UJIAN TENGAH SEMESTER

MACHINE LEARNING

Oleh:
FAJAR ALFIANTINO – 202310072
MICHAEL MERVIN RUSWAN – 202310016
YUDHISTIRA KUSUMA – 202310067

KELAS:

7 TI-20-PA

PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS INFORMATIKA DAN PARIWISATA
INSTITUT BISNIS DAN INFORMATIKA KESATUAN
BOGOR

2023

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

1. ATRIBUT [10 Poin]

1.1 Atribut Prediktor

Tabel 1.1 Tabel Atribut Prediktor

Nama	**Jenis Atribut**	**Keterangan**
battery_power	Numerik	Skala Rasio
blue	Biner	
clock_speed	Numerik	Skala Rasio
dual_sim	Biner	
fc	Numerik	Skala Rasio
four_g	Biner	
int_memory	Numerik	Skala Rasio
m_dep	Numerik	Skala Rasio
mobile_wt	Numerik	Skala Rasio
n_cores	Numerik	Skala Rasio
pc	Numerik	Skala Rasio
px_height	Numerik	Skala Rasio
px_width	Numerik	Skala Rasio
ram	Numerik	Skala Rasio
sc_h	Numerik	Skala Rasio
sc_w	Numerik	Skala Rasio
talk_time	Numerik	Skala Rasio
three_g	Biner	
touch_screen	Biner	
wifi	Biner	

1.2 Atribut Label

Tabel 1.2 Tabel Atribut Label

Nama	**Jenis Atribut**
price_range	Ordinal

2. STATISTIK DESKRIPTIF

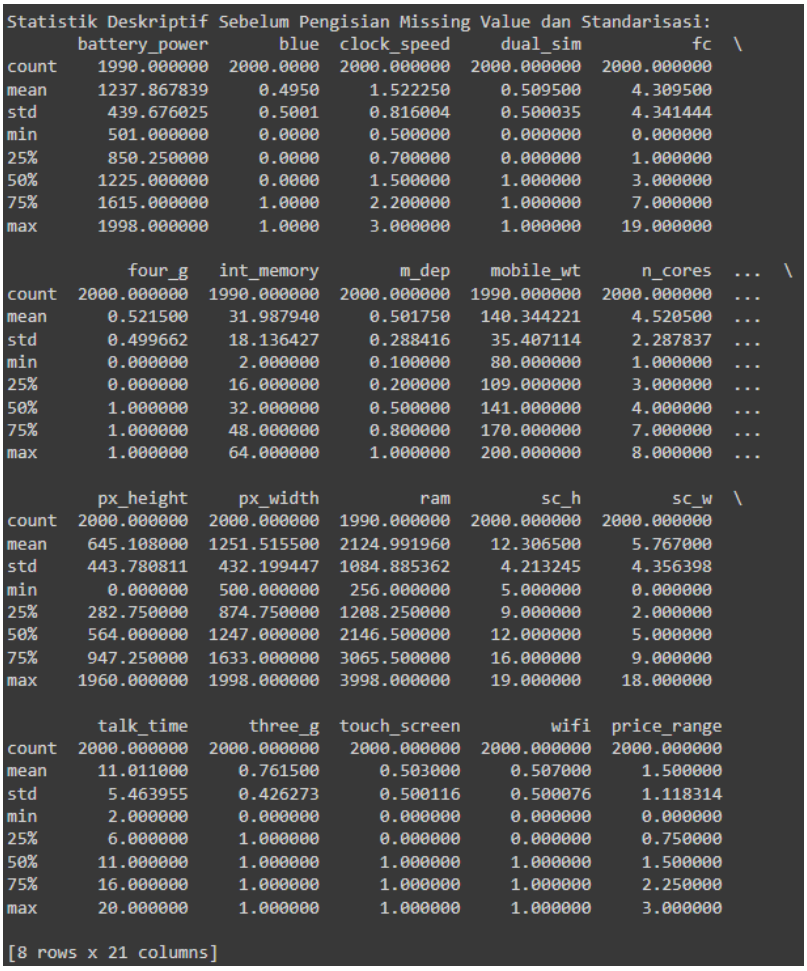
2.1 Sebelum Praproses Data

1. Statistik deskriptif merupakan cabang dari statistik untuk mendeskripsikan dan merangkum data. Contoh hal umum yang biasa dilakukan di dalam tipe statistik ini seperti pembuatan graph, dan menghitung berbagai macam pengukuran data seperti Mean.
2. Sebelum dilakukan praproses data, pada dataset masih terdapat *missing values.*
3. Untuk menampilkan statistik deskriptif dapat menjalankan perintah seperti kode berikut pada Google Collab.

Tabel 2.1 Kode Menampilkan Statistik Deskriptif Sebelum Praproses Data

```
# Tampilkan statistik deskriptif sebelum pengisian missing value dan standarisasi
# Statistik deskriptif => cabang dari statistik untuk mendeskripsikan dan merangkum data.
# 2 jenis statistik deskriptif => Measures of Central Tendency (mean, median, modus), Measures of Spread (std, kuartil)



deskripsi_awal = dataset.describe()
print("Statistik Deskriptif Sebelum Pengisian Missing Value dan Standarisasi:")
print(deskripsi_awal)
```

- Hasil dari kode di atas menghasilkan statistik deskriptif seperti berikut.



```
Statistik Deskriptif Sebelum Pengisian Missing Value dan Standarisasi:
       battery_power       blue  clock_speed     dual_sim           fc  \
count    1990.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean     1237.867839     0.4950     1.522250     0.509500     4.309500
std       439.676025     0.5001     0.816004     0.500035     4.341444
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       850.250000     0.0000     0.700000     0.000000     1.000000
50%      1225.000000     0.0000     1.500000     1.000000     3.000000
75%      1615.000000     1.0000     2.200000     1.000000     7.000000
max      1998.000000     1.0000     3.000000     1.000000    19.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores  ...  \
count  2000.000000  1990.000000  2000.000000  1990.000000  2000.000000  ...
mean      0.521500    31.987940     0.501750   140.344221     4.520500  ...
std       0.499662    18.136427     0.288416    35.407114     2.287837  ...
min       0.000000     2.000000     0.100000    80.000000     1.000000  ...
25%       0.000000    16.000000     0.200000   109.000000     3.000000  ...
50%       1.000000    32.000000     0.500000   141.000000     4.000000  ...
75%       1.000000    48.000000     0.800000   170.000000     7.000000  ...
max       1.000000    64.000000     1.000000   200.000000     8.000000  ...

         px_height     px_width          ram         sc_h         sc_w  \
count  2000.000000  2000.000000  1990.000000  2000.000000  2000.000000
mean    645.108000  1251.515500  2124.991960    12.306500     5.767000
std     443.780811   432.199447  1084.885362     4.213245     4.356398
min       0.000000   500.000000   256.000000     5.000000     0.000000
25%     282.750000   874.750000  1208.250000     9.000000     2.000000
50%     564.000000  1247.000000  2146.500000    12.000000     5.000000
75%     947.250000  1633.000000  3065.500000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000

         talk_time      three_g  touch_screen         wifi  price_range
count  2000.000000  2000.000000   2000.000000  2000.000000  2000.000000
mean     11.011000     0.761500      0.503000     0.507000     1.500000
std       5.463955     0.426273      0.500116     0.500076     1.118314
min       2.000000     0.000000      0.000000     0.000000     0.000000
25%       6.000000     1.000000      0.000000     0.000000     0.750000
50%      11.000000     1.000000      1.000000     1.000000     1.500000
75%      16.000000     1.000000      1.000000     1.000000     2.250000
max      20.000000     1.000000      1.000000     1.000000     3.000000

[8 rows x 21 columns]
```

Gambar 2.1 Statistik Deskriptif Sebelum Praproses Data

- Pada hasil di atas, terlihat terdapat beberapa informasi yang disampaikan dari 21 kolom, mulai dari jumlah data, rata-rata atau mean, standar deviasi, nilai minimal, nilai maksimal, kuartil pertama, kuartil kedua atau median, serta kuartil ketiga.

2.2 Setelah Praproses Data

1. Praproses data merupakan langkah yang dilakukan sebelum data diproses lebih lanjut.
2. Terdapat tiga tahap pada praproses data yang dilakukan, yaitu memisahkan bagian atribut prediktor dengan label data menjadi dua variabel terpisah,

memastikan tidak adanya missing values pada dataset, serta menyeragamkan nilai dari masing-masing atribut.

3. Untuk langkah memisahkan bagian atribut prediktor dilakukan dengan pertama-tama membuat dua variabel, misalnya variabel X merupakan variabel yang berisi atribut prediktor yaitu atribut selain atribut price_range sehingga kita menghapus kolom price range pada variabel x.
4. Untuk variabel kedua yaitu variabel Y ditetapkan kolom price_range yang digunakan sebagai atribut label.
5. Untuk contoh kode dapat dilihat seperti berikut.

Tabel 2.2 Kode Memisahkan Atribut Prediktor dengan Atribut Label

```
# Pisahkan atribut prediktor dan atribut label
X = dataset.drop("price_range", axis=1)  # Atribut prediktor
y = dataset["price_range"]  # Atribut label
```

6. Dari dataset yang kami dapatkan terdapat beberapa missing values. Untuk mengisi missing values dilakukan dengan menggunakan SimpleImputer dan nilai median sebagai strategi pengisian nilai.
7. Hal yang pertama dilakukan adalah inisialisasi SimpleImputer dan menetapkan strategi median.
8. Kedua, melakukan imputasi nilai mneggunakan SimpleImputer yang telah diinisialisasi sebelumnya.
9. Untuk menjalankan perintah kode dapat dilihat seperti berikut.

Tabel 2.3 Kode Menggunakan SimpleImputer untuk Menangani *Missing Values*

```
# Inisialisasi SimpleImputer untuk mengisi nilai-nilai yang hilang dengan median
imputer = SimpleImputer(strategy="median")

# Imputasi missing value dengan median
X_imputed = imputer.fit_transform(X)
```

10. Setelah missing values terisi, maka langkah selanjutnya adalah menyeragamkan nilai dari masing-masing atribut. Terdapat dua opsi metode yang dapat dilakukan, yaitu menggunakan MinMaxScaler atau StandardScaler.
11. Baik untuk MinMaxScaler dan StandardScaler, langkah yang perlu dilakukan serupa, dimulai dari inisialasi metode, penerapan metode pada atribut prediktor yang sudah diimputasi, mengonversi hasil transformasi ke Pandas DataFrame, serta menggabungkan atribut prediktor yang sudah diimputasi dan distandarisasi dengan atribut label
12. Untuk kode menggunakan MinMaxScaler dapat dilihat seperti berikut.

Tabel 2.4 Kode Menggunakan MinMaxScaler untuk Mengubah Nilai Atribut Prediktor

```
# Inisialisasi Min-Max Scaler
scaler = MinMaxScaler()

# Lakukan Min-Max Scaling pada atribut prediktor yang sudah diimputasi
X_scaled = scaler.fit_transform(X_imputed)

# Konversi hasil transformasi kembali ke Pandas DataFrame (opsional)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Gabungkan atribut prediktor yang sudah diimputasi dan distandarisasi dengan atribut label
data_scaled = pd.concat([X_scaled_df, y], axis=1)
```

13. Untuk menampilkan hasil statistik deskriptif setelah dilakukan praproses data menggunakan MinMaxScaler dapat menjalankan kode berikut.

Tabel 2.5 Kode Menampilkan Statistik Deskriptif MinMaxScaler

```
# Tampilkan statistik deskriptif setelah pengisian dan standarisasi
deskripsi_setelah = data_scaled.describe()
print("Statistik Deskriptif Setelah Pengisian dan Standarisasi (Menggunakan MinMaxScaler):")
print(deskripsi_setelah)
```

14. Untuk statistik deskriptif setelah dilakukannya praproses data mengguanakan MinMaxScaler dapat dilihat seperti di bawah ini.

```
Statistik Deskriptif Setelah Pengisian dan Standarisasi (Menggunakan MinMaxScaler):
       battery_power       blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean        0.492187     0.4950     0.408900     0.509500     0.226816
std         0.292970     0.5001     0.326402     0.500035     0.228497
min         0.000000     0.0000     0.000000     0.000000     0.000000
25%         0.234302     0.0000     0.080000     0.000000     0.052632
50%         0.483634     0.0000     0.400000     1.000000     0.157895
75%         0.743487     1.0000     0.680000     1.000000     0.368421
max         1.000000     1.0000     1.000000     1.000000     1.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean      0.521500     0.483677     0.446389     0.502896     0.502929
std       0.499662     0.291790     0.320462     0.294321     0.326834
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.225806     0.111111     0.241667     0.285714
50%       1.000000     0.483871     0.444444     0.508333     0.428571
75%       1.000000     0.741935     0.777778     0.750000     0.857143
max       1.000000     1.000000     1.000000     1.000000     1.000000

                pc    px_height     px_width          ram         sc_h  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean      0.495825     0.329137     0.501679     0.499492     0.521893
std       0.303216     0.226419     0.288518     0.289195     0.300946
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.250000     0.144260     0.250167     0.255144     0.285714
50%       0.500000     0.287755     0.498665     0.505211     0.500000
75%       0.750000     0.483291     0.756342     0.749332     0.785714
max       1.000000     1.000000     1.000000     1.000000     1.000000

             sc_w    talk_time      three_g  touch_screen         wifi
count  2000.000000  2000.000000  2000.000000   2000.000000  2000.000000
mean      0.320389     0.500611     0.761500      0.503000     0.507000
std       0.242022     0.303553     0.426273      0.500116     0.500076
min       0.000000     0.000000     0.000000      0.000000     0.000000
25%       0.111111     0.222222     1.000000      0.000000     0.000000
50%       0.277778     0.500000     1.000000      1.000000     1.000000
75%       0.500000     0.777778     1.000000      1.000000     1.000000
max       1.000000     1.000000     1.000000      1.000000     1.000000
```

Gambar 2.2 Hasil Statistik Deskriptif Menggunakan MinMaxScaler

15. Sedangkan kode menggunakan StandardScaler dapat dilihat seperti berikut.

Tabel 2.6 Kode Menggunakan StandardScaler untuk Mengubah Nilai Atribut Prediktor

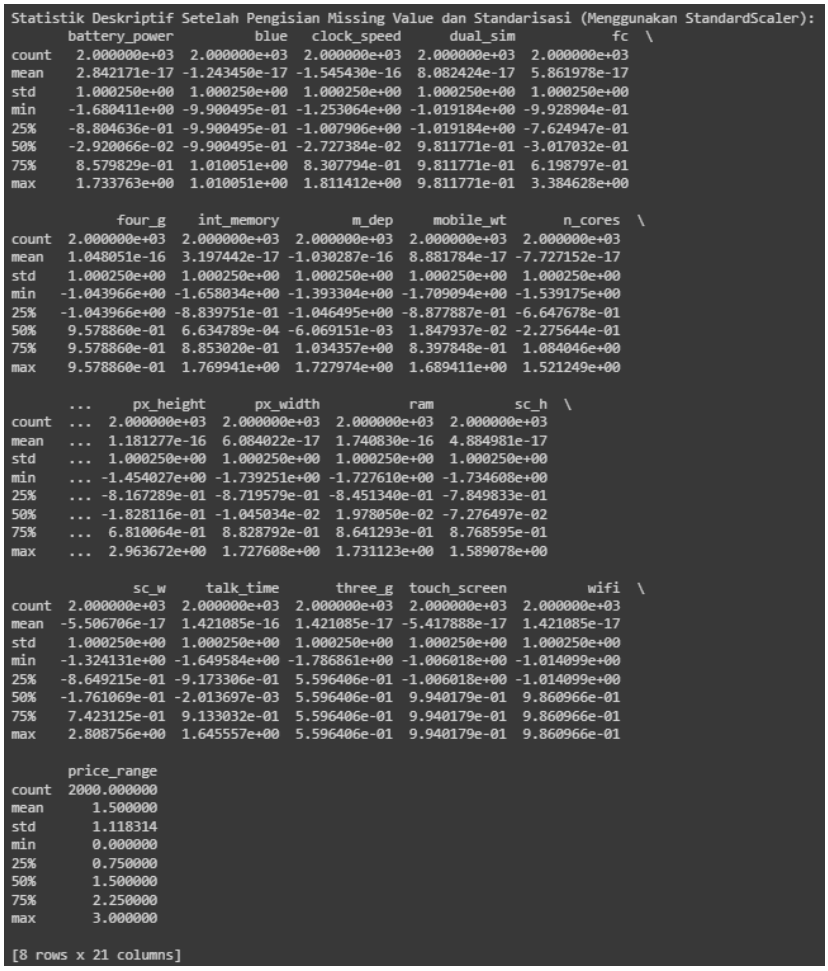
```
# Inisialisasi StandardScaler untuk mengubah nilai atribut prediktor agar nilai dari masing-masing atribut seragam
scaler = StandardScaler()

# Lakukan standarisasi menggunakan StandardScaling Scaling pada atribut prediktor yang sudah diimputasi

X_scaled = scaler.fit_transform(X_imputed)

# Konversi hasil transformasi kembali ke Pandas DataFrame (opsional)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

# Gabungkan atribut prediktor yang sudah diimputasi dan distandarisasi dengan atribut label
data_scaled = pd.concat([X_scaled_df, y], axis=1)
```

16. Untuk menampilkan hasil statistik deskriptif setelah dilakukan praproses data menggunakan StandardScaler dapat menjalankan kode berikut.

Tabel 2.7 Kode Menampilkan Statistik Deskriptif StandardScaler

```
# Tampilkan statistik deskriptif setelah pengisian missing value dan standarisasi
deskripsi_setelah = data_scaled.describe()
print("\nStatistik Deskriptif Setelah Pengisian Missing Value dan Standarisasi (Menggunakan StandardScaler):")
print(deskripsi_setelah)
```

17. Adapun hasil statistik deskriptif setelah dilakukan praproses data menggunakan StandardScaler



```
Statistik Deskriptif Setelah Pengisian Missing Value dan Standarisasi (Menggunakan StandardScaler):
       battery_power          blue   clock_speed      dual_sim            fc  \
count   2.000000e+03  2.000000e+03  2.000000e+03  2.000000e+03  2.000000e+03
mean    2.842171e-17 -1.243450e-17 -1.545430e-16  8.082424e-17  5.861978e-17
std     1.000250e+00  1.000250e+00  1.000250e+00  1.000250e+00  1.000250e+00
min    -1.680411e+00 -9.900495e-01 -1.253064e+00 -1.019184e+00 -9.928904e-01
25%    -8.804636e-01 -9.900495e-01 -1.007906e+00 -1.019184e+00 -7.624947e-01
50%    -2.920066e-02 -9.900495e-01 -2.727384e-02  9.811771e-01 -3.017032e-01
75%     8.579829e-01  1.010051e+00  8.307794e-01  9.811771e-01  6.198797e-01
max     1.733763e+00  1.010051e+00  1.811412e+00  9.811771e-01  3.384628e+00

            four_g    int_memory         m_dep     mobile_wt       n_cores  \
count  2.000000e+03  2.000000e+03  2.000000e+03  2.000000e+03  2.000000e+03
mean   1.048051e-16  3.197442e-17 -1.030287e-16  8.881784e-17 -7.727152e-17
std    1.000250e+00  1.000250e+00  1.000250e+00  1.000250e+00  1.000250e+00
min   -1.043966e+00 -1.658034e+00 -1.393304e+00 -1.709094e+00 -1.539175e+00
25%   -1.043966e+00 -8.839751e-01 -1.046495e+00 -8.877887e-01 -6.647678e-01
50%    9.578860e-01  6.634789e-04 -6.069151e-03  1.847937e-02 -2.275644e-01
75%    9.578860e-01  8.853020e-01  1.034357e+00  8.397848e-01  1.084046e+00
max    9.578860e-01  1.769941e+00  1.727974e+00  1.689411e+00  1.521249e+00

       ...     px_height      px_width           ram          sc_h  \
count  ...  2.000000e+03  2.000000e+03  2.000000e+03  2.000000e+03
mean   ...  1.181277e-16  6.084022e-17  1.740830e-16  4.884981e-17
std    ...  1.000250e+00  1.000250e+00  1.000250e+00  1.000250e+00
min    ... -1.454027e+00 -1.739251e+00 -1.727610e+00 -1.734608e+00
25%    ... -8.167289e-01 -8.719579e-01 -8.451340e-01 -7.849833e-01
50%    ... -1.828116e-01 -1.045034e-02  1.978050e-02 -7.276497e-02
75%    ...  6.810064e-01  8.828792e-01  8.641293e-01  8.768595e-01
max    ...  2.963672e+00  1.727608e+00  1.731123e+00  1.589078e+00

              sc_w     talk_time       three_g  touch_screen          wifi  \
count  2.000000e+03  2.000000e+03  2.000000e+03  2.000000e+03  2.000000e+03
mean  -5.506706e-17  1.421085e-16  1.421085e-17 -5.417888e-17  1.421085e-17
std    1.000250e+00  1.000250e+00  1.000250e+00  1.000250e+00  1.000250e+00
min   -1.324131e+00 -1.649584e+00 -1.786861e+00 -1.006018e+00 -1.014099e+00
25%   -8.649215e-01 -9.173306e-01  5.596406e-01 -1.006018e+00 -1.014099e+00
50%   -1.761069e-01 -2.013697e-03  5.596406e-01  9.940179e-01  9.860966e-01
75%    7.423125e-01  9.133032e-01  5.596406e-01  9.940179e-01  9.860966e-01
max    2.808756e+00  1.645557e+00  5.596406e-01  9.940179e-01  9.860966e-01

       price_range
count  2000.000000
mean      1.500000
std       1.118314
min       0.000000
25%       0.750000
50%       1.500000
75%       2.250000
max       3.000000

[8 rows x 21 columns]
```

Gambar 2.3 Hasil Statistik Deskriptif Menggunakan StandardScaler

18. Dari dua metode di atas terlihat terdapat perbedaan representasi statistik deskriptif. Pada metode MinMaxScaler tidak menggunakan e untuk merepresentasikan nilai, sementara pada StandardScaler menggunakan e untuk merepresentasikan nilai.
19. Dari statistik deskriptif juga dapat terlihat bahwa terdapat perbedaan baik dari hasil sebelum praproses data maupun setelah praproses data. Bahkan, perbedaan metode yang digunakan praproses data juga menghasilkan statistik deskriptif yang berbeda. Misalkan, standar deviasi battery_power bernilai 439.676025 sebelum praproses data, 0.292970 pada MinMaxScaler, dan 1.000250e+00 pada StandardScaler.

3. MODEL KLASIFIKASI

3.1 KNN

Berikut ini merupakan langkah-langkah dalam membangun model klasifikasi menggunakan algoritma KNN:

1. Hal pertama yang harus dilakukan adalah memisahkan atribut. Atribut akan dipisahkan berdasarkan jenisnya, yaitu atribut prediktor dan atribut label.
2. Melakukan inisialisasi SimpleInputer yang bertujuan untuk mengisi nilai-nilai yang hilang dengan menggunakan median.
3. Dataset kemudian akan dibagi menjadi data latih dan data uji dengan masing-masing persentase 80% untuk data latih dan 20% untuk data uji dengan metode *holdout.*
4. Langkah berikutnya adalah melakukan inisialisasi untuk model KNN.
5. Melakukan pelatihan model KNN dengan menggunakan data latih yang sudah disiapkan pada langkah sebelumnya.
6. Melakukan prediksi label pada data uji.
7. Langkah berikutnya adalah menghitung nilai akurasi dari model yang sudah dibuat.
8. Langkah terakhir adalah menampilkan *confusion matrix*.

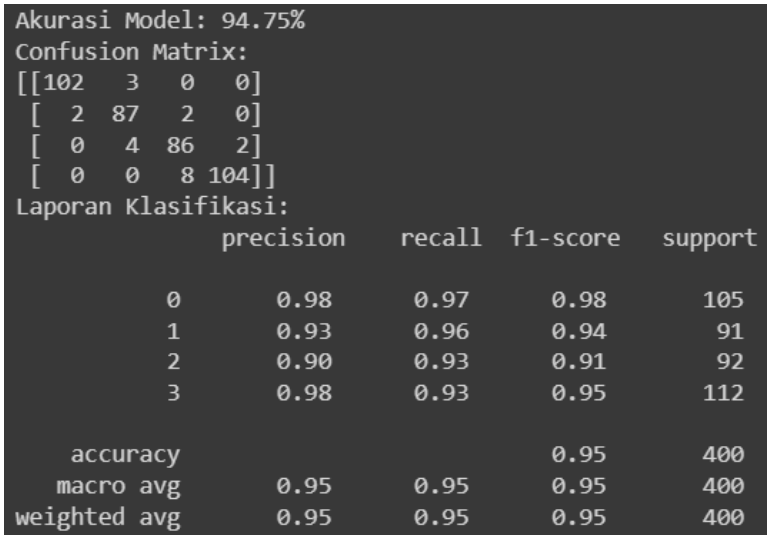
Untuk penjelasan lebih lanjut, berikut ini adalah contoh *code* dari model klasifikasi KNN :

Tabel 3.1 *Code* Model Klasifikasi Menggunakan Algoritma KNN

```
# Pisahkan atribut prediktor dan atribut label
X = dataset.drop("price_range", axis=1)  # Atribut prediktor
y = dataset["price_range"]  # Atribut label


# Inisialisasi SimpleImputer untuk mengisi nilai-nilai yang hilang dengan median
imputer = SimpleImputer(strategy="median")
X = imputer.fit_transform(X)
```

```
# Bagi data menjadi data latih (80%) dan data uji (20%) dengan metode holdout
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Inisialisasi model KNN
knn_model = KNeighborsClassifier(n_neighbors=25)  # Ganti nilai n_neighbors sesuai kebutuhan, 25 terbaik, 94,75% (?)

# Latih model KNN menggunakan data latih
knn_model.fit(X_train, y_train)

# Prediksi label pada data uji
y_pred = knn_model.predict(X_test)

# Hitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi Model: {:.2f}%".format(accuracy * 100))
class_report = classification_report(y_test, y_pred)

# Tampilkan confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
print("Laporan Klasifikasi:\n", class_report)
```

Hasil yang didapat dari model klasifikasi menggunakan algoritma KNN dapat dilihat dari gambar dibawah :



```
Akurasi Model: 94.75%
Confusion Matrix:
[[102   3   0   0]
 [  2  87   2   0]
 [  0   4  86   2]
 [  0   0   8 104]]
Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       105
           1       0.93      0.96      0.94        91
           2       0.90      0.93      0.91        92
           3       0.98      0.93      0.95       112

    accuracy                           0.95       400
   macro avg       0.95      0.95      0.95       400
weighted avg       0.95      0.95      0.95       400
```

Gambar 3.1 Hasil Model Klasifikasi Menggunakan Algoritma KNN

3.2 *Decision Tree*

Berikut ini merupakan langkah-langkah dalam membangun model klasifikasi menggunakan algoritma *Decision Tree*:

1. Hal pertama yang harus dilakukan adalah memisahkan atribut. Atribut akan dipisahkan berdasarkan jenisnya, yaitu atribut prediktor dan atribut label.
2. Melakukan inisialisasi SimpleInputer yang bertujuan untuk mengisi nilai-nilai yang hilang dengan menggunakan median.
3. Dataset kemudian akan dibagi menjadi data latih dan data uji dengan masing-masing persentase 80% untuk data latih dan 20% untuk data uji dengan metode *holdout.*
4. Langkah berikutnya adalah melakukan inisialisasi untuk model *Decision Tree*.
5. Melakukan pelatihan model *Decision Tree* dengan menggunakan data latih yang sudah disiapkan pada langkah sebelumnya.
6. Melakukan prediksi label pada data uji.
7. Langkah berikutnya adalah menghitung nilai akurasi dari model yang sudah dibuat.
8. Langkah terakhir adalah menampilkan *confusion matrix*.

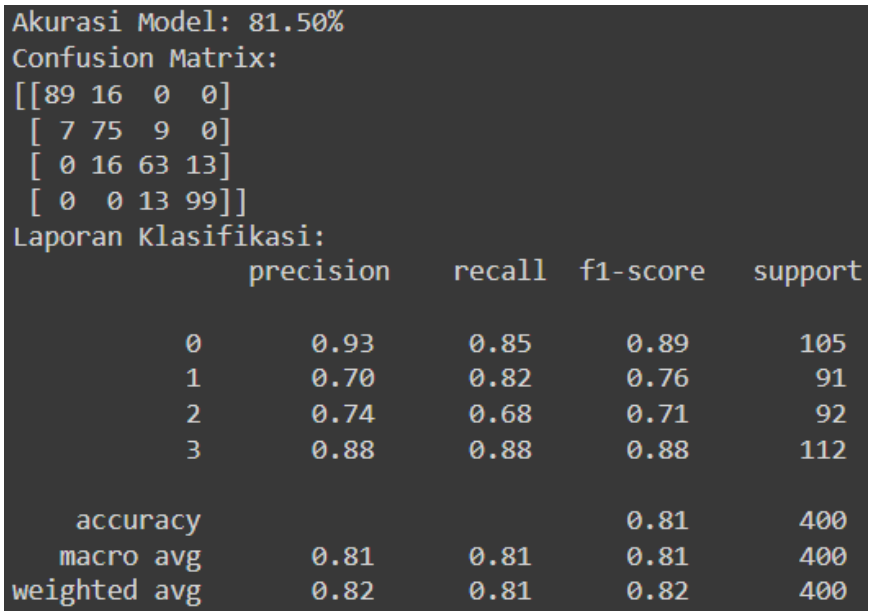
Untuk penjelasan lebih lanjut, berikut ini adalah contoh *code* dari model klasifikasi *Decision Tree*:

Tabel 3.2 *Code* Model Klasifikasi Menggunakan Algoritma *Decision Tree*

```
# Pisahkan atribut prediktor dan atribut label
X = dataset.drop("price_range", axis=1)  # Atribut prediktor
y = dataset["price_range"]  # Atribut label

# Inisialisasi SimpleImputer untuk mengisi nilai-nilai yang hilang dengan median
imputer = SimpleImputer(strategy="median")
X = imputer.fit_transform(X)

# Bagi data menjadi data latih (80%) dan data uji (20%) dengan metode holdout
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Inisialisasi model Decision Tree
decision_tree_model = DecisionTreeClassifier()

# Latih model Decision Tree menggunakan data latih
decision_tree_model.fit(X_train, y_train)

# Prediksi label pada data uji
y_pred = decision_tree_model.predict(X_test)

# Hitung akurasi
accuracy = accuracy_score(y_test, y_pred)
print("Akurasi Model: {:.2f}%".format(accuracy * 100))
class_report = classification_report(y_test, y_pred)

# Tampilkan confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
```

```
print(conf_matrix)
print("Laporan Klasifikasi:\n", class_report)
```

Hasil yang didapat dari model klasifikasi menggunakan algoritma *Decision Tree* dapat dilihat dari gambar dibawah :



Gambar 3.2 Hasil Model Klasifikasi Menggunakan Algoritma *Decision Tree*

3.3 SVM

Berikut ini merupakan langkah-langkah dalam membangun model klasifikasi menggunakan algoritma SVM :

1. Hal pertama yang harus dilakukan adalah memisahkan atribut. Atribut akan dipisahkan berdasarkan jenisnya, yaitu atribut prediktor dan atribut label.
2. Melakukan inisialisasi SimpleInputer yang bertujuan untuk mengisi nilai-nilai yang hilang dengan menggunakan median dan StandardScaler untuk melakukan normalisasi data agar data yang digunakan tidak memiliki penyimpangan yang besar.
3. Melakukan imputasi *missing value* dan standarisasi atribut prediktor.
4. Melakukan inisialisasi model SVM (*Support Vector Machine*) dengan kernel linear.

5. Dataset kemudian akan dibagi menjadi data latih dan data uji dengan masing-masing persentase 80% untuk data latih dan 20% untuk data uji dengan metode *holdout.*
6. Melakukan pelatihan model SVM dengan menggunakan data latih yang sudah disiapkan pada langkah sebelumnya.
7. Melakukan prediksi label pada data uji.
8. Langkah berikutnya adalah menghitung nilai akurasi dari model yang sudah dibuat.
9. Langkah terakhir adalah menampilkan *confusion matrix*.

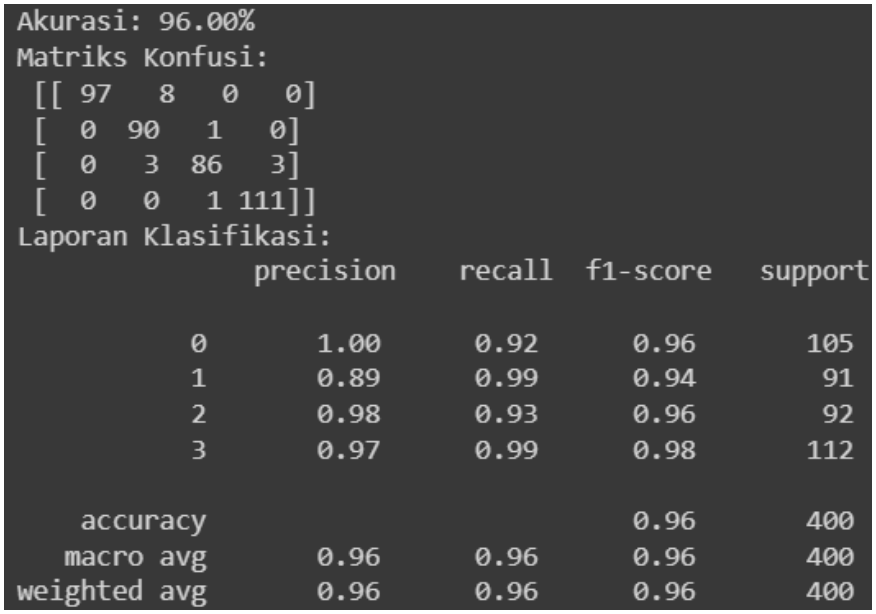
Untuk penjelasan lebih lanjut, berikut ini adalah contoh *code* dari model klasifikasi SVM :

Tabel 3.3 *Code* Model Klasifikasi Menggunakan Algoritma SVM

```
# Pisahkan atribut prediktor dan atribut label
X = dataset.drop("price_range", axis=1)  # Atribut prediktor yang sudah diimputasi dan distandarisasi
y = dataset["price_range"]  # Atribut label

# Inisialisasi SimpleImputer dan StandardScaler
imputer = SimpleImputer(strategy="median")
scaler = StandardScaler()

# Imputasi missing value dan standarisasi atribut prediktor
X_imputed = imputer.fit_transform(X)
X_scaled = scaler.fit_transform(X_imputed)

# Inisialisasi model SVM (Support Vector Machine) dengan kernel linear
svm_model = SVC(kernel='linear')

# Bagi data menjadi data latih (80%) dan data uji (20%) dengan metode holdout
```

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

# Latih model SVM menggunakan data latih
svm_model.fit(X_train, y_train)

# Prediksi label pada data uji
y_pred = svm_model.predict(X_test)

# Evaluasi model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Tampilkan hasil evaluasi
print("Akurasi: {:.2f}%".format(accuracy * 100))
print("Matriks Konfusi:\n", conf_matrix)
print("Laporan Klasifikasi:\n", class_report)
```

Hasil yang didapat dari model klasifikasi menggunakan algoritma SVM dapat dilihat dari gambar dibawah :



```
Akurasi: 96.00%
Matriks Konfusi:
 [[ 97   8   0   0]
 [  0  90   1   0]
 [  0   3  86   3]
 [  0   0   1 111]]
Laporan Klasifikasi:
              precision    recall  f1-score   support

           0       1.00      0.92      0.96       105
           1       0.89      0.99      0.94        91
           2       0.98      0.93      0.96        92
           3       0.97      0.99      0.98       112

    accuracy                           0.96       400
   macro avg       0.96      0.96      0.96       400
weighted avg       0.96      0.96      0.96       400
```

Gambar 3.3 Hasil Model Klasifikasi Menggunakan Algoritma SVM

4. MODEL *CLUSTERING*

4.1 Agglomerative *Clustering*

Berikut cara yang dapat dilakukan untuk mencari silhouette *score* dengan menggunakan Agglomerative *Clustering*:

1. Memisahkan atribut prediktor dengan cara menggunakan fungsi *drop*
2. Melakukan *simple* imputer untuk mengisi nilai-nilai kosong dengan menggunakan median
3. Melakukan inisiasi agglomerative *clustering* dengan menggunakan metode penggabungan dan jumlah cluster yang ingin diketahui
4. Disini kami menggunakan *looping* untuk menampilkan dan menentukan pengaturan terbaik dari metode penggabungan dan jumlah cluster
5. Menampilkan hasil silhouette *score* terbaik dari agglomerative *clustering*

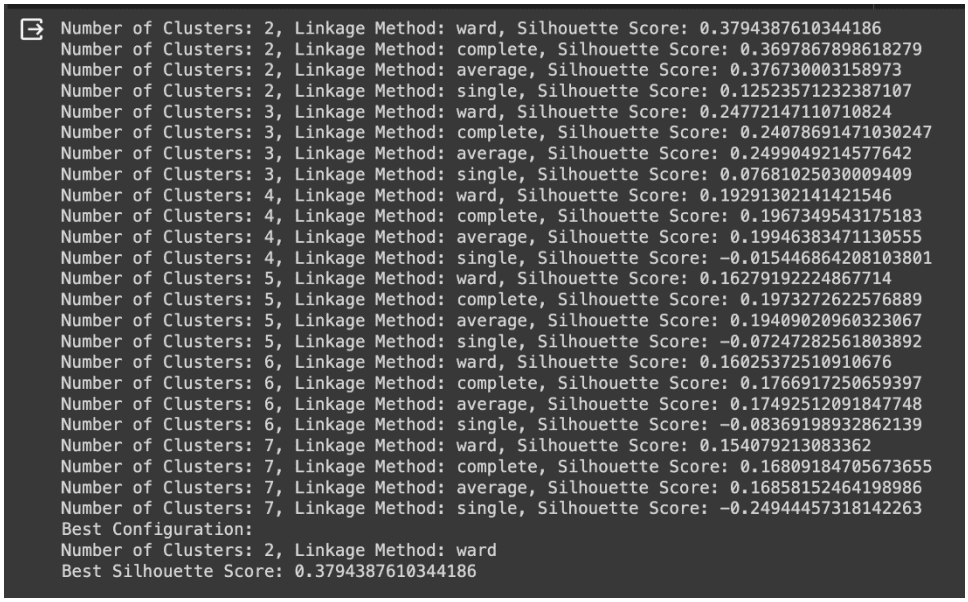
Berikut contoh kode yang kami gunakan dalam mencari silhouette *score* pada agglomerative *clustering*:

Tabel 4.1 Kode Agglomerative Clustering

```
# Pisahkan atribut prediktor
X = dataset.drop("price_range", axis=1) # Atribut prediktor

# Inisialisasi SimpleImputer untuk mengisi nilai-nilai yang hilang dengan median
imputer = SimpleImputer(strategy="median")
X_imputed = imputer.fit_transform(X)

# Inisialisasi model Agglomerative Clustering
num_clusters_range = range(2,8) # Ganti dengan jumlah cluster yang ingin dicoba
linkage_methods = ['ward', 'complete', 'average', 'single'] # Ganti dengan metode penggabungan yang ingin dicoba

best_silhouette_score = -1 # Inisialisasi skor Silhouette terbaik
best_config = None # Inisialisasi konfigurasi terbaik
```

```
# Loop melalui variasi jumlah cluster dan metode penggabungan
for num_clusters in num_clusters_range:
for linkage_method in linkage_methods:
# Inisialisasi model Agglomerative Clustering
agg_model = AgglomerativeClustering(n_clusters=num_clusters, linkage=linkage_method)

# Lakukan klustering dengan Agglomerative Clustering
labels = agg_model.fit_predict(X_imputed)

# Hitung Silhouette Score
silhouette_avg = silhouette_score(X_imputed, labels)

# Cetak hasil
print(f"Number of Clusters: {num_clusters}, Linkage Method: {linkage_method}, Silhouette Score: {silhouette_avg}")

# Periksa apakah hasil saat ini lebih baik
if silhouette_avg > best_silhouette_score:
best_silhouette_score = silhouette_avg
best_config = (num_clusters, linkage_method)

print("Best Configuration:")
print(f"Number of Clusters: {best_config[0]}, Linkage Method: {best_config[1]}")
print("Best Silhouette Score:", best_silhouette_score)
```

Hasil dari percobaan kami adalah mendapatkan silhouette *score* sebesar 37% atau 0.3794 dengan jumlah cluster 2 dan metode penggabungan ward.



Gambar 4.1 Hasil Silhouette Score Menggunakan Agglomerative Clustering

4.2 DBSCAN

Berikut cara yang dapat dilakukan untuk mencari silhouette score dengan menggunakan DBSCAN:

1. Memisahkan atribut prediktor dengan cara menggunakan fungsi *drop*
2. Melakukan *simple* imputer untuk mengisi nilai-nilai kosong dengan menggunakan median
3. Melakukan inisiasi standar *scaler* untuk standarisasi attribut prediktor
4. Menentukan nilai epsilon dan min sample value
5. Disini kami menggunakan looping untuk menampilkan dan menentukan nilai epsilon dan min sample terbaik untuk mencari silhouette score
6. Menampilkan hasil silhouette score terbaik dari DBCSAN

Berikut contoh code yang kami gunakan dalam mencari silhouette score pada DBSCAN:

Tabel 4.2 Kode DBSCAN

```
# Pisahkan atribut prediktor
X = dataset.drop("price_range", axis=1) # Atribut prediktor
```

```
# Inisialisasi SimpleImputer untuk mengisi nilai-nilai yang hilang dengan median
imputer = SimpleImputer(strategy="median")
X_imputed = imputer.fit_transform(X)

# Inisialisasi StandardScaler untuk standarisasi atribut prediktor
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Range nilai eps dan min_samples yang ingin dicoba
eps_values = [4.0, 4.35, 4.5, 4.65] # Ganti dengan nilai eps yang ingin dicoba
min_samples_values = [20, 30, 40, 50] # Ganti dengan nilai min_samples yang ingin dicoba

best_silhouette_score = -1 # Inisialisasi skor Silhouette terbaik
best_eps = None # Inisialisasi nilai eps terbaik
best_min_samples = None # Inisialisasi nilai min_samples terbaik

for eps in eps_values:
for min_samples in min_samples_values:
# Inisialisasi model DBSCAN dengan parameter yang disesuaikan
dbscan_model = DBSCAN(eps=eps, min_samples=min_samples)

# Lakukan klustering dengan DBSCAN
labels = dbscan_model.fit_predict(X_scaled)

# Hitung Silhouette Score
silhouette_avg = silhouette_score(X_scaled, labels)

# Tampilkan hasil
print(f"eps: {eps}, min_samples: {min_samples}, Silhouette Score: {silhouette_avg}")
```

```
# Periksa apakah hasil saat ini lebih baik
if silhouette_avg > best_silhouette_score:
best_silhouette_score = silhouette_avg
best_eps = eps
best_min_samples = min_samples

print("Best Configuration:")
print(f"eps: {best_eps}, min_samples: {best_min_samples}")
print("Best Silhouette Score:", best_silhouette_score)
```

Hasil dari percobaan kami adalah mendapatkan silhouette score sebesar 15% atau 0.1558 dengan nilai epsilon 4.65 dan min sample value 20.

```
eps: 4.0, min_samples: 20, Silhouette Score: 0.023588216255901122
eps: 4.0, min_samples: 30, Silhouette Score: 0.051242108900158484
eps: 4.0, min_samples: 40, Silhouette Score: 0.03560687181759633
eps: 4.0, min_samples: 50, Silhouette Score: 0.01766015885450677
eps: 4.35, min_samples: 20, Silhouette Score: 0.12264845314750267
eps: 4.35, min_samples: 30, Silhouette Score: 0.11160132137320525
eps: 4.35, min_samples: 40, Silhouette Score: 0.09995934950750388
eps: 4.35, min_samples: 50, Silhouette Score: 0.09098370687212781
eps: 4.5, min_samples: 20, Silhouette Score: 0.1374837222320706
eps: 4.5, min_samples: 30, Silhouette Score: 0.1386198864936266
eps: 4.5, min_samples: 40, Silhouette Score: 0.11918878221428339
eps: 4.5, min_samples: 50, Silhouette Score: 0.11275756066740253
eps: 4.65, min_samples: 20, Silhouette Score: 0.15587933570561763
eps: 4.65, min_samples: 30, Silhouette Score: 0.15587933570561763
eps: 4.65, min_samples: 40, Silhouette Score: 0.15587933570561763
eps: 4.65, min_samples: 50, Silhouette Score: 0.15209777024384283
Best Configuration:
eps: 4.65, min_samples: 20
Best Silhouette Score: 0.15587933570561763
```

Gambar 4.2 Hasil Silhouette Score Menggunakan DBSCAN

4.3 K-Means

Berikut cara yang dapat dilakukan untuk mencari silhouette score dengan menggunakan K-Means:

1. Memisahkan atribut prediktor dengan cara menggunakan fungsi drop
2. Melakukan simple imputer untuk mengisi nilai-nilai kosong dengan menggunakan median

3. Melakukan inisiasi jumlah cluster yang ingin digunakan
4. Disini kami menggunakan looping untuk menampilkan dan menentukan pengaturan terbaik dari jumlah cluster yang dicoba
5. Menampilkan hasil silhouette score terbaik dari K-Means

Berikut contoh code yang kami gunakan dalam mencari silhouette score pada k-means:

Tabel 4.3 Kode K-Means

```
# Pisahkan atribut prediktor
X = dataset.drop("price_range", axis=1) # Atribut prediktor

# Inisialisasi SimpleImputer untuk mengisi nilai-nilai yang hilang dengan median
imputer = SimpleImputer(strategy="median")
X_imputed = imputer.fit_transform(X)

# Range jumlah kluster yang ingin dicoba
num_clusters_range = range(2, 8) # Coba dari 2 hingga 7 kluster

# Inisialisasi nilai n_init yang ingin digunakan
n_init_value = 10 # Ganti dengan nilai n_init yang Anda inginkan

best_silhouette_score = -1 # Inisialisasi skor Silhouette terbaik

for num_clusters in num_clusters_range:
# Inisialisasi model K-Means dan setel n_init secara eksplisit
kmeans_model = KMeans(n_clusters=num_clusters, n_init=n_init_value)

# Lakukan klustering dengan K-Means
labels = kmeans_model.fit_predict(X_imputed)

# Hitung Silhouette Score
silhouette_avg = silhouette_score(X_imputed, labels)
```

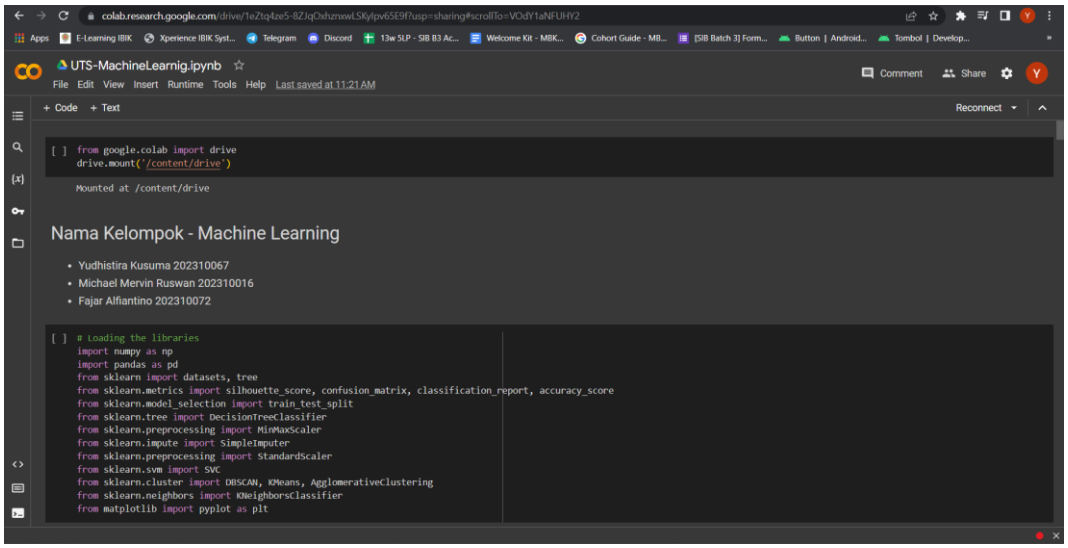
```
# Tampilkan hasil
print(f"Number of Clusters: {num_clusters}, Silhouette Score: {silhouette_avg}")

# Periksa apakah hasil saat ini lebih baik
if silhouette_avg > best_silhouette_score:
best_silhouette_score = silhouette_avg
best_config = num_clusters

print("Best Configuration:")
print(f"Number of Clusters: {best_config}")
print("Best Silhouette Score:", best_silhouette_score)
```

Hasil dari percobaan kami adalah mendapatkan silhouette score sebesar 39% atau 0.3983 dengan nilai cluster 2.

```
Number of Clusters: 2, Silhouette Score: 0.3983422741001593
Number of Clusters: 3, Silhouette Score: 0.27794814569235043
Number of Clusters: 4, Silhouette Score: 0.2620004173839964
Number of Clusters: 5, Silhouette Score: 0.24263073582900407
Number of Clusters: 6, Silhouette Score: 0.22662952026243344
Number of Clusters: 7, Silhouette Score: 0.22627946273518137
Best Configuration:
Number of Clusters: 2
Best Silhouette Score: 0.3983422741001593
```

Gambar 4.3 Hasil Silhouette Score Menggunakan K-Means

LAMPIRAN



Lampiran 1 *Screenshot* Google Collab

Link GoogleCollab:

https://colab.research.google.com/drive/1eZtq4ze5-8ZJqOxhznxwLSKyIpv65E9f?usp=sharing