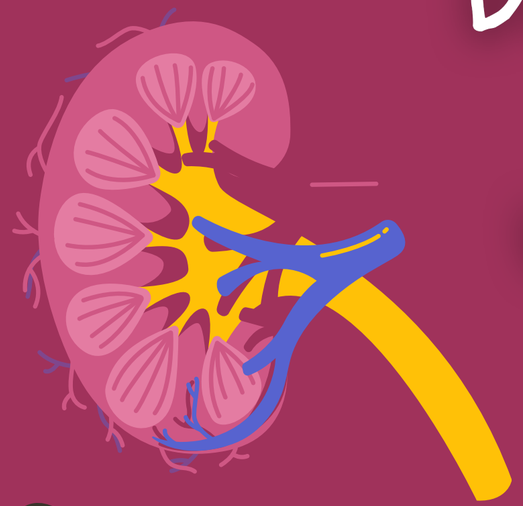
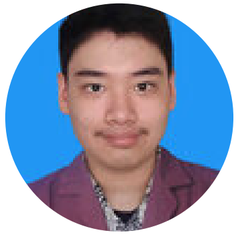
Proteomics Research

# ANALISIS SAMPEL DARAH DENGAN MACHINE LEARNING PREDICTIVE ALGORITHMS UNTUK PREDIKSI PENYAKIT GAGAL GINJAL



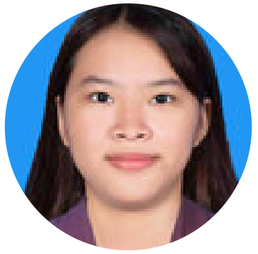
Kelompok 7

# ANGGOTA



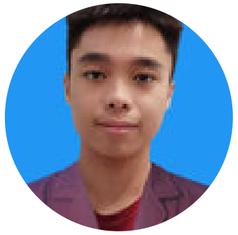
2602111723

Matheus Ariel Reinhart Sidharta



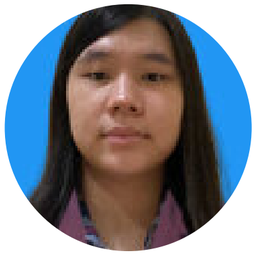
2602084016

Misia Callista Abdipatra



2602091053

Raffa Winters

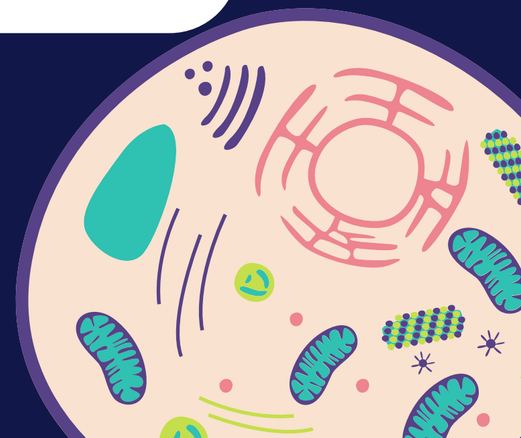


2602090731

Jesika Purnomo

# APA ITU PROTEIN?

**Molekul biologis yang terdiri dari rantai polipeptida (tersusun dari rantai asam amino yang terhubung bersama oleh ikatan peptida)**

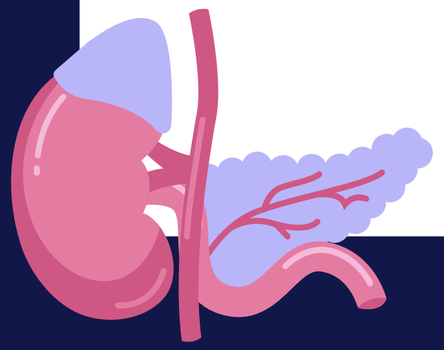


# FUNGSI PROTEIN

- **Membantu dan Memelihara Jaringan Tubuh**
- **Memproduksi Enzim dan Hormon**
- **Membantu Memperkuat Sistem Kekebalan Tubuh**

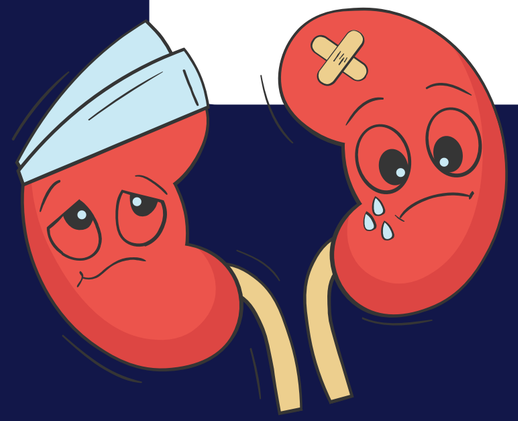
# PENGARUH STRUKTUR PROTEIN TERHADAP FUNGSI ORGAN TUBUH

**Struktur protein memungkinkan interaksi spesifik dengan molekul lain. Gangguan struktur protein dapat menyebabkan kehilangan fungsi, berdampak pada kesehatan dan fungsi organ tubuh.**



# PENYAKIT GINJAL

**Saat ginjal rusak dan sudah tidak bisa menyaring darah**



## KENAPA?

- **Diabetes**
- **Hipertensi**

# PENGOBATAN TERBAIK UNTUK PENYAKIT GINJAL?

**Deteksi Dini!**

- **Diet**
- **Olahraga**
- **Ubah gaya hidup**
- **Mencegah penyebab (Diabetes & Hipertensi)**

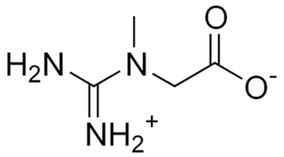
# HUBUNGAN PROTEIN DAN PENYAKIT GAGAL GINJAL

**Gagal Ginjal mengakibatkan Proteinuria, yaitu kondisi dimana protein bocor ke dalam urin karena ginjal tak lagi mampu menyaringnya dengan baik.**

**Hal ini dapat mempengaruhi metabolisme protein secara keseluruhan, termasuk menurunkan produksi protein yang penting untuk kesehatan tubuh.**

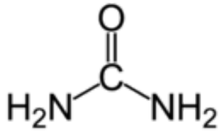
# Metode Tradisional

## Tes Darah



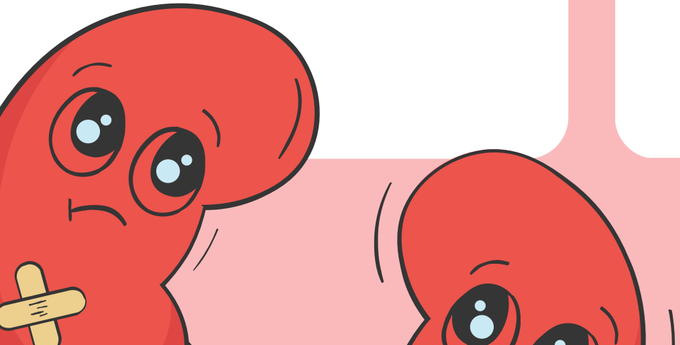

### TES KREATININ

Mengukur kadar kreatinin dalam darah. Kadar kreatinin yang tinggi menunjukkan penurunan fungsi ginjal.

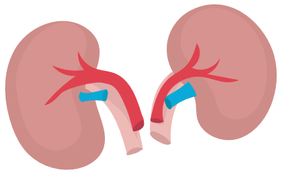



### TES UREUM

Mengukur kadar ureum dalam darah. Kadar ureum yang tinggi juga menunjukkan penurunan fungsi ginjal.

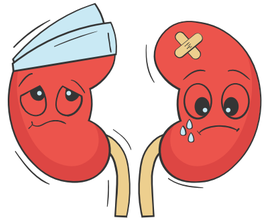


## Tes Urin



### TES GLOMERULAR FILTRATION RATE (GFR)

Memperkirakan tingkat penyaringan ginjal dengan mengukur kadar kreatinin dan faktor lain. GFR yang rendah menunjukkan penurunan fungsi ginjal.



### URINALYSIS

Memeriksa urin untuk protein, darah, dan sel abnormal, yang dapat mengindikasikan kerusakan ginjal.

# Limitasi Metode Tradisional

## Proses yang kompleks dan memakan waktu

Memerlukan peralatan dan personel laboratorium khusus, serta waktu tunggu yang lama untuk hasil tes.

## Keakuratan terbatas pada tahap awal

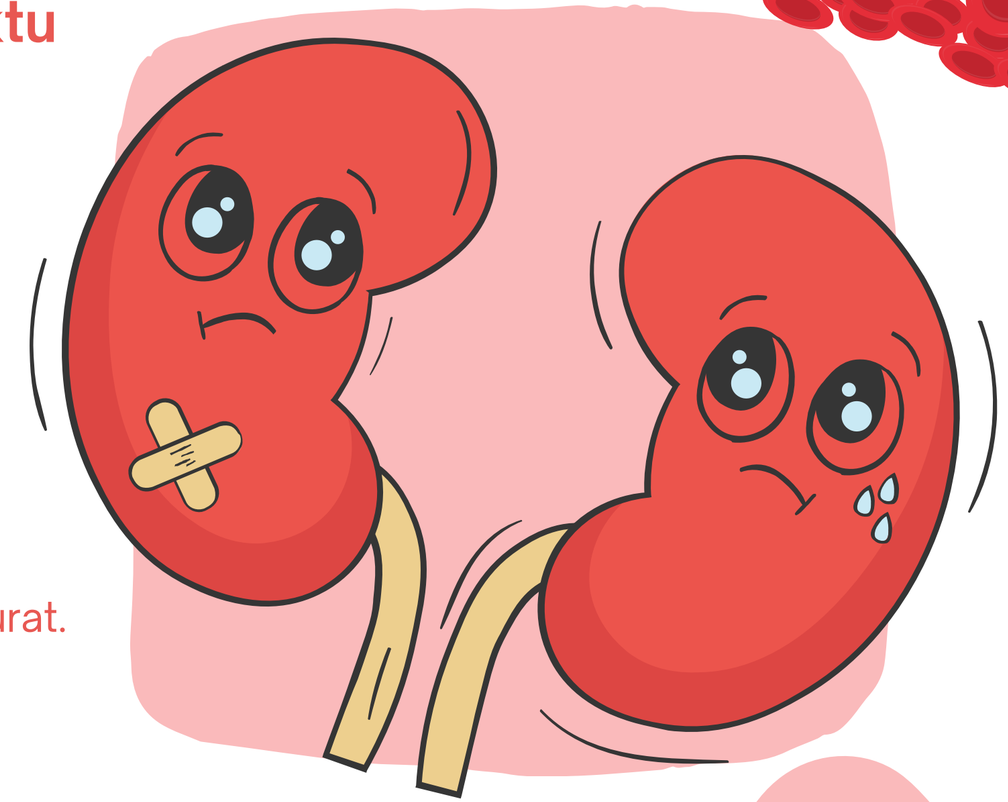
Tes tradisional mungkin tidak selalu akurat, terutama pada tahap awal penyakit ginjal.

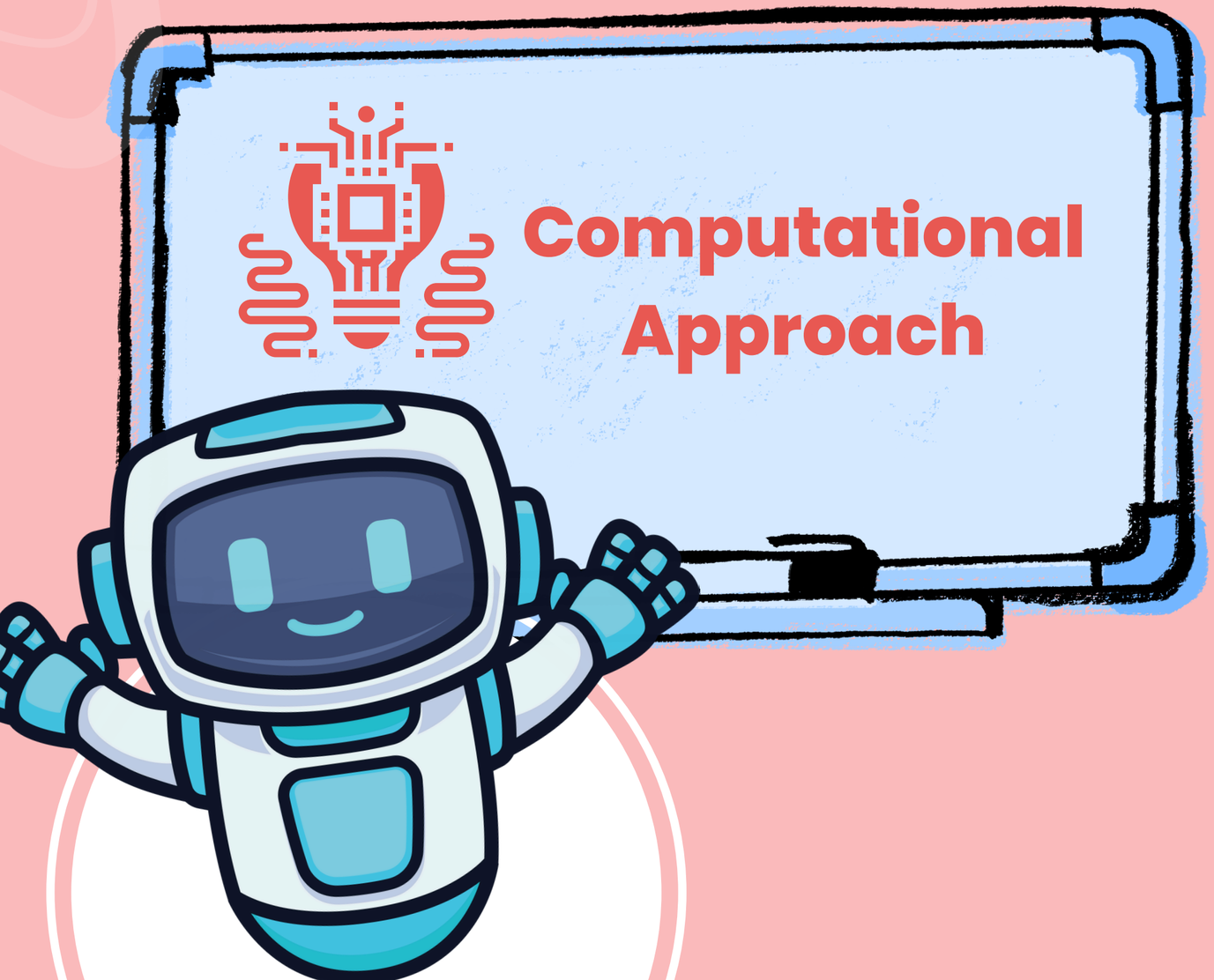
## Interpretasi yang sulit pada tahap awal

Hasil tes seringkali sulit diinterpretasikan, dan memerlukan keahlian medis untuk mendiagnosis gagal ginjal secara akurat.

## Kurang sensitif

Tes tradisional mungkin tidak dapat mendeteksi kerusakan ginjal dini, yang dapat menunda diagnosis dan pengobatan.

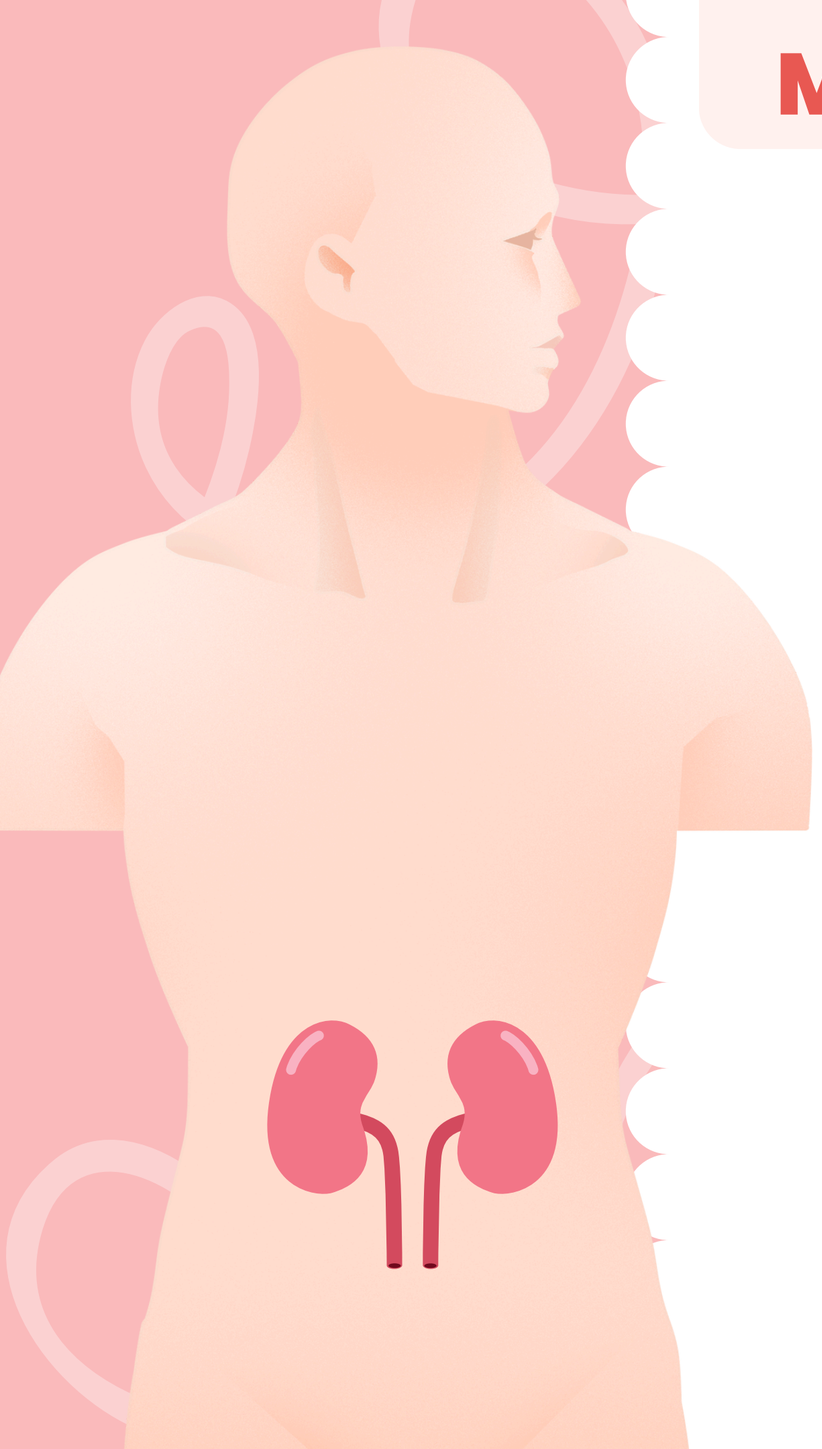





## MACHINE LEARNING

- **Bidang** kecerdasan buatan yang memungkinkan komputer untuk belajar dari data dan membuat prediksi. Dalam konteks prediksi gagal ginjal, machine learning digunakan untuk menganalisis data yang didapat dari dataset. Algoritma machine learning dapat mengidentifikasi pola dan hubungan dalam data yang terkait dengan gagal ginjal, dan menggunakan informasi ini untuk memprediksi risiko gagal ginjal di masa depan.

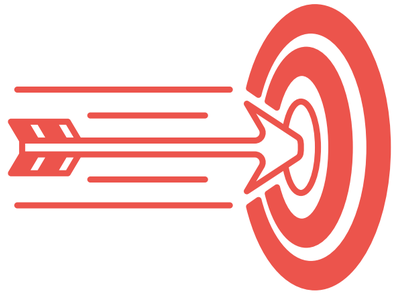
- Data yang didapat dari database akan diproses menggunakan algoritma predictive machine learning kami.



# Mengapa Computational Approach ?

## Meningkatkan sensitivitas pendeteksian

Algoritma machine learning  dapat menganalisis data tes darah dengan lebih sensitive, dengan mempelajari sumber data yang begtu banyak, sehingga didapat pola / ciri yag sesuai.



## Mempercepat diagnosis

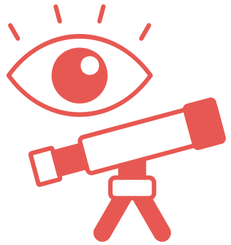
Analisis data yang lebih cepat dapat mempersingkat waktu tunggu untuk diagnosis dan pengobatan.

## Skalabilitas

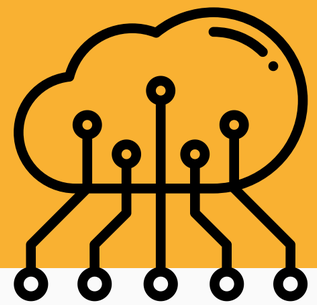
Memungkinkan peneliti untuk mempelajari data atau memproses dan menganalisis data yang berjumlah besar dengan waktu yang lebih singkat.

## Memungkinkan prediksi

Computational approach dapat digunakan untuk memprediksi risiko gagal ginjal di masa depan, memungkinkan intervensi pencegahan dini.



# Our Progress

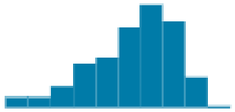
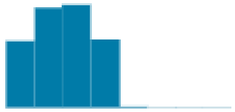



## Dataset Information

### Additional Information

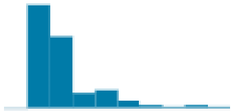
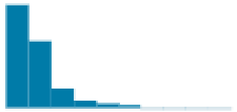
We use the following representation to collect the dataset

age - age
bp - blood pressure
sg - specific gravity
al - albumin
su - sugar
rbc - red blood cells
pc - pus cell
pcc - pus cell clumps
ba - bacteria
bgr - blood glucose random
bu - blood urea
sc - serum creatinine
sod - sodium
pot - potassium
hemo - hemoglobin
pcv - packed cell volume
wc - white blood cell count
rc - red blood cell count
htn - hypertension
dm - diabetes mellitus
cad - coronary artery disease
appet - appetite
pe - pedal edema
ane - anemia
class - class

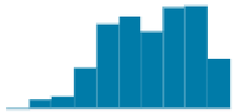
| id | # age | # bp | # sg | # al |
|---|---|---|---|---|
|  |  |  |  |  |
| 0 399 | 2 90 | 50 180 | 1 1.02 | 0 5 |

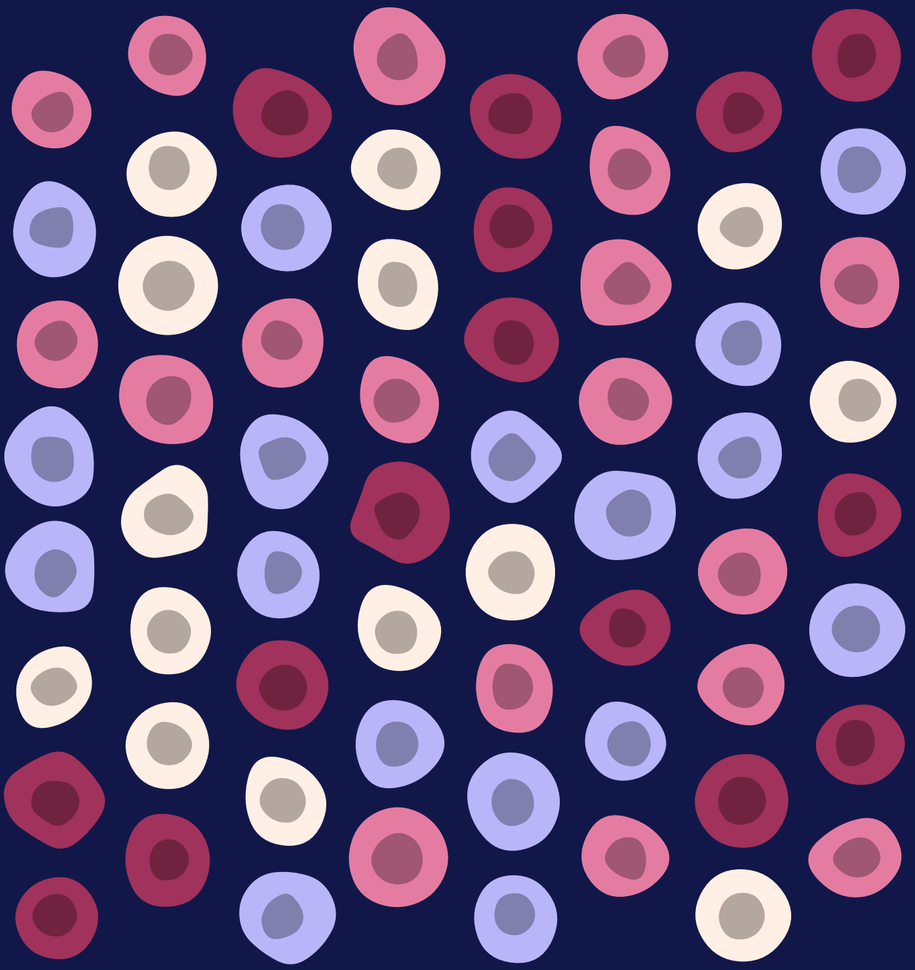
| # su | ▵ rbc | | ▵ pc | | ▵ pcc | | ▵ ba | |
|---|---|---|---|---|---|---|---|---|
|  | normal | 50% | normal | 65% | notpresent | 89% | notpresent | 94% |
| 0 5 | [null] | 38% | abnormal | 19% | present | 11% | present | 6% |
| | Other (47) | 12% | Other (65) | 16% | Other (4) | 1% | Other (4) | 1% |

| ▵ dm | | ▵ cad | | ▵ appet | |
|---|---|---|---|---|---|
| no | 65% | no | 91% | good | 79% |
| yes | 34% | yes | 9% | poor | 21% |
| Other (8) | 2% | Other (4) | 1% | Other (1) | 0% |

| # bgr | # bu | # sc | # sod | # pot |
|---|---|---|---|---|
|  |  |  |  |  |
| 22 490 | 1.5 391 | 0.4 76 | 4.5 163 | 2.5 47 |

| ✓ pe | | ✓ ane | | ▵ classification | |
|---|---|---|---|---|---|
|  | true 0 0% |  | true 0 0% | ckd | 62% |
| | false 0 0% | | false 0 0% | notckd | 38% |
| | [null] 1 0% | | [null] 1 0% | Other (2) | 1% |

| # hemo | ▵ pcv | | ▵ wc | | ▵ rc | | ✓ htn | |
|---|---|---|---|---|---|---|---|---|
|  | [null] | 18% | [null] | 26% | [null] | 33% |  | true 0 0% |
| 3.1 17.8 | 52 | 5% | 9800 | 3% | 5.2 | 5% | | false 0 0% |
| | Other (309) | 77% | Other (284) | 71% | Other (252) | 63% | | [null] 2 1% |



# LINK CODE

# PREDIKSI PENYAKIT GAGAL GINJAL

**Memuat dan menampilkan kolom-kolom yang ada pada dataset serta beberapa baris data pertama**

```
# loading data

df= pd.read_csv('../input/ckdisease/kidney_disease.csv')
df.head()
```

| | id | age | bp | sg | al | su | rbc | pc | pcc | ba | bgr | bu | sc | sod | pot | hemo | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 48.0 | 80.0 | 1.020 | 1.0 | 0.0 | NaN | normal | notpresent | notpresent | 121.0 | 36.0 | 1.2 | NaN | NaN | 15.4 | 44 | 7800 | 5.2 | yes | yes | no | good | no | no | ckd |
| 1 | 1 | 7.0 | 50.0 | 1.020 | 4.0 | 0.0 | NaN | normal | notpresent | notpresent | NaN | 18.0 | 0.8 | NaN | NaN | 11.3 | 38 | 6000 | NaN | no | no | no | good | no | no | ckd |
| 2 | 2 | 62.0 | 80.0 | 1.010 | 2.0 | 3.0 | normal | normal | notpresent | notpresent | 423.0 | 53.0 | 1.8 | NaN | NaN | 9.6 | 31 | 7500 | NaN | no | yes | no | poor | no | yes | ckd |
| 3 | 3 | 48.0 | 70.0 | 1.005 | 4.0 | 0.0 | normal | abnormal | present | notpresent | 117.0 | 56.0 | 3.8 | 111.0 | 2.5 | 11.2 | 32 | 6700 | 3.9 | yes | no | no | poor | yes | yes | ckd |
| 4 | 4 | 51.0 | 80.0 | 1.010 | 2.0 | 0.0 | normal | normal | notpresent | notpresent | 106.0 | 26.0 | 1.4 | NaN | NaN | 11.6 | 35 | 7300 | 4.6 | no | no | no | good | no | no | ckd |

# PREDIKSI PENYAKIT GAGAL GINJAL

**Menamakan ulang kolom-kolom agar lebih mudah dimengerti**

```
# rename column names agar lebih user-friendly

df.columns = ['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar', 'red_blood_cells', 'pus_cell',
              'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium',
              'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_count',
              'hypertension', 'diabetes_mellitus', 'coronary_artery_disease', 'appetite', 'peda_edema',
              'aanemia', 'class']
```

# PREDIKSI PENYAKIT GAGAL GINJAL

**Mengubah kolom 'packed_cell_volume', 'white_blood_cell_count' dan 'red_blood_cell_count' dari  tipe objek menjadi tipe numerik.**

```
[ ]  # converting necessary columns to numerical type

     df['packed_cell_volume'] = pd.to_numeric(df['packed_cell_volume'], errors='coerce')
     df['white_blood_cell_count'] = pd.to_numeric(df['white_blood_cell_count'], errors='coerce')
     df['red_blood_cell_count'] = pd.to_numeric(df['red_blood_cell_count'], errors='coerce')
```

# PREDIKSI PENYAKIT GAGAL GINJAL

**Menyimpan kolom-kolom kategorikal dan numerik dalam variable tersendiri**

```
[ ]  # Extracting categorical and numerical columns

     cat_cols = [col for col in df.columns if df[col].dtype == 'object']
     num_cols = [col for col in df.columns if df[col].dtype != 'object']
```

**Mencari nilai-nilai unik dalam kolom-kolom kategorikal**

```
[ ]  # Mencari unique values dalam categorical columns

     for col in cat_cols:
         print(f"{col} has {df[col].unique()} values\n")
```

# PREDIKSI PENYAKIT GAGAL GINJAL

**Menggantikan nilai-nilai incorrect pada beberapa kolom**

```
# Menggantikan incorrect values

df['diabetes_mellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes':'yes'},inplace=True)

df['coronary_artery_disease'] = df['coronary_artery_disease'].replace(to_replace = '\tno', value='no')

df['class'] = df['class'].replace(to_replace = {'ckd\t': 'ckd', 'notckd': 'not ckd'})
```

```
[ ] df['class'] = df['class'].map({'ckd': 0, 'not ckd': 1})
    df['class'] = pd.to_numeric(df['class'], errors='coerce')
```

```
[ ] cols = ['diabetes_mellitus', 'coronary_artery_disease', 'class']

    for col in cols:
        print(f"{col} has {df[col].unique()} values\n")
```

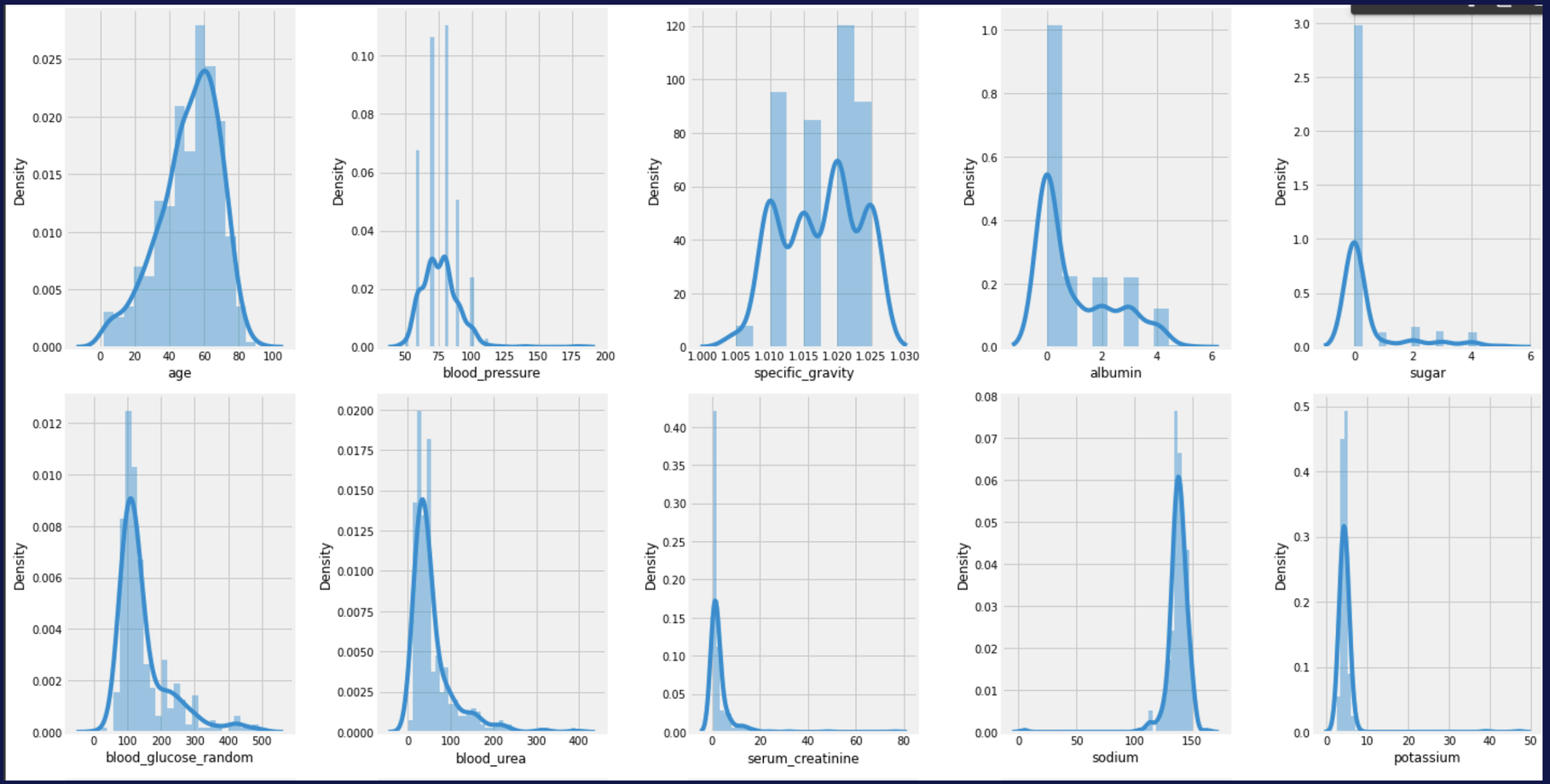
```
diabetes_mellitus has ['yes' 'no' nan] values

coronary_artery_disease has ['no' 'yes' nan] values

class has [0 1] values
```

# PREDIKSI PENYAKIT GAGAL GINJAL

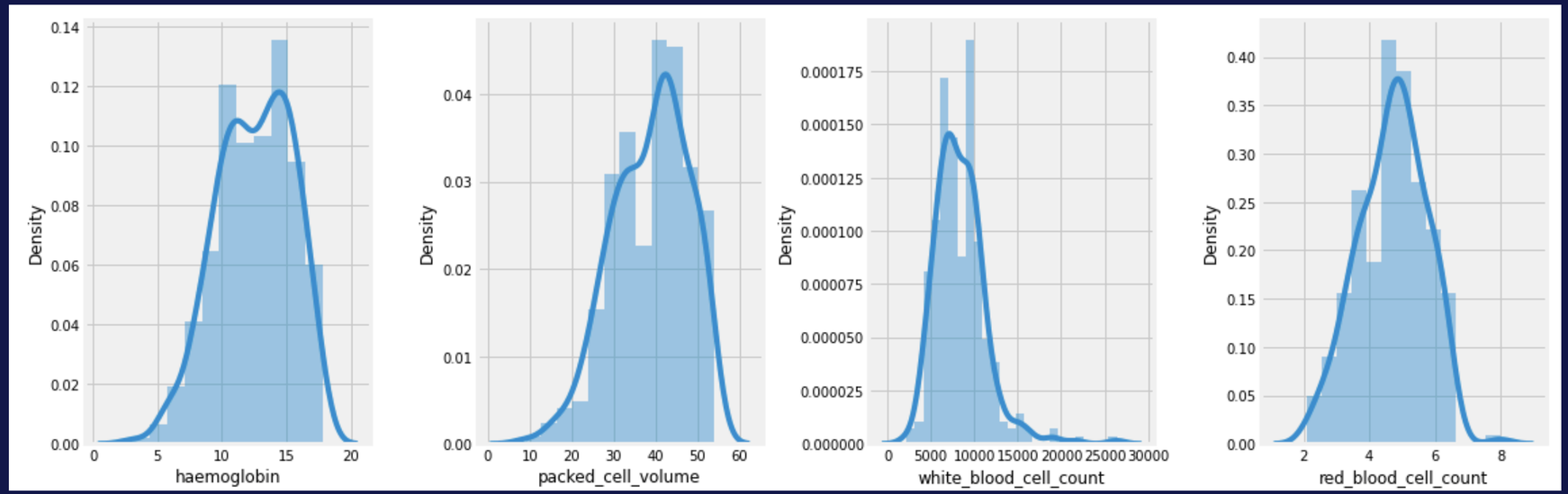
```
# memeriksa distribusi numerical

plt.figure(figsize = (20, 15))
plotnumber = 1

for column in num_cols:
    if plotnumber <= 14:
        ax = plt.subplot(3, 5, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column)

    plotnumber += 1

plt.tight_layout()
plt.show()
```

**Memeriksa distribusi numerik pada kolom-kolom numerik**

# PREDIKSI PENYAKIT GAGAL GINJAL



# PREDIKSI PENYAKIT GAGAL GINJAL



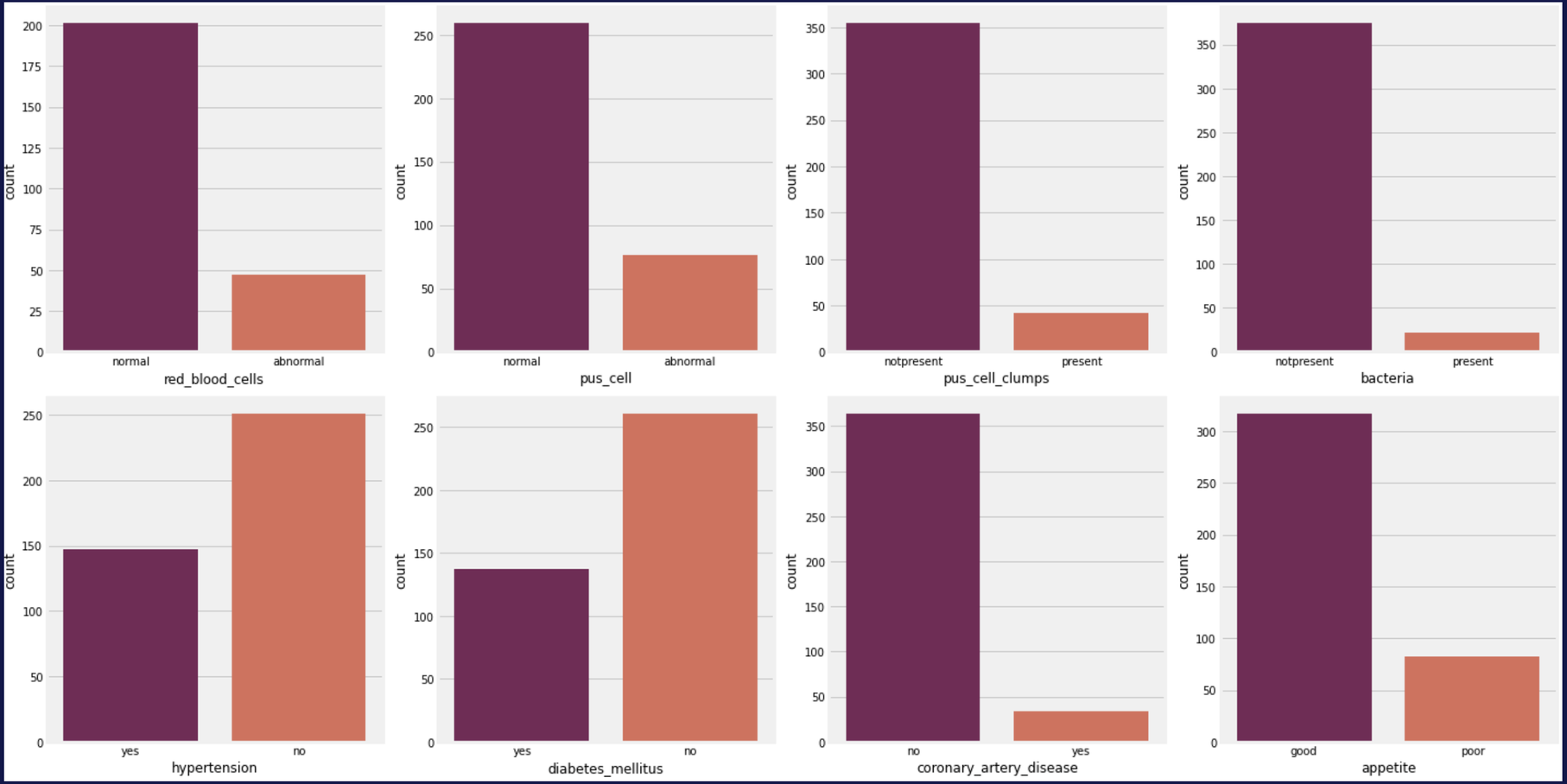
# PREDIKSI PENYAKIT GAGAL GINJAL

Skewness atau tidak simetris distribusi data terdapat dalam beberapa kolom.

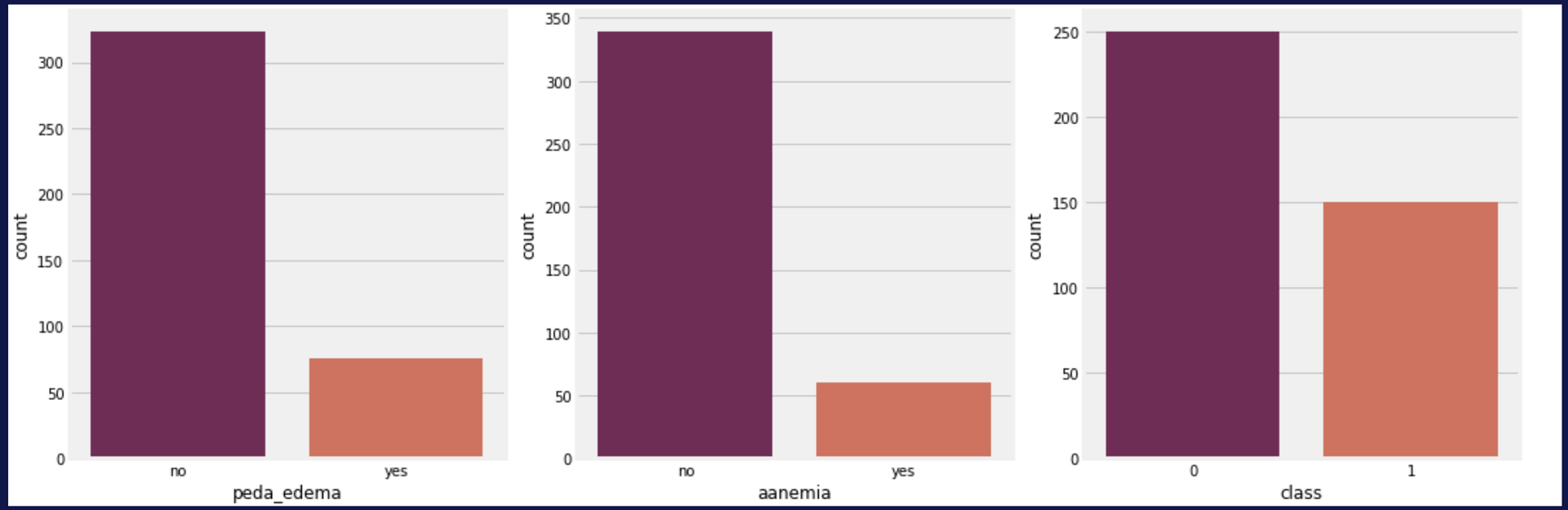
```
# Memperhatikan categorical columns

plt.figure(figsize = (20, 15))
plotnumber = 1

for column in cat_cols:
    if plotnumber <= 11:
        ax = plt.subplot(3, 4, plotnumber)
        sns.countplot(df[column], palette = 'rocket')
        plt.xlabel(column)

    plotnumber += 1

plt.tight_layout()
plt.show()
```

**Memeriksa kolom-kolom kategorikal**

# PREDIKSI PENYAKIT GAGAL GINJAL



# PREDIKSI PENYAKIT GAGAL GINJAL

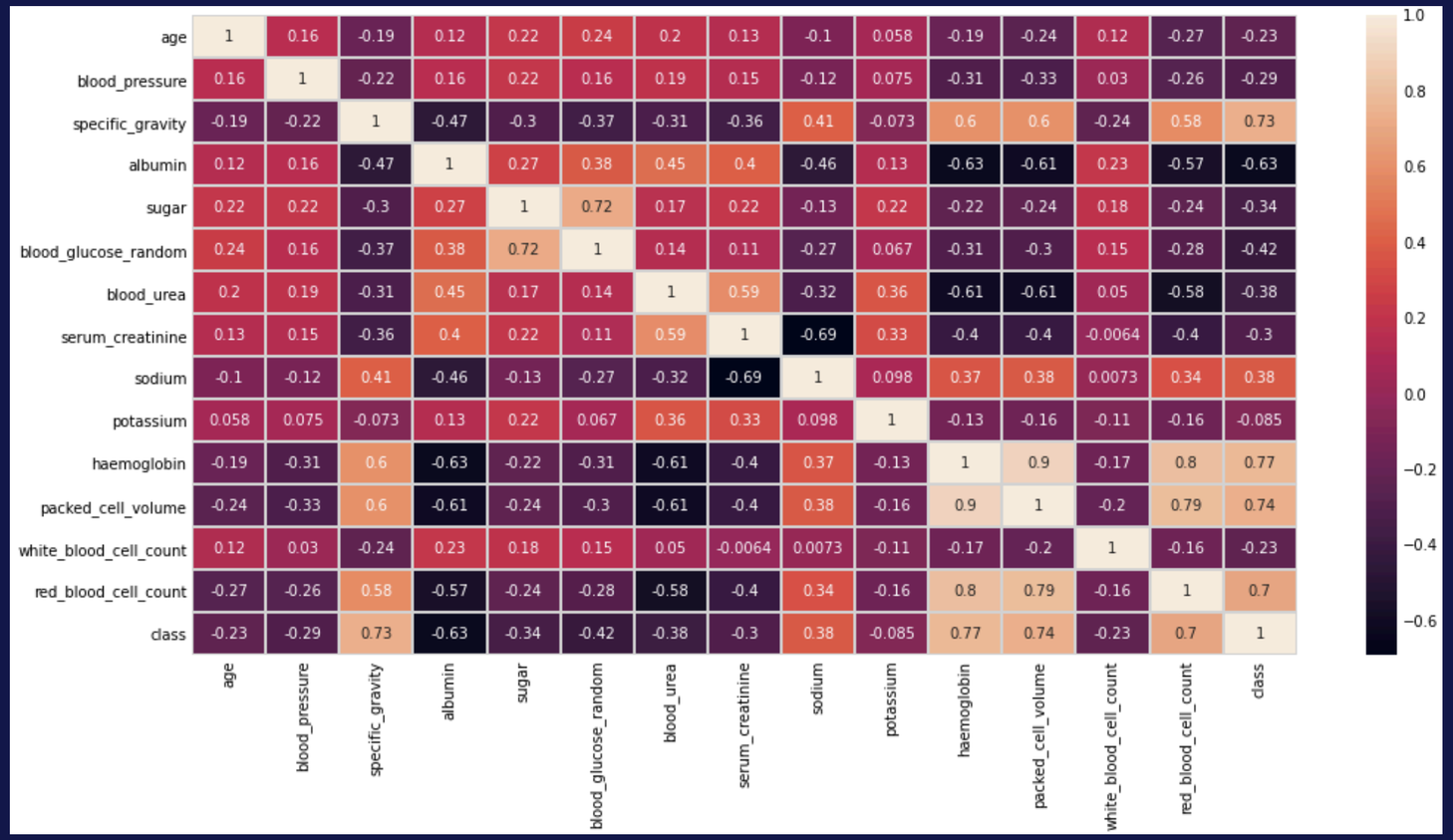


# PREDIKSI PENYAKIT GAGAL GINJAL

**Membuat ‘Heatmap’ untuk menunjukan korelasi antara variable-variable secara keseluruhan**

```
[ ]  # heatmap datanya

     plt.figure(figsize = (15, 8))

     sns.heatmap(df.corr(), annot = True, linewidths = 2, linecolor = 'lightgrey')
     plt.show()
```

# PREDIKSI PENYAKIT GAGAL GINJAL



# EXPLORATORY DATA ANALYSIS

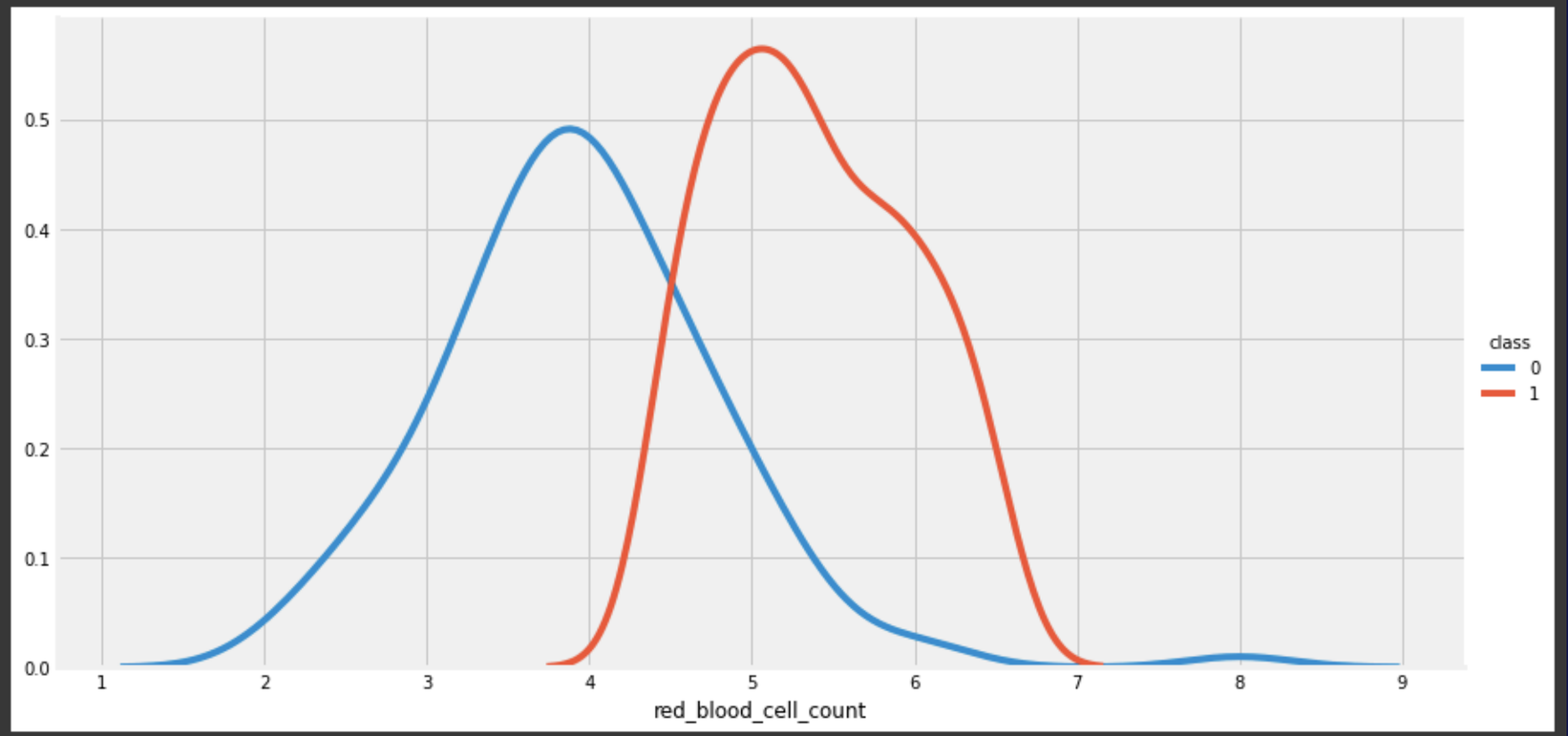
EDA merupakan pendekatan dalam analisis data yang bertujuan untuk memahami karakteristik utama dari data yang diberikan dan memberikan wawasan awal tentang pola, hubungan, anomali, dan potensi masalah yang terdapat dalam data tersebut.

```
# Mendefinisikan fungsi untuk membuat plot

def violin(col):
    fig = px.violin(df, y=col, x="class", color="class", box=True, template = 'plotly_dark')
    return fig.show()

def kde(col):
    grid = sns.FacetGrid(df, hue="class", height = 6, aspect=2)
    grid.map(sns.kdeplot, col)
    grid.add_legend()

def scatter(col1, col2):
    fig = px.scatter(df, x=col1, y=col2, color="class", template = 'plotly_dark')
    return fig.show()
```

# EXPLORATORY DATA ANALYSIS

Contoh penggunaan

```
kde('red_blood_cell_count')
```



# PRE PROCESSING

Serangkaian langkah atau proses yang **dilakukan untuk mempersiapkan data mentah (raw data) sebelum dilakukan analisis atau pemodelan.**

Tujuan dari pre-processing data adalah untuk membersihkan, mengorganisir, dan mengubah data sehingga data tersebut dapat digunakan secara efektif dalam analisis atau pemodelan.

## Memeriksa nilai ‘kosong’ dalam file data

```
[ ]  # memeriksa null values

     df.isna().sum().sort_values(ascending = False)
```

```
red_blood_cells             152
red_blood_cell_count        131
white_blood_cell_count      106
potassium                    88
sodium                       87
packed_cell_volume           71
pus_cell                     65
haemoglobin                  52
sugar                        49
specific_gravity             47
albumin                      46
blood_glucose_random         44
blood_urea                   19
serum_creatinine             17
blood_pressure               12
age                           9
bacteria                      4
pus_cell_clumps               4
hypertension                  2
diabetes_mellitus             2
coronary_artery_disease       2
appetite                      1
peda_edema                    1
aanemia                       1
class                         0
dtype: int64
```

# PRE PROCESSING

**Memeriksa nilai ‘kosong’ dalam kolom-kolom numerik**

```
df[num_cols].isnull().sum()
```

```
age                        9
blood_pressure            12
specific_gravity          47
albumin                   46
sugar                     49
blood_glucose_random      44
blood_urea                19
serum_creatinine          17
sodium                    87
potassium                 88
haemoglobin               52
packed_cell_volume        71
white_blood_cell_count   106
red_blood_cell_count     131
dtype: int64
```

**Memeriksa nilai ‘kosong’ dalam kolom-kolom non-numerik**

```
df[cat_cols].isnull().sum()
```

```
red_blood_cells          152
pus_cell                  65
pus_cell_clumps            4
bacteria                   4
hypertension               2
diabetes_mellitus          2
coronary_artery_disease    2
appetite                   1
peda_edema                 1
aanemia                    1
class                      0
dtype: int64
```

# PRE PROCESSING

## Mendefinisikan fungsi metode Random Sampling dan Mean/Mode Sampling

```
# mengisi nilai null, kita akan menggunakan dua metode, random sampling method untuk nilai null yang lebih tinggi dan
# mean/mode sampling untuk nilai null yang lebih rendah

def random_value_imputation(feature):
    random_sample = df[feature].dropna().sample(df[feature].isna().sum())
    random_sample.index = df[df[feature].isnull()].index
    df.loc[df[feature].isnull(), feature] = random_sample

def impute_mode(feature):
    mode = df[feature].mode()[0]
    df[feature] = df[feature].fillna(mode)
```

# PRE PROCESSING

**Menggunakan fungis Random Sampling untuk mengisi nilai ‘kosong’  yang lebih tinggi.**

```
[ ]  # mengisi num_cols null values menggunakan random sampling method

     for col in num_cols:
         random_value_imputation(col)
```

```
df[num_cols].isnull().sum()
```

```
age                          0
blood_pressure               0
specific_gravity             0
albumin                      0
sugar                        0
blood_glucose_random         0
blood_urea                   0
serum_creatinine             0
sodium                       0
potassium                    0
haemoglobin                  0
packed_cell_volume           0
white_blood_cell_count       0
red_blood_cell_count         0
dtype: int64
```

# PRE PROCESSING

**Menggunakan fungsi Mean/Mode Sampling untuk mengisi nilai ‘kosong’ yang lebih rendah.**

```
# menisi "red_blood_cells" and "pus_cell" menggunakan random sampling method dan
# sisa dari cat_cols menggunakan mode imputation

random_value_imputation('red_blood_cells')
random_value_imputation('pus_cell')

for col in cat_cols:
    impute_mode(col)
```

```
df[cat_cols].isnull().sum()
```

```
red_blood_cells             0
pus_cell                    0
pus_cell_clumps             0
bacteria                    0
hypertension                0
diabetes_mellitus           0
coronary_artery_disease     0
appetite                    0
peda_edema                  0
aanemia                     0
class                       0
dtype: int64
```

# FEATURE ENCODING

Proses **mengubah data kategori menjadi bentuk yang dapat diproses oleh algoritma pembelajaran mesin.**

Data kategori sering kali muncul dalam bentuk teks atau string, sedangkan algoritma pembelajaran mesin umumnya membutuhkan input berupa nilai numerik.

## Memeriksa jumlah kategori

```
[ ]  for col in cat_cols:
         print(f"{col} has {df[col].nunique()} categories\n")
```

```
red_blood_cells has 2 categories

pus_cell has 2 categories

pus_cell_clumps has 2 categories

bacteria has 2 categories

hypertension has 2 categories

diabetes_mellitus has 2 categories

coronary_artery_disease has 2 categories

appetite has 2 categories

peda_edema has 2 categories

aanemia has 2 categories

class has 2 categories
```

# FEATURE ENCODING

**Mengubah kategori bentuk string menjadi bentuk numerik**

Karena semua kolom kategorikal memiliki 2 kategori, kita dapat menggunakan encoder label

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for col in cat_cols:
    df[col] = le.fit_transform(df[col])
```

# MODEL BUILDING

Model building adalah **proses membangun model prediktif atau deskriptif dari data.**

Tujuan utamanya adalah untuk menghasilkan model yang dapat digunakan untuk memprediksi atau menjelaskan fenomena yang diamati berdasarkan data yang ada.

```
[ ]  ind_col = [col for col in df.columns if col != 'class']
     dep_col = 'class'

     X = df[ind_col]
     y = df[dep_col]
```

```
[ ]  # Memisahkan data untuk training dan test set

     from sklearn.model_selection import train_test_split

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 0)
```

# MODEL BUILDING

**Kami menggunakan 10 model sebagai berikut:**

1. KNN
2. Decision Tree Classifier
3. Random Forest Classifier
4. Ada Boost Classifier
5. Gradient Boosting Classifier
6. Stochastic Gradient Boosting (SGB)
7. XgBoost
8. Cat Boost Classifier
9. Extra Trees Classifier
10. LGBM Classifier

# MODEL BUILDING

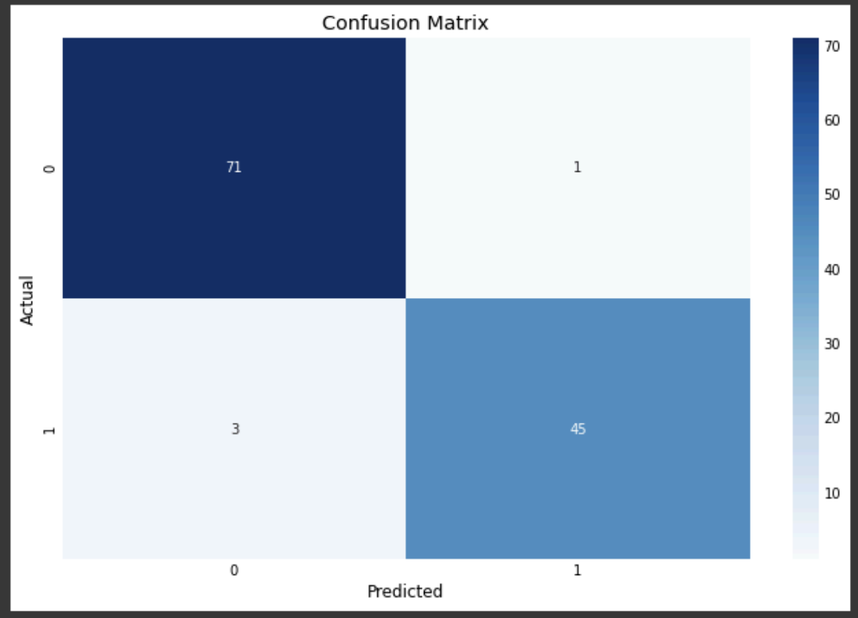
## KNN (K Nearest Neighbor)

```python
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming X_train, X_test, y_train, y_test are already defined

# Instantiate the KNeighborsClassifier
knn = KNeighborsClassifier()

# Fit the model on the training data
knn.fit(X_train, y_train)

# Predict the labels on the test set
y_pred_train = knn.predict(X_train)
y_pred_test = knn.predict(X_test)

# Calculate accuracy score for both training and test sets
knn_train_acc = accuracy_score(y_train, y_pred_train)
knn_test_acc = accuracy_score(y_test, y_pred_test)

# Print training and test accuracy
print(f"Training Accuracy of KNN is {knn_train_acc}")
print(f"Test Accuracy of KNN is {knn_test_acc} \n")

# Generate and print the confusion matrix
knn_cm = confusion_matrix(y_test, y_pred_test)
print(f"Confusion Matrix: \n{knn_cm}\n")

# Generate and print the classification report
class_report = classification_report(y_test, y_pred_test)
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

# MODEL BUILDING

**KNN (K Nearest Neighbor)**

```
Training Accuracy of KNN is 0.7821428571428571
Test Accuracy of KNN is 0.65

Confusion Matrix:
[[44 28]
 [14 34]]

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.61      0.68        72
           1       0.55      0.71      0.62        48

    accuracy                           0.65       120
   macro avg       0.65      0.66      0.65       120
weighted avg       0.67      0.65      0.65       120
```



# MODEL BUILDING

## Decision Tree Classifier

### Men-tuning hyperparameter untuk decision tree

```
# hyper parameter tuning of decision tree

from sklearn.model_selection import GridSearchCV
grid_param = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'splitter' : ['best', 'random'],
    'min_samples_leaf' : [1, 2, 3, 5, 7],
    'min_samples_split' : [1, 2, 3, 5, 7],
    'max_features' : ['auto', 'sqrt', 'log2']
}

grid_search_dtc = GridSearchCV(dtc, grid_param, cv = 5, n_jobs = -1, verbose = 1)
grid_search_dtc.fit(X_train, y_train)
```

# MODEL BUILDING

## Decision Tree Classifier

### Men-tuning hyperparameter untuk decision tree

```
Fitting 5 folds for each of 1200 candidates, totalling 6000 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 280 tasks      | elapsed:    1.1s
[Parallel(n_jobs=-1)]: Done 2680 tasks      | elapsed:    8.9s
[Parallel(n_jobs=-1)]: Done 5990 tasks      | elapsed:   19.2s
[Parallel(n_jobs=-1)]: Done 6000 out of 6000 | elapsed:   19.3s finished
GridSearchCV(cv=5,
             estimator=DecisionTreeClassifier(max_depth=10, max_features='auto',
                                              min_samples_split=3),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 5, 7, 10],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_leaf': [1, 2, 3, 5, 7],
                         'min_samples_split': [1, 2, 3, 5, 7],
                         'splitter': ['best', 'random']},
             verbose=1)
```

# MODEL BUILDING

## Decision Tree Classifier

```
# best parameters dan best score

print(grid_search_dtc.best_params_)
print(grid_search_dtc.best_score_)
```

```
{'criterion': 'entropy', 'max_depth': 7, 'max_features': 'log2', 'min_samples_leaf': 2, 'min_samples_split': 7, 'splitter': 'best'}
0.9821428571428571
```

# MODEL BUILDING

## Decision Tree Classifier

```
# best estimator

dtc = grid_search_dtc.best_estimator_

# accuracy score, confusion matrix and classification report dari decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))

# Predict the labels on the test set
y_pred_train = dtc.predict(X_train)
y_pred_test = dtc.predict(X_test)

# Calculate accuracy score for both training and test sets
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print training and test accuracy
print(f"Training Accuracy of Decision Tree Classifier is {train_accuracy}")
print(f"Test Accuracy of Decision Tree Classifier is {test_accuracy} \n")

# Generate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_test)
print(f"Confusion Matrix: \n{conf_matrix}\n")

# Generate and print the classification report
class_report = classification_report(y_test, y_pred_test)
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=dtc.classes_, yticklabels=dtc.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Decision Tree Classifier')
plt.show()
```

# MODEL BUILDING

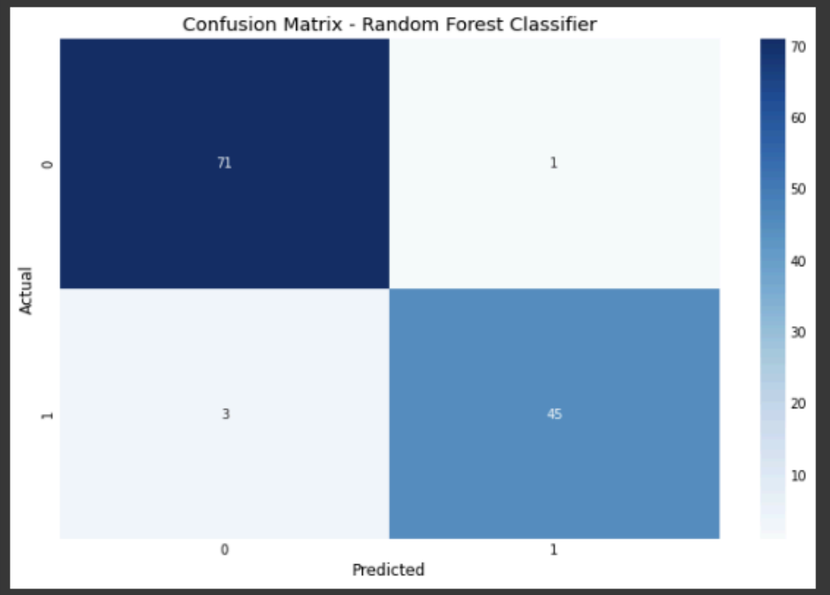
## Random Forest Classifier

```
# best estimator

dtc = grid_search_dtc.best_estimator_

# accuracy score, confusion matrix and classification report dari decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))

# Predict the labels on the test set
y_pred_train = dtc.predict(X_train)
y_pred_test = dtc.predict(X_test)

# Calculate accuracy score for both training and test sets
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print training and test accuracy
print(f"Training Accuracy of Decision Tree Classifier is {train_accuracy}")
print(f"Test Accuracy of Decision Tree Classifier is {test_accuracy} \n")

# Generate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_test)
print(f"Confusion Matrix: \n{conf_matrix}\n")

# Generate and print the classification report
class_report = classification_report(y_test, y_pred_test)
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=dtc.classes_, yticklabels=dtc.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Decision Tree Classifier')
plt.show()
```

# MODEL BUILDING

## Random Forest Classifier

```
Training Accuracy of Random Forest Classifier is 0.9964285714285714
Test Accuracy of Random Forest Classifier is 0.975

Confusion Matrix:
[[72  0]
 [ 3 45]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120
```



# MODEL BUILDING

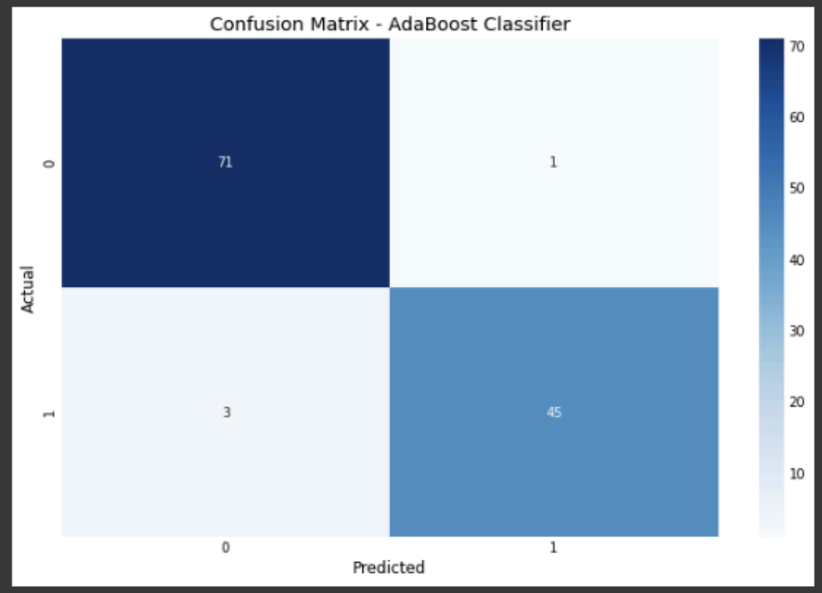
## Ada Boost Classifier

```python
# Import necessary libraries
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Instantiate the AdaBoostClassifier with the Decision Tree classifier (dtc) as the base estimator
ada = AdaBoostClassifier(base_estimator=dtc)

# Fit the model on the training data
ada.fit(X_train, y_train)

# Predict the labels on the test set
y_pred_train = ada.predict(X_train)
y_pred_test = ada.predict(X_test)

# Calculate accuracy score for both training and test sets
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print training and test accuracy
print(f"Training Accuracy of Ada Boost Classifier is {train_accuracy}")
print(f"Test Accuracy of Ada Boost Classifier is {test_accuracy} \n")

# Generate and print the confusion matrix
ada_cm = confusion_matrix(y_test, y_pred_test)
print(f"Confusion Matrix: \n{ada_cm}\n")

# Generate and print the classification report
class_report = classification_report(y_test, y_pred_test)
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=ada.classes_, yticklabels=ada.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - AdaBoost Classifier')
plt.show()
```

# MODEL BUILDING

## Ada Boost Classifier

```
Training Accuracy of Ada Boost Classifier is 1.0
Test Accuracy of Ada Boost Classifier is 0.975

Confusion Matrix:
[[72  0]
 [ 3 45]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120
```



# MODEL BUILDING

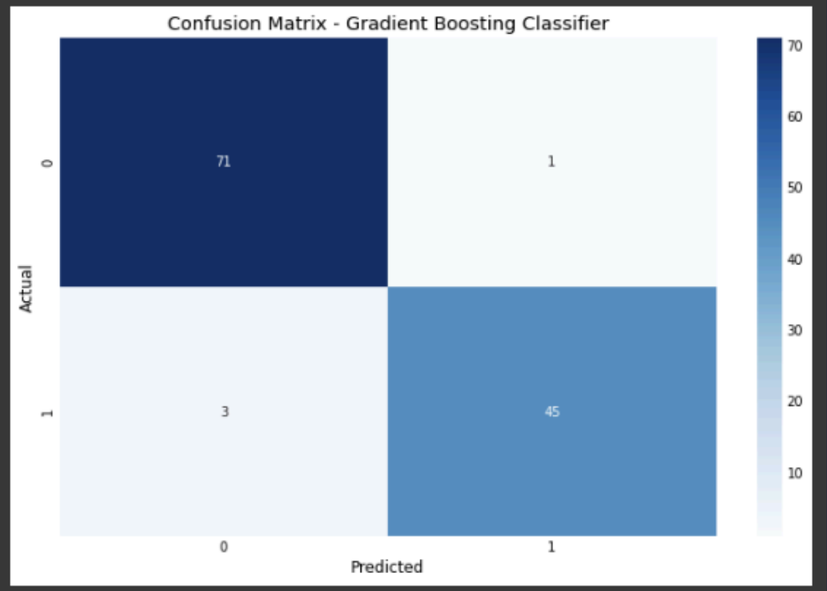
## Gradient Boosting Classifier

```python
# Import necessary libraries
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Instantiate the GradientBoostingClassifier
gb = GradientBoostingClassifier()

# Fit the model on the training data
gb.fit(X_train, y_train)

# Predict the labels on the test set
y_pred_train = gb.predict(X_train)
y_pred_test = gb.predict(X_test)

# Calculate accuracy score for both training and test sets
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print training and test accuracy
print(f"Training Accuracy of Gradient Boosting Classifier is {train_accuracy}")
print(f"Test Accuracy of Gradient Boosting Classifier is {test_accuracy} \n")

# Generate and print the confusion matrix
gb_cm = confusion_matrix(y_test, y_pred_test)
print(f"Confusion Matrix: \n{gb_cm}\n")

# Generate and print the classification report
class_report = classification_report(y_test, y_pred_test)
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=gb.classes_, yticklabels=gb.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Gradient Boosting Classifier')
plt.show()
```

# MODEL BUILDING

## Gradient Boosting Classifier

```
Training Accuracy of Gradient Boosting Classifier is 1.0
Test Accuracy of Gradient Boosting Classifier is 0.9833333333333333

Confusion Matrix:
[[72  0]
 [ 2 46]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        72
           1       1.00      0.96      0.98        48

    accuracy                           0.98       120
   macro avg       0.99      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120
```



# MODEL BUILDING

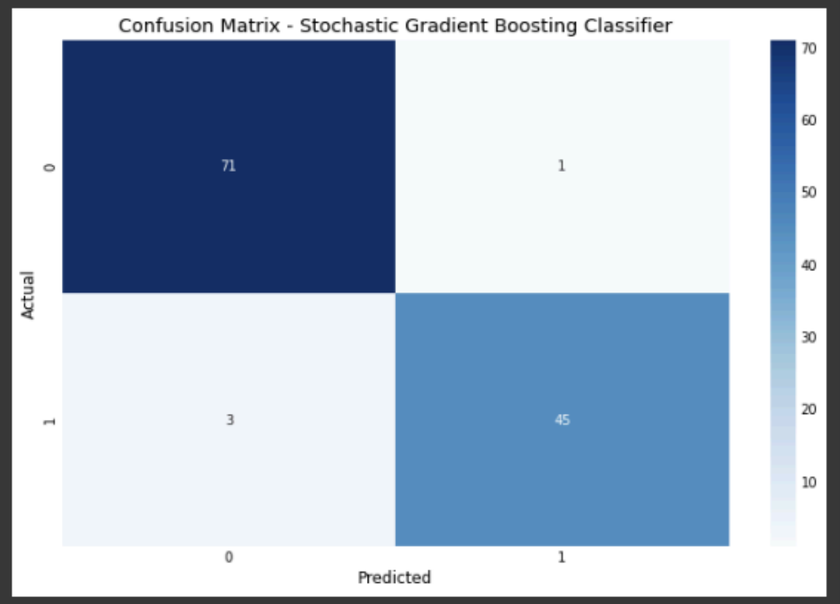
## Stochastic Gradient Boosting (SGB)

```python
# Import necessary libraries
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Instantiate the Stochastic GradientBoostingClassifier
sgb = GradientBoostingClassifier(max_depth=4, subsample=0.90, max_features=0.75, n_estimators=200)

# Fit the model on the training data
sgb.fit(X_train, y_train)

# Predict the labels on the test set
y_pred_train = sgb.predict(X_train)
y_pred_test = sgb.predict(X_test)

# Calculate accuracy score for both training and test sets
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print training and test accuracy
print(f"Training Accuracy of Stochastic Gradient Boosting is {train_accuracy}")
print(f"Test Accuracy of Stochastic Gradient Boosting is {test_accuracy} \n")

# Generate and print the confusion matrix
sgb_cm = confusion_matrix(y_test, y_pred_test)
print(f"Confusion Matrix: \n{sgb_cm}\n")

# Generate and print the classification report
class_report = classification_report(y_test, y_pred_test)
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=sgb.classes_, yticklabels=sgb.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Stochastic Gradient Boosting Classifier')
plt.show()
```

# MODEL BUILDING

**Stochastic Gradient Boosting (SGB)**

```
Training Accuracy of Stochastic Gradient Boosting is 1.0
Test Accuracy of Stochastic Gradient Boosting is 0.9833333333333333

Confusion Matrix:
[[72  0]
 [ 2 46]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        72
           1       1.00      0.96      0.98        48

    accuracy                           0.98       120
   macro avg       0.99      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120
```



# MODEL BUILDING

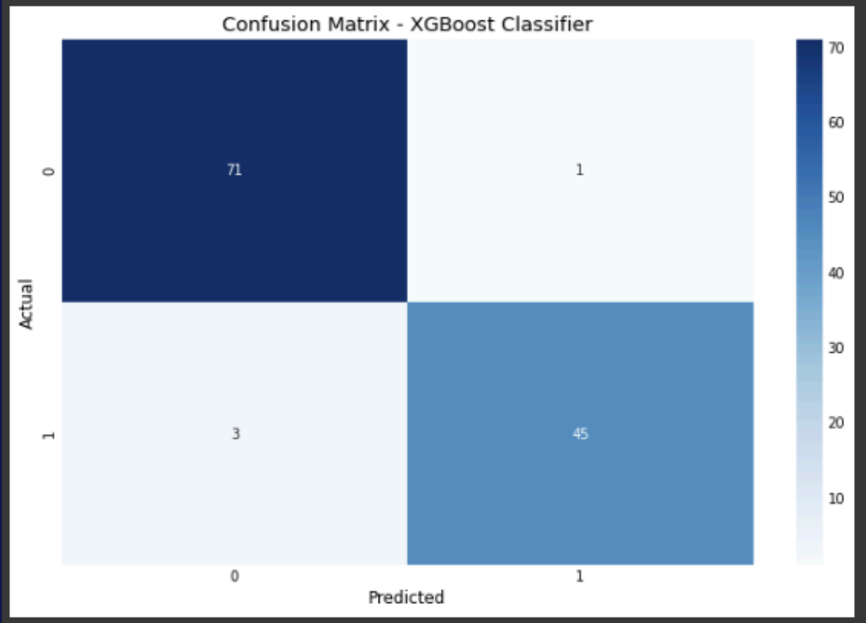
## XgBoost

```python
# Import necessary libraries
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Instantiate the XGBClassifier
xgb = XGBClassifier(objective='binary:logistic', learning_rate=0.5, max_depth=5, n_estimators=150)

# Fit the model on the training data
xgb.fit(X_train, y_train)

# Predict the labels on the test set
y_pred_train = xgb.predict(X_train)
y_pred_test = xgb.predict(X_test)

# Calculate accuracy score for both training and test sets
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print training and test accuracy
print(f"Training Accuracy of XGBoost is {train_accuracy}")
print(f"Test Accuracy of XGBoost is {test_accuracy} \n")

# Generate and print the confusion matrix
xgb_cm = confusion_matrix(y_test, y_pred_test)
print(f"Confusion Matrix: \n{sgb_cm}\n")

# Generate and print the classification report
class_report = classification_report(y_test, y_pred_test)
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=xgb.classes_, yticklabels=xgb.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - XGBoost Classifier')
plt.show()
```

# MODEL BUILDING

**XgBoost**

```
Training Accuracy of XGBoost is 1.0
Test Accuracy of XGBoost is 0.9833333333333333

Confusion Matrix:
[[72  0]
 [ 2 46]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        72
           1       1.00      0.96      0.98        48

    accuracy                           0.98       120
   macro avg       0.99      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120
```



# MODEL BUILDING

**Cat Boost Classifier**

```
from catboost import CatBoostClassifier

cat = CatBoostClassifier(iterations=10)
cat.fit(X_train, y_train)
```

```
Learning rate set to 0.408198
0:      learn: 0.3236955        total: 3.08ms   remaining: 27.7ms
1:      learn: 0.1870546        total: 5.23ms   remaining: 20.9ms
2:      learn: 0.1158561        total: 7.67ms   remaining: 17.9ms
3:      learn: 0.0791225        total: 9.87ms   remaining: 14.8ms
4:      learn: 0.0608841        total: 11.9ms   remaining: 11.9ms
5:      learn: 0.0502265        total: 14.1ms   remaining: 9.39ms
6:      learn: 0.0341717        total: 16.5ms   remaining: 7.06ms
7:      learn: 0.0298214        total: 18.7ms   remaining: 4.67ms
8:      learn: 0.0263479        total: 20.9ms   remaining: 2.32ms
9:      learn: 0.0223712        total: 23ms     remaining: 0us
<catboost.core.CatBoostClassifier at 0x7946790bfed0>
```

# MODEL BUILDING

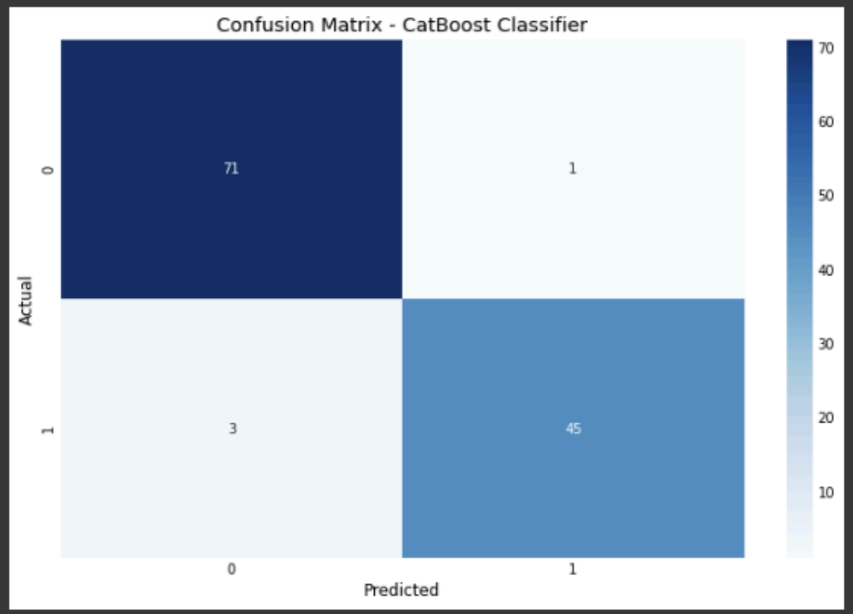
## Cat Boost Classifier

```python
# Import necessary libraries
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming `cat` is the trained CatBoost classifier

# Calculate accuracy score for the test set
cat_acc = accuracy_score(y_test, cat.predict(X_test))

# Print training and test accuracy
print(f"Training Accuracy of Cat Boost Classifier is {accuracy_score(y_train, cat.predict(X_train))}")
print(f"Test Accuracy of Cat Boost Classifier is {cat_acc} \n")

# Generate and print the confusion matrix
cat_cm = confusion_matrix(y_test, cat.predict(X_test))
print(f"Confusion Matrix: \n{cat_cm}\n")

# Generate and print the classification report
class_report = classification_report(y_test, cat.predict(X_test))
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=cat.classes_, yticklabels=cat.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - CatBoost Classifier')
plt.show()
```

# MODEL BUILDING

## Cat Boost Classifier

```
Training Accuracy of Cat Boost Classifier is 1.0
Test Accuracy of Cat Boost Classifier is 0.975

Confusion Matrix:
[[72  0]
 [ 3 45]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120
```



# MODEL BUILDING

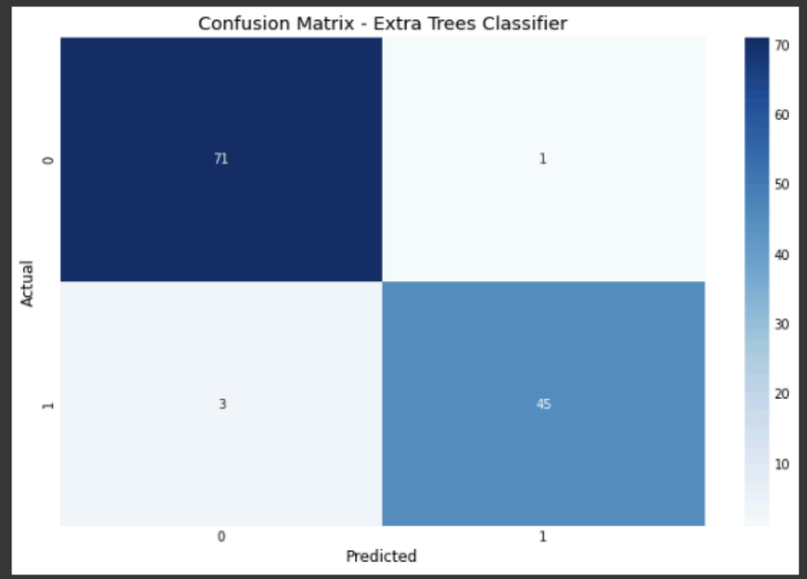
## Extra Trees Classifier

```python
# Import necessary libraries
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Instantiate the ExtraTreesClassifier
etc = ExtraTreesClassifier()

# Fit the model on the training data
etc.fit(X_train, y_train)

# Calculate accuracy score for the test set
etc_acc = accuracy_score(y_test, etc.predict(X_test))

# Print training and test accuracy
print(f"Training Accuracy of Extra Trees Classifier is {accuracy_score(y_train, etc.predict(X_train))}")
print(f"Test Accuracy of Extra Trees Classifier is {etc_acc} \n")

# Generate and print the confusion matrix
etc_cm = confusion_matrix(y_test, etc.predict(X_test))
print(f"Confusion Matrix: \n{etc_cm}\n")

# Generate and print the classification report
class_report = classification_report(y_test, etc.predict(X_test))
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=etc.classes_, yticklabels=etc.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Extra Trees Classifier')
plt.show()
```

# MODEL BUILDING

## Extra Trees Classifier

```
Training Accuracy of Extra Trees Classifier is 1.0
Test Accuracy of Extra Trees Classifier is 0.975

Confusion Matrix:
[[72  0]
 [ 3 45]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120
```



# MODEL BUILDING

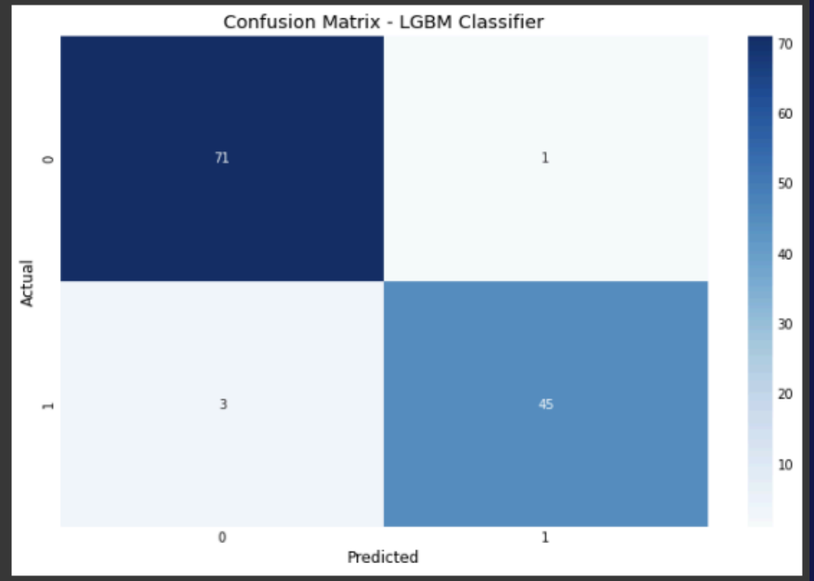
## LGBM Classifier

```python
# Import necessary libraries
from lightgbm import LGBMClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Instantiate the LGBMClassifier
lgbm = LGBMClassifier(learning_rate=1)

# Fit the model on the training data
lgbm.fit(X_train, y_train)

# Calculate accuracy score for the test set
lgbm_acc = accuracy_score(y_test, lgbm.predict(X_test))

# Print training and test accuracy
print(f"Training Accuracy of LGBM Classifier is {accuracy_score(y_train, lgbm.predict(X_train))}")
print(f"Test Accuracy of LGBM Classifier is {lgbm_acc} \n")

# Generate and print the confusion matrix
lgbm_cm = confusion_matrix(y_test, lgbm.predict(X_test))
print(f"Confusion Matrix: \n{lgbm_cm}\n")

# Generate and print the classification report
class_report = classification_report(y_test, lgbm.predict(X_test))
print(f"Classification Report: \n{class_report}")

# Visualize the confusion matrix using seaborn
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=lgbm.classes_, yticklabels=lgbm.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - LGBM Classifier')
plt.show()
```

# MODEL BUILDING

## LGBM Classifier

```
Training Accuracy of LGBM Classifier is 1.0
Test Accuracy of LGBM Classifier is 0.9833333333333333

Confusion Matrix:
[[72  0]
 [ 2 46]]

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        72
           1       1.00      0.96      0.98        48

    accuracy                           0.98       120
   macro avg       0.99      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120
```



# MODEL COMPARISON

Proses **membandingkan kinerja beberapa model pembelajaran mesin untuk menentukan model mana yang paling sesuai untuk tugas tertentu.**

Tujuan dari model comparison adalah untuk memilih model yang memberikan prediksi atau estimasi terbaik berdasarkan data yang tersedia.

```
[ ]  import pandas as pd

     models = pd.DataFrame({
         'Model': ['KNN', 'Decision Tree Classifier', 'Random Forest Classifier', 'Ada Boost Classifier',
                   'Gradient Boosting Classifier', 'Stochastic Gradient Boosting', 'XGBoost', 'Cat Boost',
                   'Extra Trees Classifier', 'LGBM Classifier'],
         'Score': [knn_test_acc, dtc_acc, rd_clf_acc, ada_acc, gb_acc, sgb_acc, xgb_acc, cat_acc, etc_acc, lgbm_acc]
     })

     sorted_models = models.sort_values(by='Score', ascending=False)
     print(sorted_models)
```

# MODEL COMPARISON

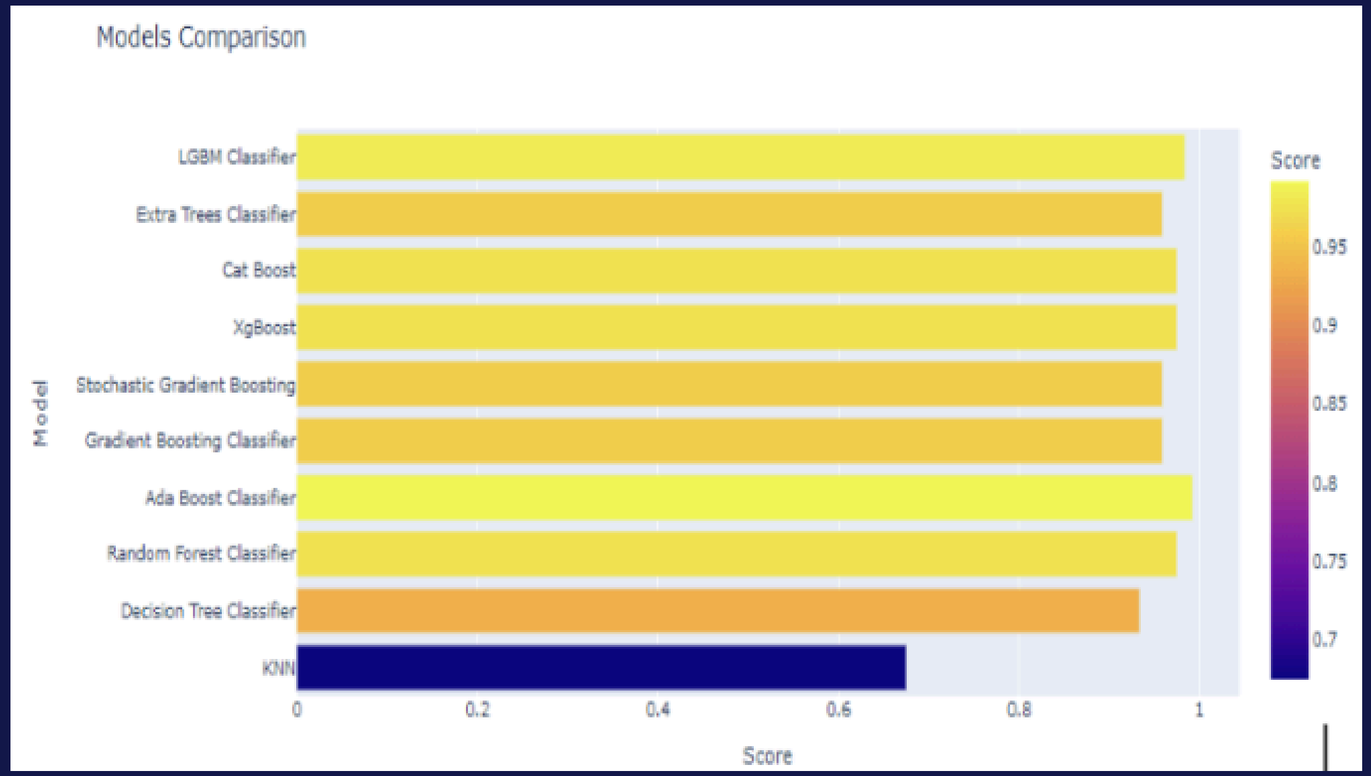
**Hasil dari code pada slide sebelumnya**

| | Model | Score |
|---|---|---|
| 3 | Ada Boost Classifier | 0.991667 |
| 6 | XGBoost | 0.983333 |
| 9 | LGBM Classifier | 0.983333 |
| 2 | Random Forest Classifier | 0.975000 |
| 4 | Gradient Boosting Classifier | 0.975000 |
| 5 | Stochastic Gradient Boosting | 0.975000 |
| 7 | Cat Boost | 0.975000 |
| 8 | Extra Trees Classifier | 0.975000 |
| 1 | Decision Tree Classifier | 0.966667 |
| 0 | KNN | 0.650000 |

# MODEL COMPARISON

**Membandingkan score masing-masing model**

```
[ ]  px.bar(data_frame = models, x = 'Score', y = 'Model', color = 'Score',
            title = 'Models Comparison')
```

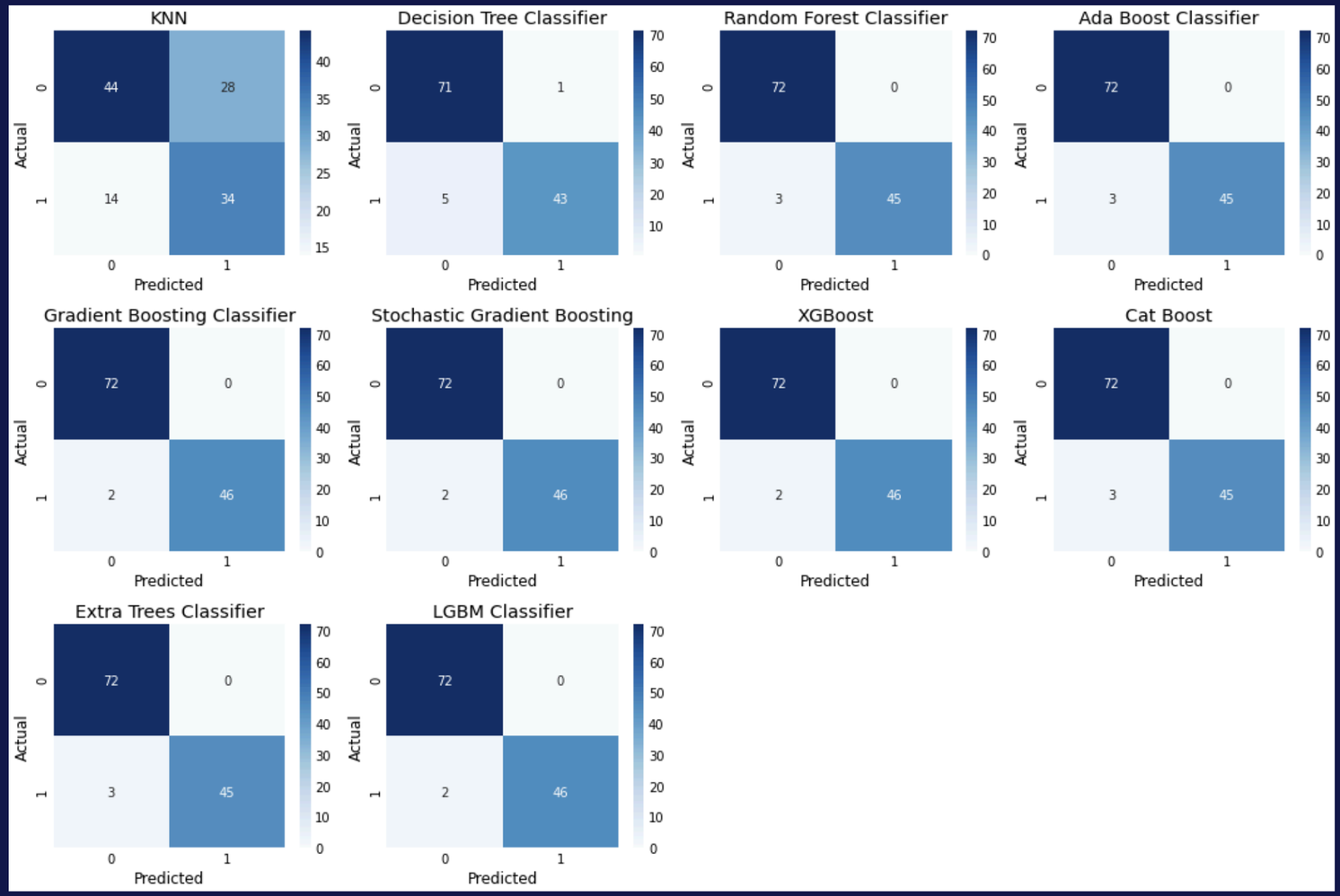


# MODEL COMPARISON

**Membuat Confusion Matrix untuk menilai performa algoritma-algoritma yang digunakan**

```
[ ]  import matplotlib.pyplot as plt
     import seaborn as sns
     import numpy as np

     # Assuming you have defined confusion matrices for each model: knn_cm, dtc_cm, rd_clf_cm, etc.

     models = ['KNN', 'Decision Tree Classifier', 'Random Forest Classifier', 'Ada Boost Classifier',
               'Gradient Boosting Classifier', 'Stochastic Gradient Boosting', 'XGBoost', 'Cat Boost',
               'Extra Trees Classifier', 'LGBM Classifier']

     conf_matrices = [knn_cm, dtc_cm, rd_clf_cm, ada_cm, gb_cm, sgb_cm, xgb_cm, cat_cm, etc_cm, lgbm_cm]

     # Plotting the confusion matrices
     plt.figure(figsize=(15, 10))
     for i, (model, cm) in enumerate(zip(models, conf_matrices)):
         plt.subplot(3, 4, i + 1)
         sns.heatmap(cm, annot=True, cmap='Blues', fmt='d', xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
         plt.title(f'{model}')
         plt.xlabel('Predicted')
         plt.ylabel('Actual')
     plt.tight_layout()
     plt.show()
```

# MODEL COMPARISON



# Conclusion

## Tantangan

- Keterbatasan data
- Pengujian dan validasi yang kompleks
- Kredibilitas yang harus dipastikan
- Intrepretasi yang sulit

## Peluang

- Pendeteksian Dini
- Meningkatkan kualitas hidup
- Pengurangan Beban Kerja Medis
- Penelitian lanjutan

# References

## Paper Reference

https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-022-01951-1

## Kaggle Dataset Refernece

https://www.kaggle.com/datasets/mansoordaku/ckdisease

# Thank you for your attention