target variable y
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the Random Forest model with the specified best parameters
best_params = {
    'max_depth': None,
    'max_features': 'log2',
    'min_samples_leaf': 1,
    'min_samples_split': 5,
    'n_estimators': 192
}

rf_model = OneVsRestClassifier(RandomForestClassifier(**best_params, random_state=42))
rf_model.fit(X_train, y_train)

# Predict probabilities
y_score_rf = rf_model.predict_proba(X_test)

# Binarize the output
y_test_bin = LabelBinarizer().fit(y_test).transform(y_test)
n_classes = y_test_bin.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_rf[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute macro-average ROC curve and ROC area
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(figsize=(10, 6))
plt.plot(fpr["macro"], tpr["macro"], color='navy', linestyle='--', linewidth=2, label='Macro-average ROC curve (area = {:.2f})'.format(roc_auc["macro"]))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label='ROC curve of class {} (area = {:.2f})'.format(i, roc_auc[i]))

```

```

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Print AUC scores
print(f'Macro-average AUC: {roc_auc["macro"]:.2f}')
for i in range(n_classes):
    print(f'AUC of class {i}: {roc_auc[i]:.2f}')

# Predict Labels
y_pred_rf = rf_model.predict(X_test)

# Evaluate the Random Forest model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f'Accuracy: {accuracy_rf:.4f}')

# Confusion matrix and classification report
matrix = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(matrix, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print(classification_report(y_test, y_pred_rf))

```

## K-Nearest Neighbors (KNN)

```

# Import Libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
((168, 9), (42, 9), (168,), (42,))

#normalize the data
scaler = StandardScaler()
X_train_knn = scaler.fit_transform(X_train)
X_test_knn = scaler.transform(X_test)

#initialize the KNN algorithm
knn = KNeighborsClassifier(n_neighbors=5)

#train the KNN algorithm
knn.fit(X_train_knn, y_train)
KNeighborsClassifier()

```

```

#make predictions on the test set
y_pred_knn = knn.predict(X_test_knn)

#evaluate the performance of the KNN algorithm
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'Accuracy: {accuracy_knn:.4f}')
Accuracy: 0.9048

#confusion matrix and classification report
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix

matrix = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(matrix, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, y_pred_knn))

def find_best_k_with_grid_search(X_train, y_train, X_test, y_test, param_grid):
    # Create a KNN classifier
    knn = KNeighborsClassifier()

    # Perform GridSearchCV
    grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train_knn, y_train)

    # Get the best parameter
    best_k = grid_search.best_params_['n_neighbors']

    # Fit model with best k on entire training set
    best_knn = KNeighborsClassifier(n_neighbors=best_k)
    best_knn.fit(X_train_knn, y_train)

    # calculate accuracy on train and test set
    train_accuracy = best_knn.score(X_train_knn, y_train)
    test_accuracy = best_knn.score(X_test_knn, y_test)

    return grid_search.best_params_, train_accuracy, test_accuracy

```

```

# Assuming you have X_train, y_train variables
max_k=50

# Define a grid of hyperparameters
parameters = {'n_neighbors': range(2, max_k + 1)}

best_params, train_accuracy, test_accuracy = find_best_k_with_grid_search(X_train, y_train, X_test, y_test, parameters)
print(f'Best k: {best_params["n_neighbors"]}')
print(f'Train Accuracy with Best k: {train_accuracy*100:.2f}%')
print(f'Test Accuracy with Best k: {test_accuracy*100:.2f}%')

Best k: 5
Train Accuracy with Best k: 89.29%
Test Accuracy with Best k: 90.48%

```

```

# Import necessary Libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import roc_curve, auc, accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
X_train_knn = scaler.fit_transform(X_train)
X_test_knn = scaler.transform(X_test)

# Initialize and fit the KNN model with OneVsRestClassifier
knn_model = OneVsRestClassifier(KNeighborsClassifier(n_neighbors=5))
knn_model.fit(X_train_knn, y_train)

# Predict probabilities
y_score = knn_model.predict_proba(X_test_knn)

# Binarize the output
y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute macro-average ROC curve and ROC area
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(figsize=(10, 6))
plt.plot(fpr["macro"], tpr["macro"], color='navy', linestyle='--', linewidth=2, label='Macro-average ROC curve (area = {0:.2f})'.format(roc_auc["macro"]))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label='ROC curve of class {0} (area = {1:.2f})'.format(i, roc_auc[i]))

```



```

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Print AUC scores
print(f'Macro-average AUC: {roc_auc["macro"]:.2f}')
for i in range(n_classes):
    print(f'AUC of class {i}: {roc_auc[i]:.2f}')

# Evaluate the KNN model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'Accuracy: {accuracy_knn:.4f}')

# Confusion matrix and classification report
matrix = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(matrix, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print(classification_report(y_test, y_pred_knn))

```

## SUPPORT VECTOR MACHINE

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
((168, 9), (42, 9), (168,), (42,))

model = SVC()
model.fit(X_train, y_train)
SVC()

grid_search.fit(X_train, y_train)

# View the best parameters found by GridSearchCV
best_parameters_svm = grid_search.best_params_
print(f"Best parameters found: {best_parameters_svm}")

# Use the best estimator to make predictions
best_estimator_svm = grid_search.best_estimator_
test_accuracy_svm = best_estimator_svm.score(X_test, y_test)
print(f"Test accuracy of the best estimator: {test_accuracy_svm}")
```

```

# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100], # Regularization parameter
    'gamma': ['scale', 'auto'], # Kernel coefficient
    'kernel': ['poly'] # Kernel types to try
}

# Create a base model
svc = SVC()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=svc, param_grid=param_grid,
                           cv=10, verbose=2, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# View the best parameters found by GridSearchCV
best_parameters_svm = grid_search.best_params_
print(f"Best parameters found: {best_parameters_svm}")

# Use the best estimator to make predictions
best_estimator_svm = grid_search.best_estimator_
test_accuracy_svm = best_estimator_svm.score(X_test, y_test)
print(f"Test accuracy of the best estimator: {test_accuracy_svm}")


# Inisialisasi model SVM dengan parameter C dan gamma
C_value = 1 # Nilai C
gamma_value = 'auto' # Nilai gamma

svm = SVC(kernel = 'rbf', C=C_value, gamma=gamma_value)

# Latih model SVM pada set pelatihan
svm.fit(X_train, y_train)

# Lakukan prediksi pada set pengujian
y_pred_svm = svm.predict(X_test)

# Evaluasi kinerja model (misalnya, akurasi)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f'Akurasi: {accuracy_svm}')


Akurasi: 0.9047619047619048

svc=SVC()

svc.fit(X_train,y_train)
y_pred_svm2 = svc.predict(X_test)
print('Model accuracy score with default hyperparameters: {:.4f}'.format(accuracy_score(y_test, y_pred_svm2)))

Model accuracy score with default hyperparameters: 0.6190

```

## SVM with linear kernel dan C = 1.0

```
# instantiate classifier with Linear kernel and C=1.0
linear_svc=SVC(kernel='linear', C=1.0, gamma = 'scale')

# fit classifier to training set
linear_svc.fit(X_train, y_train)

# make predictions on test set
y_pred_svm_linear = linear_svc.predict(X_test)

# compute and print accuracy score
print('Model accuracy score with linear kernel : {0:0.4f}'. format(accuracy_score(y_test, y_pred_svm_linear))
Model accuracy score with linear kernel : 0.7619
```

## SVM with RBF

```
clf_rbf = SVC(kernel='rbf', C = 1, gamma = 'auto')
clf_rbf.fit(X_train, y_train)

# make predictions on test set
y_pred_rbf = clf_rbf.predict(X_test)

# compute and print accuracy score
print('Model accuracy score with Poly kernel : {0:0.4f}'. format(accuracy_score(y_test, y_pred_rbf)))
Model accuracy score with Poly kernel : 0.9048
```

```
confusion_matrix(y_test, y_pred_rbf)
```

```
print(classification_report(y_test, y_pred_rbf))
```

```
print("Unique values in y_test:", set(y_test))
```

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, roc_curve, auc
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.multiclass import OneVsRestClassifier
import matplotlib.pyplot as plt
import numpy as np

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale the features
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Fit the SVM model with OneVsRestClassifier
svm_model = OneVsRestClassifier(SVC(kernel='rbf', C=1, gamma='auto', probability=True))
svm_model.fit(X_train_scaled, y_train)

# Predict probabilities
y_score = svm_model.predict_proba(X_test_scaled)

# Binarize the output
y_test_bin = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_bin.shape[1]

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute macro-average ROC curve and ROC area
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure(figsize=(10, 6))
plt.plot(fpr["macro"], tpr["macro"], color='navy', linestyle='--', linewidth=2, label="Macro-average ROC curve (area = {0:0.2f})".format(roc_auc["macro"]))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Print AUC scores
print(f'Macro-average AUC: {roc_auc["macro"]:.2f}')
for i in range(n_classes):
    print(f'AUC of class {i}: {roc_auc[i]:.2f}')
```

# TERIMA KASIH

