

TUGAS 1

Kecerdasan Artifisial Lanjut

Teknik Informatika Kelas B

Nama:

Gaung Taqwa Indraswara (235150207111043)

Muhammad Fauzan (235150201111044)

Ferrel Destatiananda Edwardo (235150207111044)

Muhammad Fatir Zaira (23515020011039)

Dosen:

Putra Pandu Adikara, S.Kom., M.Kom.



**Program Studi Teknik Informatika
Fakultas Ilmu Komputer
Universitas Brawijaya**

DAFTAR ISI

DAFTAR ISI.....	1
BAB I - PENDAHULUAN.....	3
1.1 Latar Belakang.....	3
1.2 Rumusan Masalah.....	4
1.3 Tujuan Penelitian.....	4
1.4 Manfaat Penelitian.....	4
BAB II - KAJIAN PUSTAKA.....	5
2.1 Machine Learning.....	5
2.2 Supervised Learning.....	5
2.3 Klasifikasi.....	5
2.4 Decision Tree.....	5
2.5 Support Vector Machine.....	5
2.6 Random Forest.....	5
2.7 Evaluasi.....	6
BAB III - METODE PENELITIAN.....	7
3.1 Diagram Alur.....	7
3.1.1 Pengumpulan Data.....	7
3.1.2 Eksplorasi Data.....	7
3.1.3 Pre-pemrosesan Data.....	7
3.1.4 Modeling.....	7
3.1.5 Evaluasi Model.....	8
3.2 Teknik Pra-Pemrosesan.....	8
3.2.1 Raw Data.....	8
3.2.2 Data Type Conversion.....	8
3.2.3 Drop Column.....	8
3.2.4 Missing Value Handling.....	8
3.2.4 Duplicate Handling.....	9
3.2.5 Encoding.....	9
3.2.6 Scaling.....	9
3.2.7 Preprocessed Data.....	9
3.3 Tools dan Library.....	9
BAB IV - IMPLEMENTASI.....	11
4.1 Loading & Import Libraries.....	11
4.2 Exploratory Data Analysis.....	11
4.2.1 Understanding Data.....	11
4.2.2 Missing Value.....	14
4.2.3 Data Distribution.....	16
4.2.4 Analisis untuk Fitur Numerikal.....	17
4.2.5 Analisis untuk Fitur Kategorikal.....	18
4.2.6 Outlier.....	20

4.2.7 Correlation Matrix.....	21
4.3 Data Preprocessing.....	22
4.3.1 Handling Duplicate.....	22
4.3.2 Encoding.....	23
4.3.3 Scaling.....	24
4.4 Train-Test Split.....	25
4.5 Decision Tree.....	26
4.6 SVM.....	32
4.7 Random Forest.....	35
4.8 Evaluasi.....	36
4.8.2 Persiapan Variabel.....	42
4.8.3 Evaluasi Decision Tree.....	43
4.8.4 Evaluasi SVM.....	45
4.8.5 Evaluasi Random Forest.....	48
BAB V - PEMBAHASAN & HASIL.....	50
5.1 Analisis Data.....	50
5.2 Fokus Evaluasi.....	50
5.3 Analisis Kinerja Model.....	50
5.3.1 SVM Scikit-Learn.....	50
5.3.2 Random Forest.....	51
5.3.3 Decision Tree Scikit-Learn.....	51
5.3.4 Decision Tree Manual.....	51
5.3.5 SVM Manual.....	51
5.4 Analisis Performa Model Manual vs Scikit-Learn.....	52
BAB VI - KESIMPULAN & SARAN.....	53
6.1 Kesimpulan.....	53
6.2 Saran.....	53
STUDI PUSTAKA.....	54

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam era persaingan bisnis yang semakin ketat, customer churn atau kehilangan pelanggan menjadi tantangan besar bagi perusahaan, terutama di industri telekomunikasi dan perbankan. Customer churn mengacu pada fenomena di mana pelanggan berhenti menggunakan layanan atau beralih ke kompetitor (Arina & Ulfah, 2022). Tingginya tingkat churn berdampak signifikan terhadap pendapatan perusahaan, mengingat biaya untuk mendapatkan pelanggan baru lima kali lebih mahal daripada mempertahankan pelanggan yang sudah ada (Dhangar & Anand, 2021). Selain itu, peningkatan retensi pelanggan sebesar 5% saja dapat meningkatkan keuntungan perusahaan hingga 25% (Zhao, 2023).

Industri telekomunikasi menghadapi persaingan yang sangat tinggi dengan banyaknya penyedia layanan. Pelanggan cenderung mudah beralih provider jika merasa tidak puas dengan layanan atau harga (Arina & Ulfah, 2022). Sementara itu, di industri perbankan, faktor seperti perubahan kebijakan kredit, layanan yang kurang responsif, atau tawaran lebih menarik dari kompetitor dapat memicu pelanggan untuk menutup rekening atau beralih bank (Zhao, 2023). Oleh karena itu, kemampuan memprediksi pelanggan yang berpotensi churn menjadi krusial bagi perusahaan untuk mengambil tindakan preventif, seperti memberikan insentif atau layanan khusus guna mempertahankan mereka.

Machine learning telah menjadi solusi efektif dalam memprediksi customer churn dengan akurasi tinggi. Beberapa algoritma seperti Random Forest, Decision Tree, dan Support Vector Machine (SVM) telah terbukti mampu mengidentifikasi pola perilaku pelanggan yang berisiko churn (Dhangar & Anand, 2021; Zhao, 2023). Misalnya, penelitian Zhao (2023) menunjukkan bahwa model Random Forest mencapai akurasi 91% dalam memprediksi churn pelanggan bank, dengan fitur seperti total transaksi dan perubahan frekuensi transaksi menjadi prediktor utama. Sementara itu, Arina dan Ulfah (2022) menggunakan analisis survival dengan regresi Cox untuk mengidentifikasi faktor-faktor seperti status perkawinan dan lama tinggal yang memengaruhi ketahanan pelanggan telekomunikasi.

Namun, penelitian sebelumnya masih memiliki beberapa keterbatasan. Pertama, sebagian besar studi berfokus pada satu industri tertentu, seperti telekomunikasi atau perbankan, sehingga temuan belum tentu dapat digeneralisasikan ke sektor lain. Kedua, beberapa pendekatan seperti Decision Tree cenderung overfitting jika tidak dioptimalkan dengan teknik seperti cross-validation (Dhangar & Anand, 2021). Selain itu, masalah ketidakseimbangan data, di mana jumlah pelanggan non-churn jauh lebih banyak daripada pelanggan churn, sering kali diabaikan dalam pemodelan, padahal hal ini dapat memengaruhi akurasi prediksi (Zhao, 2023).

Berdasarkan gap tersebut, penelitian ini bertujuan untuk mengembangkan model prediksi churn yang lebih robust dengan memanfaatkan algoritma machine learning terbaru dan teknik penanganan data tidak seimbang seperti Synthetic Minority Oversampling Technique. Dengan demikian,

perusahaan dapat lebih efektif mengidentifikasi pelanggan berisiko dan mengambil langkah retensi yang tepat sebelum terjadi churn.

1.2 Rumusan Masalah

- Bagaimana performa model klasifikasi seperti *Decision Tree*, *SVM* dan *Decision Tree* dalam memprediksi *churn* pelanggan?
- Algoritma mana yang paling akurat dalam kasus dataset *Telco Customer Churn*?

1.3 Tujuan Penelitian

- Membangun model prediktif untuk klasifikasi churn pelanggan.
- Membandingkan performa antara beberapa algoritma klasifikasi.
- Menyediakan evaluasi model berdasarkan metrik akurasi, presisi, recall, dan F1-score.

1.4 Manfaat Penelitian

- Memberikan alat prediktif berbasis machine learning untuk mengidentifikasi pelanggan berisiko *churn* dengan akurasi tinggi, sehingga perusahaan dapat merancang strategi retensi (seperti *loyalty program* atau *personalized offer*) secara tepat sasaran.
- Mengurangi biaya akuisisi pelanggan baru dengan mempertahankan pelanggan eksisting, mengingat biaya retensi 5 kali lebih murah daripada akuisisi (Dhangar & Anand, 2021).
- Meningkatkan kepuasan pelanggan melalui analisis pola *churn* (misalnya, fitur layanan yang sering dikeluhkan) sebagai dasar perbaikan layanan.
- Hasil penelitian dapat diadaptasi oleh industri lain (e.g., *e-commerce*, asuransi) untuk memprediksi churn dengan pendekatan serupa.
- Hasil penelitian dapat menjadi dasar eksplorasi algoritma lain (e.g., *XGBoost*, *Neural Networks*) atau pendekatan hybrid untuk optimasi prediksi *churn*.

BAB II

KAJIAN PUSTAKA

2.1 *Machine Learning*

Machine learning merupakan cabang dari *Artificial Intelligence* (AI) untuk menganalisis data dan meningkatkan performa otomatis berdasarkan pembelajaran dari data (Sarker, 2021). *Machine learning* adalah teknologi yang dapat belajar melalui data. *Machine learning* menggunakan algoritma untuk menganalisis data, mengidentifikasi pola, dan membuat prediksi atau keputusan.

2.2 *Supervised Learning*

Supervised learning merupakan salah satu pendekatan dalam machine learning yang digunakan untuk membangun model prediktif berdasarkan data yang telah diberi label. Algoritma supervised learning membangun model dari data latih, dan model tersebut dapat digunakan untuk mengklasifikasikan data yang belum diberi label. Dalam konteks ini, algoritma belajar dari pasangan data *input* dan *output* untuk memetakan pola yang dapat digunakan dalam mengklasifikasikan data baru yang belum diketahui kelasnya.

2.3 Klasifikasi

Klasifikasi adalah teknik *machine learning* yang bertujuan untuk memetakan data input ke dalam kategori atau kelas yang telah ditentukan sebelumnya.. Klasifikasi merupakan bagian dari *supervised learning*, karena memerlukan data latih yang berlabel. Model klasifikasi yang baik mampu memprediksi kelas untuk data baru secara akurat.

2.4 *Decision Tree*

Decision tree merupakan metode data mining yang umum digunakan untuk membangun sistem klasifikasi. Metode ini mengklasifikasikan populasi ke dalam segmen-segmen bercabang yang membentuk struktur pohon terbaik dengan *node* akar, *node* internal, dan *node* daun. Algoritma ini bersifat non-parametrik dan dapat secara efisien menangani dataset yang besar dan kompleks tanpa memerlukan struktur parametrik yang rumit.

2.5 *Support Vector Machine*

Support Vector Machine (SVM) adalah teknik pembelajaran mesin yang digunakan untuk klasifikasi. SVM bekerja dengan mencari *hyperplane* optimal yang memisahkan data dari kelas yang berbeda dengan margin maksimum. dalam kasus jika data tidak dapat dipisahkan secara linear, SVM menggunakan fungsi kernel untuk memetakan data ke dalam ruang berdimensi lebih tinggi yang memungkinkan pemisahan linear.

2.6 *Random Forest*

Random Forest merupakan algoritma yang dikembangkan dari konsep pohon keputusan, di mana model terdiri dari sejumlah pohon yang dibentuk melalui metode *bootstrapping*. Dengan

menggabungkan hasil prediksi dari banyak pohon, algoritma ini mampu meningkatkan akurasi klasifikasi. Random Forest terbukti efektif untuk tugas klasifikasi, regresi, dan evaluasi fitur, serta memiliki keunggulan dalam menangani dataset berskala besar dengan banyak variabel dan mengurangi risiko *overfitting* (Lebanov et al., 2020).

2.7 Evaluasi

Evaluasi model klasifikasi bertujuan untuk mengukur kinerja model dalam memprediksi kelas target secara akurat. Metrik evaluasi seperti akurasi, presisi, recall, dan F1-score merupakan indikator umum yang digunakan untuk menilai performa model klasifikasi (Vujović, 2021). Dalam praktiknya, metrik-metrik tersebut biasanya dihitung melalui confusion matrix, yang memetakan hasil prediksi terhadap label sebenarnya dalam empat kategori: true positive, false positive, true negative, dan false negative.

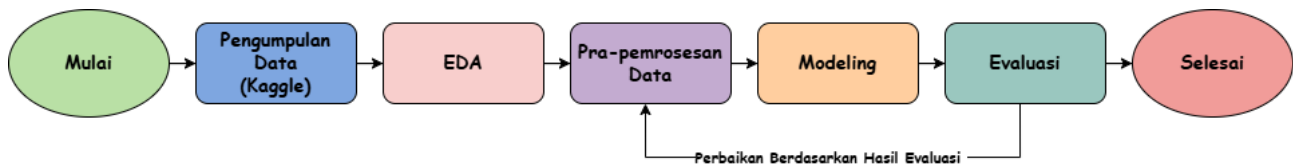
Dalam penugasan ini, evaluasi dilakukan menggunakan `accuracy_score` untuk mengukur proporsi prediksi yang benar, serta `classification_report` yang menyediakan nilai presisi, recall, dan F1-score untuk masing-masing kelas. Penggunaan metrik ini mencerminkan pendekatan evaluasi yang komprehensif, untuk menilai berbagai aspek kinerja model klasifikasi secara seimbang (Vujović, 2021).

BAB III

METODE PENELITIAN

3.1 Diagram Alur

Diagram alur berikut menggambarkan tahapan utama dalam proses pembuatan model prediksi churn pelanggan, dimulai dari pengumpulan data hingga evaluasi dan perbaikan model jika diperlukan. Diagram ini membantu memahami urutan dan hubungan antar proses dalam proyek machine learning.



3.1.1 Pengumpulan Data

Dataset yang digunakan adalah *Telco Customer Churn* yang diperoleh dari situs Kaggle. Dataset ini berisi informasi dan status churn pelanggan dari perusahaan telekomunikasi, apakah pelanggan tersebut berhenti berlangganan atau tidak. Dataset ini mencakup berbagai informasi pelanggan seperti lama berlangganan, jenis layanan yang digunakan, metode pembayaran, hingga total biaya bulanan. Data ini bertujuan untuk membantu mengidentifikasi karakteristik pelanggan yang berisiko churn agar perusahaan dapat mengambil tindakan preventif.

3.1.2 Eksplorasi Data

Melakukan eksplorasi data untuk memahami karakteristik *dataset* yang digunakan. Menganalisis data untuk mengidentifikasi pola dan anomali pada data. Dari hasil analisis, dapat ditentukan perlakuan yang diperlukan untuk mempersiapkan data di tahap selanjutnya. Langkah ini juga membantu dalam pemilihan fitur yang paling relevan untuk membangun model prediktif.

3.1.3 Pre-pemrosesan Data

Melakukan pembersihan data yang mencakup tahapan seperti menangani nilai yang hilang, serta menangani pencilaan pada data. Fitur kategorikal diubah ke bentuk numerik agar lebih mudah diimplementasikan dalam algoritma klasifikasi. Selain itu, fitur numerik juga diskalakan agar tidak terpengaruh nilai ekstrim dan tidak memperburuk akurasi model. Pada tahap ini juga dilakukan pembagian data menjadi data latih dan data uji dengan proporsi 80:20 untuk memastikan evaluasi model dilakukan terhadap data yang belum pernah dilihat sebelumnya.

3.1.4 Modeling

Model klasifikasi yang dilatih adalah yaitu *Decision Tree*, *Support Vector Machine* dan *Random Forest*. *Decision Tree* dan *Random Forest* dipilih sebagai bagian dari ketentuan

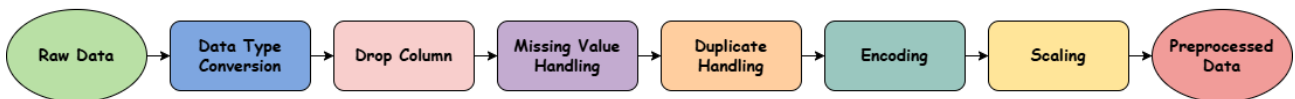
tugas yang mengharuskan penggunaan kedua algoritma yang diterapkan secara manual dan menggunakan *library*. *Random forest* dipilih karena efektif untuk klasifikasi data, prediksi nilai kontinu, dan penilaian kualitas, serta mampu menangani *dataset* besar dengan variabel kompleks dan tahan terhadap *overfitting* (Lebanov *et al.*, 2020).

3.1.5 Evaluasi Model

Evaluasi dilakukan dengan melakukan prediksi pada data uji menggunakan model yang telah dilatih. Kinerja model diukur dengan metrik akurasi, presisi, *recall*, *F1-score* untuk menilai seberapa baik model dalam mengklasifikasikan data *churn*. Dengan evaluasi, dapat diketahui model mana yang paling optimal dalam prediksi *churn* pelanggan

3.2 Teknik Pra-Pemrosesan

Diagram teknik pra-pemrosesan merinci langkah-langkah yang dilakukan untuk membersihkan dan menyiapkan data mentah sebelum digunakan dalam pemodelan. Setiap tahap, mulai dari konversi tipe data hingga *scaling*, bertujuan untuk memastikan data dalam kondisi optimal untuk analisis dan pelatihan model.



3.2.1 Raw Data

Raw data atau data mentah merupakan data asli yang diperoleh langsung sumber dan belum melalui proses pengolahan. Data mentah sering memiliki berbagai kekurangan seperti nilai yang hilang, data duplikat, pencilan, dan sebagainya. Memahami data mentah menjadi langkah awal dalam *pipeline* analisis data untuk menentukan pemrosesan yang tepat.

3.2.2 Data Type Conversion

Data Type Conversion adalah proses mengubah tipe data kolom agar sesuai dengan kebutuhan analisis dan algoritma yang akan digunakan. Tahap ini penting karena jika tipe data tidak sesuai dengan algoritma akan menghasilkan model yang buruk bahkan terjadi eror pada prosesnya. Konversi yang umum dilakukan adalah pengubahan tipe data kategorik ke numerik, dan ekstrak informasi dari format *datetime*.

3.2.3 Drop Column

Drop column adalah teknik untuk menghapus kolom atau fitur yang tidak terpakai. Indikasi kolom yang dapat dihapus seperti fitur yang memiliki pengaruh kecil terhadap model prediktif atau data pada kolom tersebut hanya memiliki satu nilai unik. Tahap ini berguna untuk mengurangi dimensional data, mengurangi risiko *overfitting*.

3.2.4 Missing Value Handling

Missing Value Handling merupakan tahap penanganan terhadap nilai yang hilang. Penanganan nilai yang hilang dapat berupa pengisian data dengan rata-rata, median, atau

modus, tergantung pada tipe data dan distribusinya. Tujuan dari *handling missing value* adalah untuk memastikan bahwa analisis data atau proses machine learning berjalan dengan akurat dan efisien tanpa terganggu oleh data yang tidak lengkap.

3.2.4 Duplicate Handling

Mengatasi permasalahan data yang teridentifikasi duplikat, yang dimana data duplikat dapat menyebabkan bias. Penanganan duplikasi termasuk penghapusan langsung, atau penandaan khusus untuk analisis lebih lanjut. Selain mengurangi risiko bias, *Duplicate handling* dapat mengatasi pembengkakan ukuran *dataset* yang tidak perlu.

3.2.5 Encoding

Encoding merupakan transformasi data kategorik menjadi format numerik yang dapat diproses oleh algoritma *machine learning*, kebanyakan algoritma tidak dapat langsung memproses data tekstual sehingga teknik *encoding* perlu digunakan. Beberapa teknik *encoding* yang umum digunakan adalah *one-hot encoding*, *label encoding*, dan *target encoding*. Pemilihan teknik encoding yang didasari pada kardinalitas fitur, hubungan antar kategori, dan kebutuhan algoritma yang digunakan.

3.2.6 Scaling

Scaling adalah proses menyesuaikan rentang nilai fitur numerik agar berada dalam skala yang sebanding. Tahap ini mencegah fitur dengan nilai besar mendominasi yang bernilai kecil dalam perhitungan jarak atau bobot. Metode umum termasuk *Min-Max Scaling*, *Standardization*, dan *Robust Scaling*. *Scaling* sangat penting untuk algoritma yang sensitif terhadap skala dan metode berbasis gradien. *Scaling* mengurangi jarak antara unit data dengan membuatnya lebih generik (Sharma, 2022).

3.2.7 Preprocessed Data

Preprocessed data adalah data yang telah melewati tahapan pembersihan dan transformasi. Pada tahap ini, data siap untuk digunakan dalam pemodelan. Data ini memiliki format yang konsisten, bebas dari anomali, dan strukturnya telah disesuaikan dengan kebutuhan dari algoritma yang akan diterapkan. Kualitas *preprocessed data* memengaruhi performa model akhir, sehingga proses pra pemrosesan harus dilakukan dengan cermat dengan mempertimbangkan karakteristik data dan tujuan analisis.

3.3 Tools dan Library

Dalam pengerjaan digunakan beberapa *tools* dan *library* dari Python untuk membantu proses analisis dan pembuatan model *machine learning*. Berikut adalah *tools* dan *library* yang digunakan:

1. Python
Bahasa pemrograman utama yang digunakan untuk mengembangkan model machine learning dan melakukan analisis data.
2. Pandas
Digunakan untuk manipulasi dan analisis data, termasuk membaca file CSV, menangani data yang hilang, serta melakukan encoding terhadap data kategori.

3. NumPy

Digunakan untuk operasi matematika dan array numerik, mendukung komputasi numerik yang efisien.

4. Matplotlib dan Seaborn

Kedua *library* ini digunakan untuk visualisasi data, membantu memahami distribusi data dan hubungan antar fitur.

5. Scikit-learn (sklearn)

Library utama yang digunakan untuk membangun model machine learning mencakup beberapa komponen penting, seperti *preprocessing* data menggunakan *StandardScaler*, *OneHotEncoder*, dan *LabelEncoder*; pembangunan model menggunakan *DecisionTreeClassifier*, *SVC*, dan algoritma lainnya; evaluasi performa model dengan *classification_report* dan *accuracy_score*; serta pembagian data latih dan uji menggunakan *train_test_split*.

BAB IV

IMPLEMENTASI

4.1 *Loading & Import Libraries*

Pada tahap ini dilakukan pemanggilan pustaka dan men-*download* data yang kemudian disimpan pada suatu variabel.

4.1.1 Kode

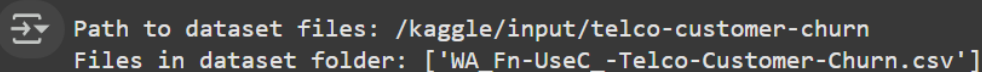
```
import kagglehub
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import LabelEncoder, RobustScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

path = kagglehub.dataset_download("blastchar/telco-customer-churn")
print("Path to dataset files:", path)

files = os.listdir(path)
print("Files in dataset folder:", files)

data_path = os.path.join(path, 'WA_Fn-UseC_-Telco-Customer-Churn.csv')
data = pd.read_csv(data_path)
```

4.1.2 Output



```
Path to dataset files: /kaggle/input/telco-customer-churn
Files in dataset folder: ['WA_Fn-UseC_-Telco-Customer-Churn.csv']
```

4.2 *Exploratory Data Analysis*

Pada tahap ini diimplementasikan beberapa kode untuk menganalisis karakteristik data. kemudian ditentukan perlakuan apa yang sesuai untuk menangani karakteristik data. Kode yang diimplementasikan diantaranya untuk identifikasi nilai yang hilang, data duplikat, pencilan, plot untuk mengetahui distribusi data, dan sebagainya.

4.2.1 *Understanding Data*

4.2.1.1 Kode

```
print(data)

data.info()

data.describe()

data['Churn'].value_counts()

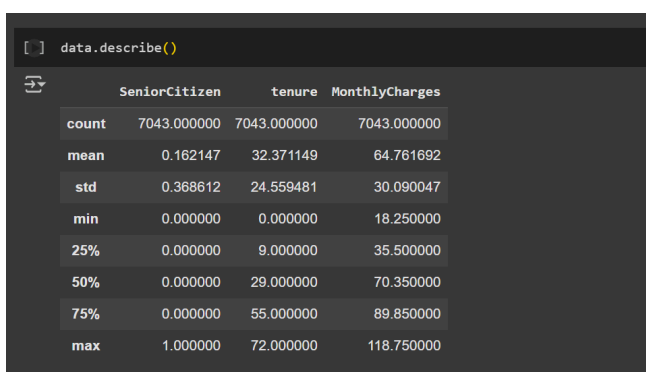
percentages = data['Churn'].value_counts(normalize=True) * 100
percentages.round(2)

unique_values = []

for col in data.columns:
    if col != 'customerID':
        unique_values.append({
            'Column': col,
            'Unique Values': data[col].unique(),
            'Len unique values': len(data[col].unique())
        })

unique_per_col = pd.DataFrame(unique_values)
unique_per_col
```

4.2.1.2 Output



The screenshot shows a Jupyter Notebook interface with a code cell containing `data.describe()` and its corresponding output. The output is a table with 8 rows and 4 columns. The first column lists statistical measures (count, mean, std, min, 25%, 50%, 75%, max), and the next three columns show the values for 'SeniorCitizen', 'tenure', and 'MonthlyCharges' respectively.

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

```
data['Churn'].value_counts()
```

Churn	count
No	5174
Yes	1869

dtype: int64

```
[ ] percentages = data['Churn'].value_counts(normalize=True) * 100
percentages.round(2)
```

Churn	proportion
No	73.46
Yes	26.54

dtype: float64

```
unique_values = []

for col in data.columns:
    if col != 'customerID':
        unique_values.append({
            'Column': col,
            'Unique Values': data[col].unique(),
            'Len unique values': len(data[col].unique())
        })

unique_per_col = pd.DataFrame(unique_values)
unique_per_col
```

	Column	Unique Values	Len unique values
0	gender	[Female, Male]	2
1	SeniorCitizen	[0, 1]	2
2	Partner	[Yes, No]	2
3	Dependents	[No, Yes]	2
4	tenure	[1, 34, 2, 45, 8, 22, 10, 28, 62, 13, 16, 58, ...]	73
5	PhoneService	[No, Yes]	2
6	MultipleLines	[No phone service, No, Yes]	3
7	InternetService	[DSL, Fiber optic, No]	3
8	OnlineSecurity	[No, Yes, No internet service]	3
9	OnlineBackup	[Yes, No, No internet service]	3
10	DeviceProtection	[No, Yes, No internet service]	3
11	TechSupport	[No, Yes, No internet service]	3
12	StreamingTV	[No, Yes, No internet service]	3
13	StreamingMovies	[No, Yes, No internet service]	3
14	Contract	[Month-to-month, One year, Two year]	3
15	PaperlessBilling	[Yes, No]	2
16	PaymentMethod	[Electronic check, Mailed check, Bank transfer...]	4
17	MonthlyCharges	[29.85, 56.95, 53.85, 42.3, 70.7, 99.65, 89.1, ...]	1585
18	TotalCharges	[29.85, 1889.5, 108.15, 1840.75, 151.65, 820.5, ...]	6531
19	Churn	[No, Yes]	2

4.2.1.3 Penjelasan

Dari data.info() dapat dilihat bahwa ada 3 kolom numerik dan 17 kolom bertipe object, namun dilihat dari print(data), sebenarnya kolom 'TotalCharges' adalah kolom numerik namun disimpan sebagai string/object, lalu ada juga kolom biner 'SeniorCitizen' yang disimpan dalam integer

padahal kolom biner lainnya tersimpan dalam string *yes* dan *no*, beberapa ketidakkonsistenan tipe data ini menjadi *concern* dalam tahap-tahap selanjutnya.

Karena `describe()` hanya menampilkan kolom numerik maka ditampilkanlah 3 kolom ini, namun masalahnya untuk 'SeniorCitizen' adalah kolom biner 1 dan 0 dan kolom 'TotalCharges' yang sebenarnya adalah kolom numerik masih belum ditampilkan karena masih dalam bentuk tipe data *object*.

Dari persebaran target bisa dilihat bahwa dataset ini imbalance dengan *record churn* berkisar antara 26% dari total keseluruhan data, *insight* ini akan ditindak lanjuti dengan melakukan *stratify train-test split*.

4.2.2 Missing Value

4.2.2.1 Kode

```
data.isnull().sum()

datav = data.copy()

datav['TotalCharges'] = pd.to_numeric(datav['TotalCharges'], errors='coerce')

datav.isna().sum()

mask = pd.to_numeric(data['TotalCharges'], errors='coerce').isna()
data.loc[mask, 'TotalCharges'].unique()

print(data[data['TotalCharges'] == ''])

datav['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coerce')
datav['TotalCharges'].fillna(0, inplace=True)

datav['TotalCharges'].isnull().sum()

datav.info()
```

4.2.2.2 Output

```
data.isnull().sum()
```

	0
customerID	0
gender	0
SeniorCitizen	0
Partner	0
Dependents	0
tenure	0
PhoneService	0
MultipleLines	0
InternetService	0
OnlineSecurity	0
OnlineBackup	0
DeviceProtection	0
TechSupport	0
StreamingTV	0
StreamingMovies	0
Contract	0
PaperlessBilling	0
PaymentMethod	0
MonthlyCharges	0
TotalCharges	0
Churn	0

dtype: int64

Setelah di cek tidak ada data yang null jika menggunakan isnull

```
[ ] datav = data.copy()
```

```
[ ] datav['TotalCharges'] = pd.to_numeric(datav['TotalCharges'], errors='coerce')
```

```
datav.isna().sum()
```

	0
customerID	0
gender	0
SeniorCitizen	0
Partner	0
Dependents	0
tenure	0
PhoneService	0
MultipleLines	0
InternetService	0
OnlineSecurity	0
OnlineBackup	0
DeviceProtection	0
TechSupport	0
StreamingTV	0
StreamingMovies	0
Contract	0
PaperlessBilling	0
PaymentMethod	0
MonthlyCharges	0
TotalCharges	11
Churn	0

```
datav.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

RangeIndex: 7043 entries, 0 to 7042

Data columns (total 21 columns):

#	Column	Non-Null Count	Dtype
0	customerID	7043 non-null	object
1	gender	7043 non-null	object
2	SeniorCitizen	7043 non-null	int64
3	Partner	7043 non-null	object
4	Dependents	7043 non-null	object
5	tenure	7043 non-null	int64
6	PhoneService	7043 non-null	object
7	MultipleLines	7043 non-null	object
8	InternetService	7043 non-null	object
9	OnlineSecurity	7043 non-null	object
10	OnlineBackup	7043 non-null	object
11	DeviceProtection	7043 non-null	object
12	TechSupport	7043 non-null	object
13	StreamingTV	7043 non-null	object
14	StreamingMovies	7043 non-null	object
15	Contract	7043 non-null	object
16	PaperlessBilling	7043 non-null	object
17	PaymentMethod	7043 non-null	object
18	MonthlyCharges	7043 non-null	float64
19	TotalCharges	7043 non-null	float64
20	Churn	7043 non-null	object

dtypes: float64(2), int64(2), object(17)

memory usage: 1.1+ MB

4.2.2.3 Penjelasan

Setelah pengecekan pertama tidak ada data yang *null* jika menggunakan *isnull*. Namun setelah tipe data kolom 'TotalCharges' diubah menjadi numerik ternyata ada kolom yang *null*, hal ini terjadi karena ada *record* yang menyimpan hanya spasi.

dari hasil analisis, ternyata data spasi atau *null* ini merupakan data para pelanggan yang baru bergabung kurang dari 1 bulan, berikut adalah *evidence* yang ditemukan untuk dilakukan penanganan missing value.

- 'tenure' = 0 untuk semua 11 *record* → artinya pelanggan baru mendaftar.
- 'TotalCharges' = " " → memang belum membayar apa-apa karena belum ada tagihan berjalan.
- 'MonthlyCharges' sudah ada nilainya → menunjukkan paket yang dipilih sudah diketahui.
- 'Churn' = No → mereka masih aktif saat data dicatat.
- Dengan ini ditetapkan untuk imputasi nilai 0 di kolom 'TotalCharges'

4.2.3 Data Distribution

4.2.3.1 Kode

```
datav['SeniorCitizen'] = datav['SeniorCitizen'].map({0: 'No', 1: 'Yes'})

num_col = datav.select_dtypes(include=['int64', 'float64']).columns
cat_col = datav.select_dtypes(include=['object']).columns.drop('customerID')

print(num_col)
print(cat_col)
```

4.2.3.2 Output

```
[ ] datav['SeniorCitizen'] = datav['SeniorCitizen'].map({0: 'No', 1: 'Yes'})

num_col = datav.select_dtypes(include=['int64', 'float64']).columns
cat_col = datav.select_dtypes(include=['object']).columns.drop('customerID')

print(num_col)
print(cat_col)

Index(['tenure', 'MonthlyCharges', 'TotalCharges'], dtype='object')
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService',
       'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
       'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn'],
      dtype='object')
```

4.2.3.3 Penjelasan

Untuk mendapatkan kekonsistenan di tahap EDA kolom 'SeniorCitizen' kamu buat untuk memiliki nilai *yes* atau *no* supaya sama seperti kolom biner lainnya dan mempermudah pembuatan grafik nantinya.

4.2.4 Analisis untuk Fitur Numerikal

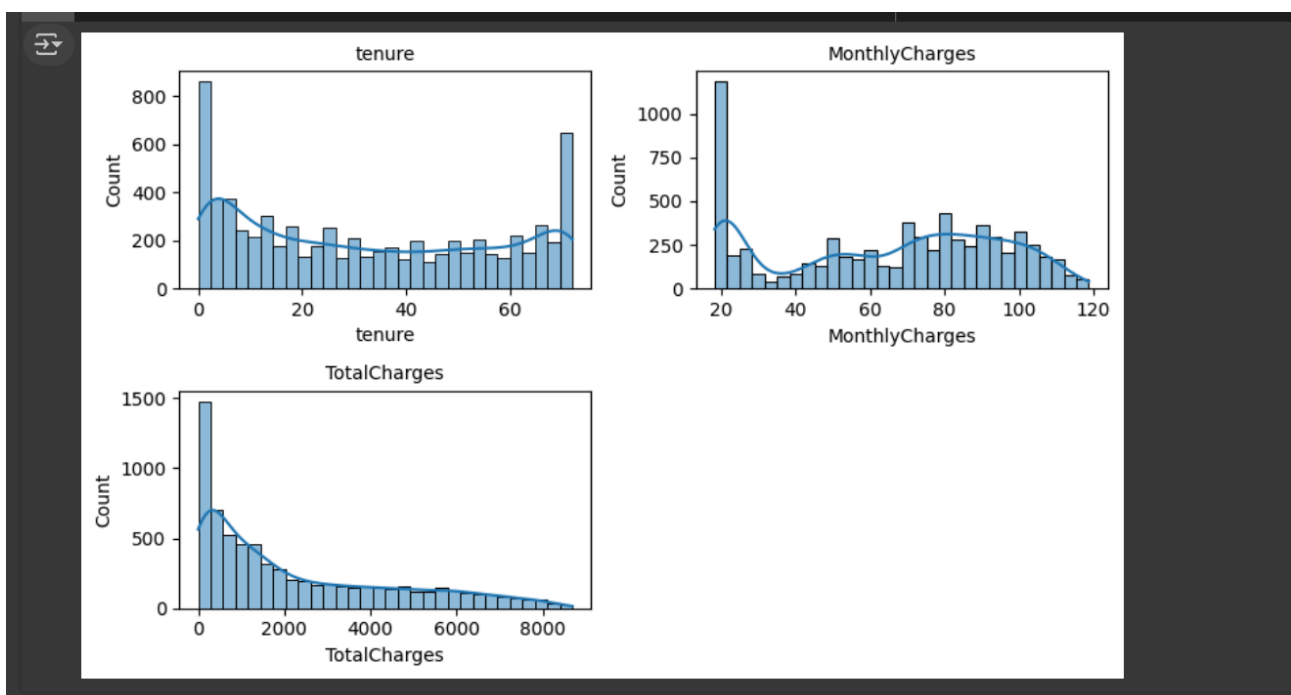
4.2.4.1 Kode

```
plt.figure(figsize=(8, 5))

# Banyak subplot berdasarkan jumlah kolom
for idx, col in enumerate(num_col):
    plt.subplot(2, (len(num_col)+1)//2, idx+1) # 2 baris, sisanya otomatis
    sns.histplot(datav[col], kde=True, bins=30)
    plt.title(col, fontsize=10)
    plt.tight_layout()

plt.show()
```

4.2.4.2 Output



4.2.4.3 Penjelasan

Dari distribusi data numerik bisa dilihat bahwa

- Tenure:
 - o Menunjukkan distribusi bimodal dengan puncak di nilai rendah (sekitar 0-1 bulan) dan nilai tinggi (sekitar 70 bulan)

- Mengindikasikan dua kelompok pelanggan yang dominan: pelanggan baru dan pelanggan loyal jangka panjang
- Distribusi cukup merata di antara kedua puncak tersebut
- MonthlyCharges :
 - Menunjukkan distribusi yang cukup merata dengan sedikit right-skewed
 - Memiliki puncak di nilai rendah (sekitar 20) dan beberapa puncak kecil di sepanjang range nilai
 - Range nilai berkisar antara 20-120
- TotalCharges :
 - Sangat right-skewed (distribusi ekor panjang ke kanan)
 - Mayoritas nilai berada di range rendah dengan ekor yang memanjang ke nilai yang lebih tinggi
 - Ini menunjukkan bahwa sebagian besar pelanggan memiliki total biaya yang relatif rendah, dengan beberapa pelanggan yang memiliki nilai ekstrem tinggi

Kondisi diatas membuat pemilihan scaler menjadi penting agar model tidak bias terhadap nilai ekstrem. Karena karakteristik *RobustScaler* yang menggunakan median dan IQR, metode ini lebih tahan terhadap pencilan dibandingkan *StandardScaler* atau *MinMaxScaler*. Oleh karena itu, dipilih *RobustScaler* untuk menangani distribusi data numerik tersebut agar hasil *preprocessing* lebih stabil dan representatif.

4.2.5 Analisis untuk Fitur Kategorikal

4.2.5.1 Kode

```
plt.figure(figsize=(18, 10))

for idx, col in enumerate(cat_col):
    plt.subplot((len(cat_col)+5)//6, 6, idx+1) # 6 kolom per baris
    ax = sns.countplot(x=datav[col], order=datav[col].value_counts().index)

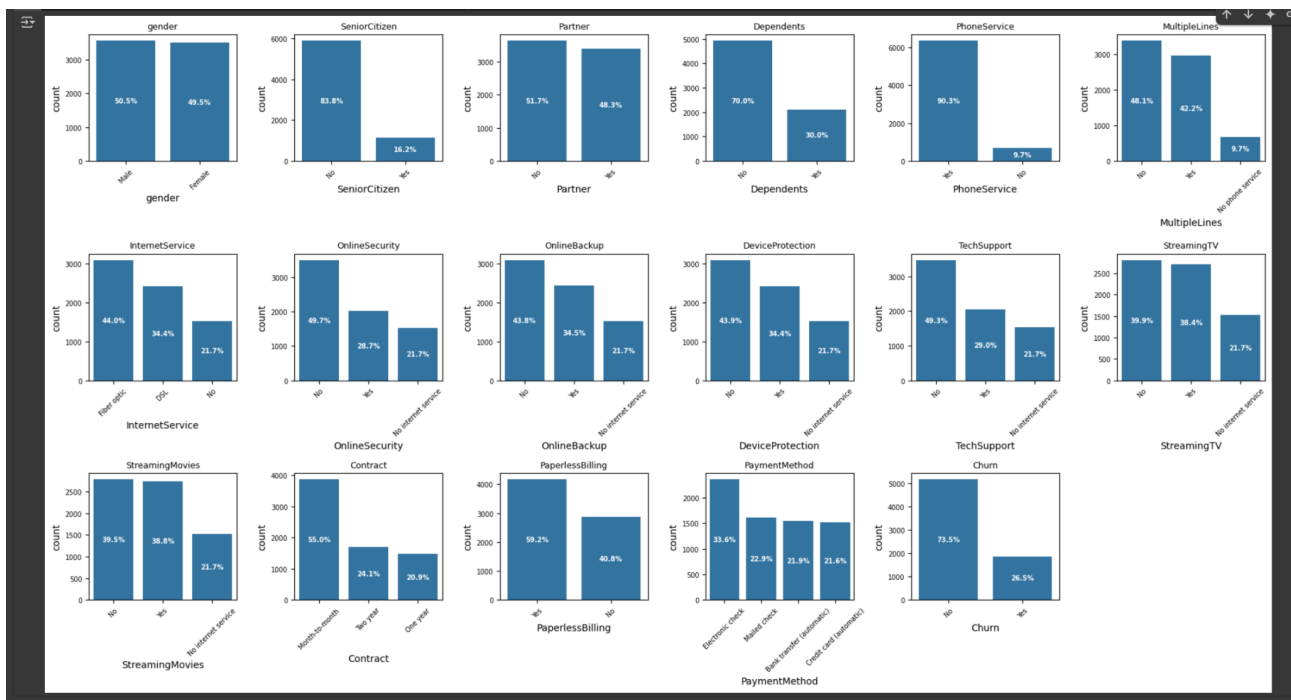
    # Tambahkan persentase di setiap bar
    total = len(datav)
    for p in ax.patches:
        height = p.get_height()
        percentage = f'{100 * height / total:.1f}%'
        ax.annotate(percentage,
                    (p.get_x() + p.get_width() / 2., height / 2),
                    ha='center', va='center',
                    fontsize=7, color='white', fontweight='bold')

    plt.title(col, fontsize=9)
    plt.xticks(rotation=45, fontsize=7)
```

```
plt.yticks(fontsize=7)
plt.tight_layout()

plt.show()
```

4.2.5.2 Output



4.2.5.3 Penjelasan

- Fitur Binary (2 kategori). Contoh: gender, SeniorCitizen, Partner, Dependents, PhoneService, PaperlessBilling, Churn.
 - o Distribusi relatif seimbang untuk gender dan Partner.
 - o Distribusi timpang pada SeniorCitizen (hanya 16.2% adalah senior), PhoneService (90.3% punya layanan), Dependents (mayoritas tidak punya tanggungan), dan Churn (hanya 26.5% churn).
 - o Insight:
 - Ketidakeimbangan seperti ini bisa menjadi faktor penting dalam modeling (dilakukan stratified split).
 - Churn yang imbalanced perlu perhatian khusus di evaluasi (pakai metrik seperti recall dan F1-score).
- Fitur Ternary atau Lebih (3+ kategori) Contoh: InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, Contract, PaymentMethod, dll.

- Banyak fitur memiliki kategori "No internet service" atau "No phone service", yang pada dasarnya berarti tidak berlaku.
- Fitur seperti Contract dan PaymentMethod punya distribusi yang berbeda-beda tapi tetap valid untuk model.
- Insight:
 - Fitur seperti OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, dan StreamingMovies hanya relevan bagi pelanggan yang memiliki layanan internet. Ketika InternetService = No, maka semua fitur tersebut akan otomatis bernilai "No internet service".
 - Beberapa fitur seperti Contract dan PaymentMethod bisa memberi sinyal kuat terhadap churn dan layak dipertahankan dengan one-hot encoding.

4.2.6 Outlier

4.2.6.1 Kode

```
datav.describe().T

plt.figure(figsize=(8, 4))

for idx, col in enumerate(num_col):
    plt.subplot(1, len(num_col), idx+1)
    sns.boxplot(y=datav[col])
    plt.title(col, fontsize=10)
    plt.tight_layout()

plt.show()

# Cek outlier pakai metode IQR
for col in num_col:
    Q1 = datav[col].quantile(0.25)
    Q3 = datav[col].quantile(0.75)
    IQR = Q3 - Q1

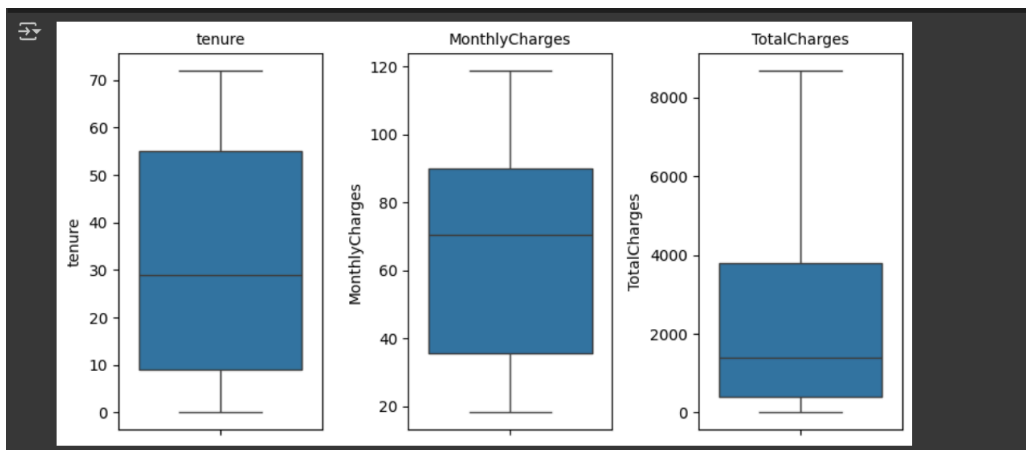
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers_iqr = datav[(datav[col] < lower_bound) | (datav[col] > upper_bound)]

    print(f'Jumlah outlier di {col} (IQR method): {outliers_iqr.shape[0]}')
```

4.2.6.2 Output

	count	mean	std	min	25%	50%	75%	max
tenure	7043.0	32.371149	24.559481	0.00	9.00	29.00	55.00	72.00
MonthlyCharges	7043.0	64.761692	30.090047	18.25	35.50	70.35	89.85	118.75
TotalCharges	7043.0	2279.734304	2266.794470	0.00	398.55	1394.55	3786.60	8684.80



```
print(f"Jumlah outlier di {col} (IQR method): {outliers_iqr.shape[0]}")
```

Jumlah outlier di tenure (IQR method): 0
 Jumlah outlier di MonthlyCharges (IQR method): 0
 Jumlah outlier di TotalCharges (IQR method): 0

4.2.6.3 Penjelasan

Dengan menggunakan metode IQR tidak didapati outlier untuk semua kolom numerik

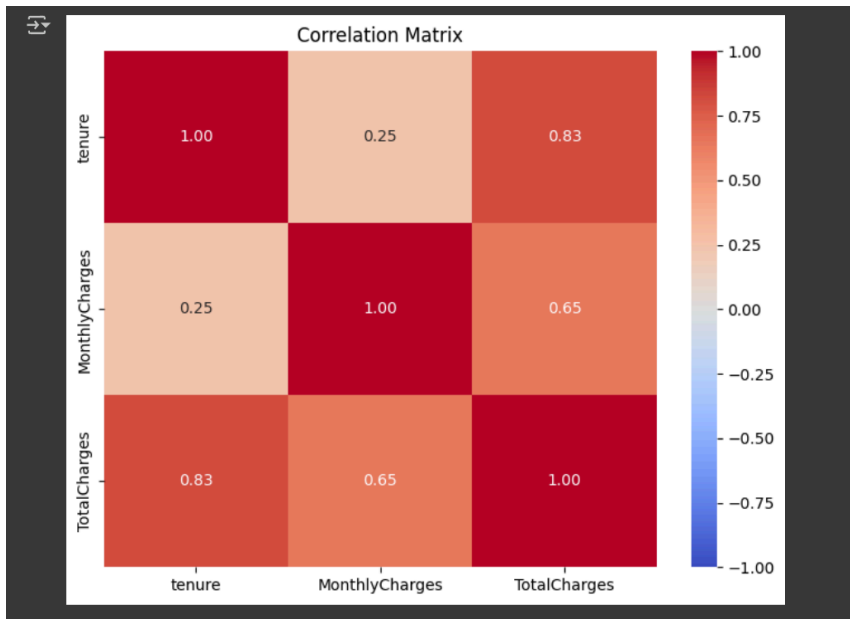
4.2.7 Correlation Matrix

4.2.7.1 Kode

```
# Menghitung korelasi
correlation_matrix = datav[num_col].corr()

# Membuat plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1,
            fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

4.2.7.2 Output



4.2.7.3 Penjelasan

Hasil Correlation Matrix tidak didapatkan kejanggalan atau insight menarik, semua korelasi logis, semakin lama nasabah berlangganan (tenure) maka semakin tinggi pula total tagihan suatu customer ('TotalCharges') begitu juga dengan total tagihan perbulan dari seorang nasabah ('MonthlyCharges') juga selaras dengan total tagihan nya selama terdaftar sebagai nasabah.

4.3 Data Preprocessing

4.3.1 Handling Duplicate

4.3.1.1 Kode

```
# copy data dan drop atribut customerID
data_final = datav.copy().drop('customerID', axis=1)

datav.duplicated().sum()

data_final.duplicated().sum()
```

4.3.1.2 Output

```
# copy data dan drop atribut customerID
data_final = datav.copy().drop(['customerID', axis=1])
```

Disini kita akan drop kolom customer id karena fitur ini tidak relevan untuk melakukan inferensi nantinya

```
[ ] datav.duplicated().sum()
np.int64(0)
```

▼ Handling Duplicate

```
[ ] data_final.duplicated().sum()
np.int64(22)
```

Ditemukan 22 records duplikat karena kita sudah menghapus kolom id, tapi kami menetapkan untuk membiarkan duplikat ini tetap ada

4.3.1.3 Penjelasan

Dilakukan drop kolom 'customerID' karena fitur ini tidak relevan untuk melakukan inferensi nantinya. Ditemukan 22 *records* duplikat karena kita sudah menghapus kolom id, tapi kami menetapkan untuk membiarkan duplikat ini tetap ada karena mereka adalah pelanggan berbeda dengan karakteristik yang sama, dan model perlu belajar dari semua kasus tersebut untuk menghasilkan prediksi akurat.

4.3.2 Encoding

4.3.2.1 Kode

```
# Encoding
from sklearn.preprocessing import LabelEncoder

# 1. Label Encoding untuk kolom binary
label_encode_cols = ['SeniorCitizen', 'gender', 'Partner', 'Dependents', 'PhoneService',
'PaperlessBilling', 'Churn']

le = LabelEncoder()
for col in label_encode_cols:
    data_final[col] = le.fit_transform(data_final[col])

# One-Hot Encoding untuk kolom kategori banyak pilihan
one_hot_encode_cols = [
    'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
    'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
    'Contract', 'PaymentMethod'
]
```



```
# Gunakan get_dummies dan konversi boolean ke integer (0/1)
data_final = pd.get_dummies(data_final, columns=one_hot_encode_cols, dtype=int)

data_final.info()
```

4.3.2.2 Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 41 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   gender                                     7043 non-null   int64
1   SeniorCitizen                             7043 non-null   int64
2   Partner                                   7043 non-null   int64
3   Dependents                               7043 non-null   int64
4   tenure                                    7043 non-null   int64
5   PhoneService                             7043 non-null   int64
6   PaperlessBilling                         7043 non-null   int64
7   MonthlyCharges                           7043 non-null   float64
8   TotalCharges                             7043 non-null   float64
9   Churn                                    7043 non-null   int64
10  MultipleLines_No                         7043 non-null   int64
11  MultipleLines_No phone service           7043 non-null   int64
12  MultipleLines_Yes                        7043 non-null   int64
13  InternetService_DSL                     7043 non-null   int64
14  InternetService_Fiber optic              7043 non-null   int64
15  InternetService_No                       7043 non-null   int64
16  OnlineSecurity_No                       7043 non-null   int64
17  OnlineSecurity_No internet service       7043 non-null   int64
18  OnlineSecurity_Yes                       7043 non-null   int64
19  OnlineBackup_No                         7043 non-null   int64
20  OnlineBackup_No internet service         7043 non-null   int64
21  OnlineBackup_Yes                        7043 non-null   int64
22  DeviceProtection_No                     7043 non-null   int64
23  DeviceProtection_No internet service     7043 non-null   int64
24  DeviceProtection_Yes                    7043 non-null   int64
25  TechSupport_No                          7043 non-null   int64
26  TechSupport_No internet service          7043 non-null   int64
27  TechSupport_Yes                         7043 non-null   int64
28  StreamingTV_No                          7043 non-null   int64
29  StreamingTV_No internet service          7043 non-null   int64
30  StreamingTV_Yes                         7043 non-null   int64
31  StreamingMovies_No                      7043 non-null   int64
32  StreamingMovies_No internet service      7043 non-null   int64
33  StreamingMovies_Yes                     7043 non-null   int64
34  Contract_Month-to-month                 7043 non-null   int64
35  Contract_One year                       7043 non-null   int64
36  Contract_Two year                       7043 non-null   int64
37  PaymentMethod_Bank transfer (automatic) 7043 non-null   int64
38  PaymentMethod_Credit card (automatic)   7043 non-null   int64
39  PaymentMethod_Electronic check          7043 non-null   int64
40  PaymentMethod_Mailed check              7043 non-null   int64
dtypes: float64(2), int64(39)
memory usage: 2.2 MB
```

4.3.3 Scaling

4.3.3.1 Kode

```
import pandas as pd
from sklearn.preprocessing import RobustScaler

# Misal df adalah DataFrame kamu
fitur_numerik = ['tenure', 'MonthlyCharges', 'TotalCharges']

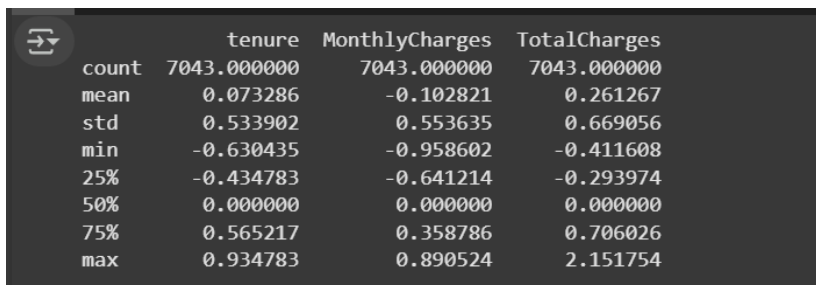
# Inisialisasi scaler
scaler = RobustScaler()
```

```
# Terapkan scaling hanya ke kolom numerik
data_final[fitur_numerik] = scaler.fit_transform(data_final[fitur_numerik])

# Jika ingin lihat hasilnya
print(data_final[fitur_numerik].describe())

# Menyimpan dataset yang sudah di lakukan preprocessing
data_final.to_csv('data_final.csv', index=False)
```

4.3.3.2 Output



	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000
mean	0.073286	-0.102821	0.261267
std	0.533902	0.553635	0.669056
min	-0.630435	-0.958602	-0.411608
25%	-0.434783	-0.641214	-0.293974
50%	0.000000	0.000000	0.000000
75%	0.565217	0.358786	0.706026
max	0.934783	0.890524	2.151754

4.3.3.3 Penjelasan

Berdasarkan visualisasi distribusi fitur numerik di bagian EDA, kolom seperti 'tenure', 'MonthlyCharges', dan 'TotalCharges', terlihat bahwa ketiganya memiliki distribusi yang tidak simetris (*skewed*) terutama pada 'TotalCharges' dan 'MonthlyCharges' yang memiliki ekor kanan panjang (*right-skewed*). Karena distribusi tidak simetris, penggunaan *RobustScaler* menjadi pilihan yang baik karena ia tidak bergantung pada *mean* dan standar deviasi, melainkan pada *median* dan IQR, sehingga lebih stabil dalam mentransformasi data yang tidak terdistribusi normal. Ini penting untuk membantu model seperti SVM dan *Decision Tree* bekerja lebih optimal, terutama dalam menjaga struktur relatif antar data yang mungkin tersebar tidak merata.

4.4 Train-Test Split

4.4.1 Kode

```
from sklearn.model_selection import train_test_split

# Split stratified berdasarkan kolom 'Churn'
data_latih, data_uji = train_test_split(
    data_final,
    test_size=0.2,
```

```
random_state=42,  
stratify=data_final["Churn"]  
)  
  
# Reset index  
data_latih.reset_index(drop=True, inplace=True)  
data_uji.reset_index(drop=True, inplace=True)
```

4.4.2 Penjelasan

Dilakukan *stratify train test split* karena adanya perbedaan distribusi yang cukup jauh antara *records churn* dan *non-churn* agar proses *training* dan evaluasi mendapatkan hasil yang seimbang diantara kedua kelas tersebut.

4.5 Decision Tree

4.5.1 Kode

```
# Hitung impurity Gini  
def hitung_gini(kolom_kelas):  
    elemen, banyak = np.unique(kolom_kelas, return_counts=True)  
    nilai_gini = 1 - np.sum([(banyak[i] / np.sum(banyak)) ** 2 for i in range(len(elemen))])  
    return nilai_gini  
  
# Hitung nilai Gini split untuk fitur kategorikal  
def gini_split_kategorikal(data, nama_fitur_split, nama_fitur_kelas):  
    nilai, banyak = np.unique(data[nama_fitur_split], return_counts=True)  
    gini_split = np.sum([  
        (banyak[i] / np.sum(banyak)) * hitung_gini(  
            data[data[nama_fitur_split] == nilai[i]][nama_fitur_kelas]  
        )  
        for i in range(len(banyak))  
    ])  
    return gini_split  
  
# Hitung nilai Gini split untuk fitur numerik  
def gini_split_numerik(data, nama_fitur_split, nama_fitur_kelas, threshold):  
    data_kiri = data[data[nama_fitur_split] <= threshold]  
    data_kanan = data[data[nama_fitur_split] > threshold]  
    gini_kiri = hitung_gini(data_kiri[nama_fitur_kelas])  
    gini_kanan = hitung_gini(data_kanan[nama_fitur_kelas])
```

```
    gini_split = (len(data_kiri) / len(data)) * gini_kiri + (len(data_kanan) / len(data)) *
    gini_kanan
    return gini_split

# Fungsi untuk mencari threshold terbaik untuk fitur numerik
def cari_split_numerik_terbaik(data, nama_fitur_split, nama_fitur_kelas):
    nilai_unik = np.unique(data[nama_fitur_split])
    threshold_coba = (nilai_unik[:-1] + nilai_unik[1:]) / 2 # Ambil tengah-tengah
    gini_splits = [gini_split_numerik(data, nama_fitur_split, nama_fitur_kelas, t) for t in
    threshold_coba]
    if len(gini_splits) == 0:
        return None, float('inf')
    index_terbaik = np.argmin(gini_splits)
    return threshold_coba[index_terbaik], gini_splits[index_terbaik]

# Memperbaiki fungsi pembuatan pohon keputusan
def buat_tree(data, data_awal, daftar_fitur, nama_fitur_kelas, kelas_parent_node=None):
    # Kasus 1: Data sudah homogen
    if len(data) > 0 and len(np.unique(data[nama_fitur_kelas])) <= 1:
        # Pastikan mengembalikan nilai integer (0 atau 1)
        return int(np.unique(data[nama_fitur_kelas])[0])

    # Kasus 2: Data kosong - ambil mayoritas dari data awal
    elif len(data) == 0:
        nilai, jumlah = np.unique(data_awal[nama_fitur_kelas], return_counts=True)
        # Pastikan mengembalikan nilai integer (0 atau 1)
        return int(nilai[np.argmax(jumlah)])

    # Kasus 3: Tidak ada fitur yang tersisa
    elif len(daftar_fitur) == 0:
        # Jika parent node tidak ada, ambil mayoritas dari data saat ini
        if kelas_parent_node is None:
            nilai, jumlah = np.unique(data[nama_fitur_kelas], return_counts=True)
            return int(nilai[np.argmax(jumlah)])
        # Jika ada parent node, gunakan nilai parent
        else:
            return int(kelas_parent_node)

    # Kasus normal: Pilih fitur terbaik dan buat split
    else:
        # Tentukan kelas mayoritas di node ini untuk digunakan sebagai nilai default
```

```
nilai, jumlah = np.unique(data[nama_fitur_kelas], return_counts=True)
kelas_parent_node = int(nilai[np.argmax(jumlah)])

# Pilih split terbaik
best_splits = []
for fitur in daftar_fitur:
    # Cek apakah fitur numerik atau kategorikal
    if (data[fitur].dtype in ['int64', 'float64', 'int32', 'float32']):
        threshold, gini = cari_split_numerik_terbaik(data, fitur, nama_fitur_kelas)
        if threshold is not None: # Pastikan threshold valid
            best_splits.append((fitur, gini, threshold))
    else:
        gini = gini_split_kategorikal(data, fitur, nama_fitur_kelas)
        best_splits.append((fitur, gini, None))

# Jika tidak ada split yang valid
if len(best_splits) == 0:
    return kelas_parent_node

# Cari fitur dengan Gini terkecil
best_splits.sort(key=lambda x: x[1])
fitur_terbaik, gini_terbaik, threshold_terbaik = best_splits[0]

tree = {fitur_terbaik: {}}
daftar_fitur = [i for i in daftar_fitur if i != fitur_terbaik]

if threshold_terbaik is not None: # fitur numerik
    data_kiri = data[data[fitur_terbaik] <= threshold_terbaik]
    data_kanan = data[data[fitur_terbaik] > threshold_terbaik]

    # Gunakan string yang konsisten untuk kondisi
    kondisi_kiri = f"<= {threshold_terbaik:.5f}"
    kondisi_kanan = f"> {threshold_terbaik:.5f}"

    tree[fitur_terbaik][kondisi_kiri] = buat_tree(
        data_kiri, data_awal, daftar_fitur, nama_fitur_kelas, kelas_parent_node
    )
    tree[fitur_terbaik][kondisi_kanan] = buat_tree(
        data_kanan, data_awal, daftar_fitur, nama_fitur_kelas, kelas_parent_node
    )
```

```
else: # fitur kategorikal
    # Simpan semua nilai kategorikal yang ada
    nilai_kategori = np.unique(data_awal[fitur_terbaik])

    # Proses setiap nilai kategori
    for nilai in nilai_kategori:
        sub_data = data[data[fitur_terbaik] == nilai]
        # Jika sub_data kosong, gunakan kelas mayoritas dari parent
        if len(sub_data) == 0:
            tree[fitur_terbaik][int(nilai)] = kelas_parent_node
        else:
            # Pastikan nilai kategori disimpan dengan tipe data yang konsisten (int)
            tree[fitur_terbaik][int(nilai)] = buat_tree(
                sub_data, data_awal, daftar_fitur, nama_fitur_kelas, kelas_parent_node
            )

    return tree

fitur_input = list(data_latih.columns)
fitur_input.remove("Churn") # Hapus label dari daftar fitur

tree = buat_tree(data_latih, data_latih, fitur_input, "Churn")
tree = buat_tree(data_latih, data_latih, fitur_input, "Churn")

# Fungsi prediksi yang sederhana
def prediksi(data_uji, tree):
    # Jika tree bukan dictionary, berarti sudah mencapai leaf node
    if not isinstance(tree, dict):
        return int(tree) # Pastikan mengembalikan integer

    # Ambil fitur yang digunakan di node ini
    fitur = list(tree.keys())[0]

    # Jika fitur tidak ada dalam data uji, gunakan prediksi default
    if fitur not in data_uji:
        # Ambil nilai pertama (default) dari sub-tree
        first_value = list(tree[fitur].values())[0]
        if isinstance(first_value, dict):
            return prediksi(data_uji, first_value)
        else:
```

```
        return int(first_value)

# Nilai fitur dari data uji
nilai_fitur = data_uji[fitur]

# Periksa jenis node (numerik atau kategorikal)
for kondisi, subtree in tree[fitur].items():
    # Jika kondisi berupa string dengan operator perbandingan (untuk fitur numerik)
    if isinstance(kondisi, str):
        if "<=" in kondisi:
            threshold = float(kondisi.split('<=')[1])
            if nilai_fitur <= threshold:
                return prediksi(data_uji, subtree)
        elif ">" in kondisi:
            threshold = float(kondisi.split('>')[1])
            if nilai_fitur > threshold:
                return prediksi(data_uji, subtree)
    # Jika kondisi berupa nilai langsung (untuk fitur kategorikal)
    elif kondisi == nilai_fitur:
        return prediksi(data_uji, subtree)

# Jika tidak ada kondisi yang terpenuhi, ambil nilai default (subtree pertama)
first_value = list(tree[fitur].values())[0]
if isinstance(first_value, dict):
    return prediksi(data_uji, first_value)
else:
    return int(first_value)

data_uji_dict = data_uji.iloc[:, :-1].to_dict(orient="records")

hasil_prediksi_dt = []
for i in range(len(data_uji_dict)):
    hasil_prediksi = (prediksi(data_uji_dict[i], tree))
    hasil_prediksi_dt.append(hasil_prediksi)
```

4.5.2 Penjelasan

Fungsi `hitung_gini` menerima nilai kelas pada variabel tertentu dengan nilai tertentu. Nilai dan frekuensi dari masing-masing kelas dihitung menggunakan fungsi `np.unique` dari *library numpy*, kemudian dilakukan kalkulasi nilai GINI.

Fungsi `gini_split_kategorikal` digunakan untuk menghitung nilai gini split dari sebuah fitur kategorikal terhadap label kelas, yang berguna dalam pemilihan fitur terbaik saat membangun pohon keputusan (*decision tree*). Fungsi ini bekerja dengan terlebih dahulu mencari semua nilai unik dari fitur kategorikal (`nama_fitur_split`) beserta jumlah kemunculannya. Kemudian, untuk setiap nilai unik tersebut, fungsi menghitung nilai gini dari *subset* data yang memiliki nilai tersebut, lalu mengalikannya dengan proporsi jumlah data subset tersebut terhadap keseluruhan data. Hasil akhir dari fungsi ini adalah nilai gini split, yaitu rata-rata tertimbang dari gini impurity untuk setiap subset, yang menunjukkan seberapa baik fitur tersebut dalam membagi data berdasarkan kelas. Nilai ini digunakan untuk membandingkan fitur mana yang paling informatif saat membuat percabangan dalam *decision tree*.

Fungsi `gini_split_numerik` digunakan untuk menghitung nilai gini split pada fitur numerik dengan menggunakan suatu nilai ambang batas (*threshold*) sebagai titik pemisah. Fungsi ini bekerja dengan membagi data menjadi dua *subset*, yaitu data yang nilai fiturnya kurang dari atau sama dengan *threshold* (`data_kiri`) dan data yang lebih besar dari *threshold* (`data_kanan`). Kemudian, fungsi menghitung nilai gini impurity untuk masing-masing subset tersebut, dan menggabungkannya dengan cara rata-rata tertimbang berdasarkan proporsi jumlah data di masing-masing subset. Hasil akhirnya adalah nilai gini split untuk *threshold* tersebut, yang menggambarkan seberapa baik *threshold* itu memisahkan data berdasarkan kelas, dan bisa digunakan untuk mencari *threshold* optimal dalam pembentukan *decision tree*.

Fungsi `cari_split_numerik_terbaik` digunakan untuk menentukan nilai *threshold* terbaik pada fitur numerik yang menghasilkan nilai gini split paling kecil, yang berarti pemisahan data paling bersih berdasarkan kelas. Fungsi ini bekerja dengan terlebih dahulu mencari semua nilai unik dari fitur numerik yang diberikan, kemudian menghitung titik-titik tengah (*midpoints*) di antara pasangan nilai unik tersebut sebagai kandidat *threshold*. Untuk setiap kandidat *threshold*, dihitung nilai gini split-nya menggunakan fungsi `gini_split_numerik`. Jika tidak ada *threshold* yang bisa dihitung (misalnya karena hanya ada satu nilai unik), fungsi akan mengembalikan *None* dan *inf* sebagai nilai *default*. Jika ada, fungsi mengembalikan *threshold* dengan nilai gini split terkecil, yang bisa digunakan untuk membuat keputusan terbaik saat membagi node pada pohon keputusan.

Fungsi `buat_tree` merupakan inti dari algoritma *decision tree* buatan sendiri yang bekerja secara rekursif untuk membentuk struktur pohon klasifikasi dari data yang diberikan. Fungsi ini menangani berbagai kondisi khusus, seperti data yang sudah homogen (semua nilai target sama), data kosong, atau tidak ada fitur yang tersisa. Dalam kasus-kasus tersebut, fungsi akan mengembalikan nilai kelas mayoritas (baik dari data awal maupun data saat ini) agar tetap bisa menghasilkan keputusan. Selain itu, fungsi juga menetapkan nilai kelas mayoritas pada setiap node sebagai fallback jika pada proses split tidak ditemukan pembagian yang valid.

Pada kasus umum, fungsi akan memilih fitur terbaik berdasarkan nilai gini split terkecil, baik dari fitur numerik maupun kategorikal. Untuk fitur numerik, digunakan *threshold* optimal untuk membagi data menjadi dua *subset*; sementara untuk fitur kategorikal, data dibagi berdasarkan setiap nilai kategorinya. Setelah pembagian, fungsi akan memanggil dirinya sendiri secara rekursif untuk masing-masing *subset* hingga kondisi berhenti tercapai. Struktur pohon akhir disimpan dalam bentuk *nested dictionary*, yang merepresentasikan jalur keputusan dari akar hingga ke daun pohon.

Pendekatan ini mendekati logika pembentukan pohon pada algoritma seperti CART (Classification and Regression Trees), namun dibuat secara manual dan eksplisit agar lebih mudah dipahami.

Fungsi prediksi digunakan untuk melakukan klasifikasi terhadap satu instance data uji berdasarkan struktur decision tree yang telah dibentuk sebelumnya. Fungsi ini bekerja secara rekursif dengan memeriksa fitur yang digunakan pada setiap node dalam pohon. Jika *node* yang diakses sudah berupa nilai (*leaf node*), maka fungsi akan langsung mengembalikannya sebagai hasil prediksi. Untuk fitur numerik, fungsi akan membandingkan nilai fitur terhadap *threshold* menggunakan operator \leq atau $>$ yang terenkapsulasi dalam string kondisi; sedangkan untuk fitur kategorikal, fungsi akan mencocokkan nilai secara langsung. Jika fitur tidak ditemukan dalam data uji atau tidak ada kondisi yang cocok, fungsi akan memilih salah satu *subtree* sebagai default untuk menjaga agar proses prediksi tetap berjalan. Pendekatan ini membuat fungsi cukup tangguh dalam menghadapi berbagai struktur pohon dan variasi data uji. y_{pred} dari *decision tree* yang sudah dibuat selanjutnya akan disimpan di variabel `hasil_prediksi_dt` dalam bentuk list

4.6 SVM

4.6.1 Kode

```
label_latih = data_latih['Churn']
label_uji = data_uji['Churn']

def buat_trainingset(dataset):
    trainingset = {}
    kolom_kelas = dataset.columns[-1]
    list_kelas = dataset[kolom_kelas].unique()
    for kelas in list_kelas:
        data_temp = dataset.copy(deep=True)
        data_temp[kolom_kelas] = data_temp[kolom_kelas].map({kelas:1})
        data_temp[kolom_kelas] = data_temp[kolom_kelas].fillna(-1)
        trainingset[kelas] = data_temp
    return trainingset

trainingset = buat_trainingset(data_latih)

def hitung_cost_gradient(W,X,Y, regularization):
    jarak = 1 - (Y * np.dot(X,W))
    dw = np.zeros(len(W))
    if max(0,jarak) == 0:
        di = W
    else:
        di = W - (regularization * Y * X)
    dw += di
```

```
return dw

from sklearn.utils import shuffle

def sgd(data_latih, label_latih, learning_rate=0.000001, max_epoch=1000,
        regularization=10000):
    data_latih = data_latih.to_numpy()
    label_latih = label_latih.to_numpy()
    bobot = np.zeros(data_latih.shape[1])

    for epoch in range(1, max_epoch):
        X,Y = shuffle(data_latih, label_latih, random_state=101)
        for index, x in enumerate(X):
            delta = hitung_cost_gradient(bobot, x, Y[index], regularization)
            bobot = bobot - (learning_rate * delta)
    return bobot

def training(trainingset):
    list_kelas = trainingset.keys()
    w = {}
    for kelas in list_kelas:
        data_latih = trainingset[kelas]
        label_latih = data_latih.pop(data_latih.columns[-1])
        w[kelas] = sgd(data_latih, label_latih)

    return w

W = training(trainingset)

def testing(W, data_uji):
    prediksi = np.array([]).astype(int)
    # Drop kolom target (jika masih ada)
    if 'Churn' in data_uji.columns:
        data_uji = data_uji.drop(columns='Churn')

    data_uji_np = data_uji.to_numpy()

    for i in range(data_uji_np.shape[0]):
        skor = {}
        for kelas in W:
            skor[kelas] = np.dot(W[kelas], data_uji_np[i])
```

```
prediksi = np.append(prediksi, max(skor, key=skor.get)) # kelas dengan skor tertinggi

return prediksi

hasil_prediksi_svm = testing(W, data_uji)
```

4.6.2 Penjelasan

Fungsi `buat_trainingset` bertujuan untuk memodifikasi *dataset* sehingga menghasilkan *dictionary* trainingset yang dapat digunakan dalam proses pelatihan model. Fungsi ini pertama-tama mengambil kolom kelas dari *dataset* dan memperoleh daftar kelas unik. Selanjutnya, untuk setiap kelas, fungsi ini menyalin dataset dan memodifikasi nilai pada kolom kelas: kelas yang sesuai dengan *key* akan diberi nilai 1, sementara kelas lainnya akan diberi nilai -1. *Dictionary* trainingset ini kemudian menyimpan hasil modifikasi dataset untuk setiap kelas, yang digunakan sebagai data latih dalam proses pelatihan model. Fungsi ini akan menghasilkan dictionary dengan *key* berupa kelas dan *value* berupa dataset yang sudah dimodifikasi sesuai dengan kelas tersebut.

Fungsi `hitung_cost_gradient` digunakan untuk menghitung nilai gradien dari cost function dalam algoritma Support Vector Machine (SVM). Tujuan dari algoritma SVM adalah meminimalkan cost function, dan untuk itu diperlukan perhitungan gradien untuk memperbarui parameter model. Fungsi ini pertama-tama menghitung jarak antara data dan margin dengan rumus $1 - (Y * \text{np.dot}(X, W))$. Jika jarak tersebut lebih besar dari 0, maka gradien diperbarui dengan mengurangi nilai regularisasi yang disesuaikan dengan data dan bobot. Sebaliknya, jika jarak sama dengan atau lebih kecil dari 0, gradien akan tetap sama dengan bobot *W*. Fungsi ini mengembalikan gradien yang akan digunakan untuk memperbarui parameter model dalam proses optimasi.

Fungsi `sgd` merupakan implementasi dari algoritma *Stochastic Gradient Descent* (SGD) yang digunakan untuk melakukan optimasi pada model dengan meminimalkan *cost function*. Fungsi ini menerima *input* berupa data latih (`data_latih`), label latih (`label_latih`), laju pembelajaran (`learning_rate`), jumlah iterasi maksimum (`max_epoch`), dan parameter regularisasi. Dalam setiap epoch, data dan label akan di-shuffle menggunakan shuffle dari `sklearn.utils` untuk memastikan bahwa model tidak terjebak pada urutan data tertentu. Kemudian, untuk setiap data individu, gradien dihitung menggunakan fungsi `hitung_cost_gradient`, dan bobot model diperbarui dengan mengurangi nilai gradien yang telah dihitung dikalikan dengan laju pembelajaran. Proses ini diulang selama jumlah *epoch* yang ditentukan. Hasil akhirnya adalah bobot model yang teroptimasi yang akan digunakan untuk prediksi.

Fungsi `training` digunakan untuk melatih model pada setiap kelas yang ada dalam trainingset. Fungsi ini pertama-tama mengambil daftar kelas yang terdapat dalam trainingset dan kemudian untuk setiap kelas, mengambil dataset yang terkait. Label latih diambil dari kolom terakhir dataset dan kemudian dihapus dari data latih. Selanjutnya, fungsi `sgd` dipanggil untuk menghitung bobot model dengan data latih dan label latih yang telah dipisahkan, dan hasil bobot untuk setiap kelas disimpan dalam *dictionary* *w*. Setelah proses pelatihan selesai untuk semua kelas, *dictionary* *w* yang

berisi bobot model untuk setiap kelas dikembalikan. Hasil akhir dari fungsi ini adalah *dictionary* W, yang berisi bobot model yang sudah dilatih untuk setiap kelas dalam trainingset.

Fungsi testing digunakan untuk melakukan prediksi pada data uji berdasarkan bobot model yang telah dilatih sebelumnya. Fungsi ini pertama-tama memeriksa apakah ada kolom target yang bernama 'Churn' dalam data_uji dan jika ada, kolom tersebut dihapus. Kemudian, data uji diubah menjadi array numpy untuk memudahkan perhitungan. Untuk setiap baris data uji, fungsi menghitung skor untuk setiap kelas dengan cara mengalikan bobot kelas yang relevan dengan data uji tersebut menggunakan operasi dot product (`np.dot`). Kelas dengan skor tertinggi akan dipilih sebagai prediksi untuk baris data tersebut. Prediksi untuk semua data uji akan disimpan dalam array prediksi, yang kemudian dikembalikan sebagai hasil akhir. Hasil dari fungsi ini adalah array yang berisi prediksi kelas untuk setiap data uji berdasarkan model yang sudah dilatih.

4.7 Random Forest

4.7.1 Kode

```
! wget https://raw.githubusercontent.com/Muzann11/KA-lanjutan/main/data\_final.csv

data_final = pd.read_csv('data_final.csv')

from sklearn.model_selection import train_test_split

# Split stratified berdasarkan kolom 'Churn'
data_latih, data_uji = train_test_split(
    data_final,
    test_size=0.2,
    random_state=42,
    stratify=data_final["Churn"]
)

# Reset index
data_latih.reset_index(drop=True, inplace=True)
data_uji.reset_index(drop=True, inplace=True)

X_train = data_latih.drop(columns=["Churn"])
y_train = data_latih["Churn"]
X_test = data_uji.drop(columns=["Churn"])
y_test = data_uji["Churn"]

from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
rf_model.fit(X_train, y_train)
hasil_prediksi_rf = rf_model.predict(X_test)
```

4.8 Evaluasi

Pada tahap ini, dibuat fungsi-fungsi untuk melakukan evaluasi seperti *confusion matrix*, presisi, akurasi, dan sebagainya. Kemudian fungsi-fungsi tersebut akan diimplementasikan pada model yang telah dibuat. Fungsinya untuk mengevaluasi seberapa baik model tersebut dalam memprediksi data baru.

4.8.1 Fungsi-Fungsi Untuk Evaluasi

4.8.1.1 Kode

```
def confusion_matrix_manual(y_true, y_pred, labels):
    num_classes = len(labels)

    # Inisialisasi matriks dengan ukuran (jumlah class)
    confusion_matrix = np.zeros((num_classes, num_classes), dtype=int)

    # Mengisi matriks
    for true, pred in zip(y_true, y_pred):
        confusion_matrix[true, pred] += 1

    return confusion_matrix

def calculate_metrics (confusion_matrix, labels):
    num_classes = confusion_matrix.shape[0]
    metrics = {}
    for i in range(num_classes):
        TP = confusion_matrix[i, i]
        FP = confusion_matrix[:, i].sum() - TP
        FN = confusion_matrix[i, :].sum() - TP
        TN = confusion_matrix.sum() - (TP + FP+ FN)
        metrics[labels[i]] = {
            "TP": TP,
            "FP": FP,
            "FN": FN,
            "TN": TN
        }
    return metrics
```

```
def normalize_confusion_matrix(cm):
    """
    Fungsi untuk melakukan normalisasi pada hasil confusion matrix dengan rentang 0-1

    parameter :
        cm : confusion matrix yang akan dinormalisasi
    return
        cm_normalized : confusion matrix yang sudah dinormalisasi
    """
    cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    return cm_normalized

def calculate_precision(metrics):
    """
    Menghitung precision per kelas dan rata-rata precision.

    Parameter:
        metrics : dict hasil dari calculate_metrics()

    Return:
        precision_per_class : dict {label: precision}
        avg_precision : float
    """
    precision_per_class = {}
    total_precision = 0
    num_classes = len(metrics)

    for label, metric in metrics.items():
        TP = metric["TP"]
        FP = metric["FP"]
        precision = TP / (TP + FP) if (TP + FP) > 0 else 0
        precision_per_class[label] = precision
        total_precision += precision

    avg_precision = total_precision / num_classes
    return precision_per_class, avg_precision

def calculate_recall(metrics):
    """
```

Menghitung recall per kelas dan rata-rata recall.

Parameter:

metrics : dict hasil dari calculate_metrics()

Return:

recall_per_class : dict {label: recall}

avg_recall : float

"""

recall_per_class = {}

total_recall = 0

num_classes = len(metrics)

for label, metric in metrics.items():

TP = metric["TP"]

FN = metric["FN"]

recall = TP / (TP + FN) if (TP + FN) > 0 else 0

recall_per_class[label] = recall

total_recall += recall

avg_recall = total_recall / num_classes

return recall_per_class, avg_recall

def calculate_f1_score(metrics):

"""

Menghitung f1-score per kelas dan rata-rata f1-score.

Parameter:

metrics : dict hasil dari calculate_metrics()

Return:

f1_per_class : dict {label: f1-score}

avg_f1 : float

"""

f1_per_class = {}

total_f1 = 0

num_classes = len(metrics)

for label, metric in metrics.items():

```

    TP = metric["TP"]
    FP = metric["FP"]
    FN = metric["FN"]
    precision = TP / (TP + FP) if (TP + FP) > 0 else 0
    recall = TP / (TP + FN) if (TP + FN) > 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
    f1_per_class[label] = f1
    total_f1 += f1

avg_f1 = total_f1 / num_classes
return f1_per_class, avg_f1

def calculate_accuracy_score(metrics):
    """
    Fungsi untuk menghitung accuracy_score

    parameter :
        metrics : hasil fungsi calculate_metrics
    return
        accuracy : nilai accuracy_score
    """
    total_accuracy = 0
    num_classes = len(metrics)

    for label, metric in metrics.items():
        accuracy = (metric["TP"] + metric["TN"]) / (metric["TP"] + metric["FP"] + metric["FN"] +
metric["TN"])
        total_accuracy += accuracy

    avg_accuracy = total_accuracy / num_classes
    return avg_accuracy

def cm_plot(cm, true_labels, predictions):
    """
    Menampilkan visualisasi Confusion Matrix dalam bentuk heatmap.

    Parameter:
    - cm : array-like
        Confusion matrix hasil prediksi model (bisa dinormalisasi atau tidak).
    - true_labels : array-like

```


Label sebenarnya (ground truth) dari data.

- predictions : array-like

Label hasil prediksi dari model.

- labels : array-like

Daftar semua label unik pada kelas target.

Output:

- Menampilkan plot heatmap Confusion Matrix menggunakan seaborn.

"""

```
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, cmap="Blues", xticklabels=np.unique(true_labels),
yticklabels=np.unique(true_labels))
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

4.8.1.2 Penjelasan

Fungsi `confusion_matrix_manual` digunakan untuk menghitung *confusion matrix* secara manual, yang menunjukkan performa prediksi model dengan membandingkan nilai prediksi (`y_pred`) dan nilai sebenarnya (`y_true`) berdasarkan kelas yang telah ditentukan. Fungsi ini menerima tiga parameter: `y_true` (nilai kelas yang sebenarnya), `y_pred` (nilai kelas yang diprediksi), dan `labels` (daftar kelas yang mungkin). Pertama, fungsi menginisialisasi *confusion matrix* berukuran (jumlah kelas x jumlah kelas) dengan nilai nol. Kemudian, untuk setiap pasangan nilai true dan pred, nilai pada *confusion matrix* akan diperbarui dengan menambah 1 pada posisi yang sesuai, di mana baris menunjukkan kelas yang benar dan kolom menunjukkan kelas yang diprediksi. Matriks yang dihasilkan akan menggambarkan berapa kali setiap kombinasi kelas yang benar dan yang diprediksi terjadi. Fungsi ini mengembalikan *confusion matrix* yang dapat digunakan untuk menganalisis performa model.

Fungsi `calculate_metrics` digunakan untuk menghitung metrik evaluasi dari *confusion matrix* (confusion matrix), seperti True Positive (TP), False Positive (FP), False Negative (FN), dan True Negative (TN) untuk setiap kelas. Fungsi ini menerima dua parameter: `confusion_matrix` (*confusion matrix* yang dihitung sebelumnya) dan `labels` (daftar kelas yang digunakan dalam perhitungan).

Fungsi ini pertama-tama menginisialisasi sebuah dictionary metrics untuk menyimpan nilai-nilai metrik untuk setiap kelas. Kemudian, untuk setiap kelas (ditandai dengan indeks `i`), fungsi menghitung:

- TP (*True Positive*): Jumlah prediksi yang benar untuk kelas tersebut.

- FP (*False Positive*): Jumlah data yang salah dikategorikan sebagai kelas tersebut, dihitung dengan menjumlahkan seluruh kolom pada *confusion matrix* untuk kelas tersebut, kecuali TP.
- FN (*False Negative*): Jumlah data yang benar dari kelas tersebut tetapi salah diprediksi sebagai kelas lain, dihitung dengan menjumlahkan seluruh baris pada *confusion matrix* untuk kelas tersebut, kecuali TP.
- TN (*True Negative*): Jumlah data yang tidak termasuk dalam kelas tersebut dan diprediksi dengan benar sebagai kelas selain kelas yang diuji, dihitung dengan mengurangi jumlah TP, FP, dan FN dari total jumlah elemen dalam matriks.

Fungsi ini akan mengembalikan sebuah *dictionary metrics* yang menyimpan metrik untuk setiap kelas, yang dapat digunakan untuk menilai performa model dalam setiap kelas.

Fungsi `normalize_confusion_matrix` digunakan untuk menormalisasi *confusion matrix* dengan membagi setiap elemen matriks dengan jumlah elemen pada baris yang sesuai, menghasilkan nilai dalam rentang 0 hingga 1. Ini membantu dalam mengevaluasi performa model dengan lebih adil, mengingat perbedaan jumlah data di setiap kelas. Fungsi ini mengembalikan *confusion matrix* yang sudah dinormalisasi.

Fungsi `calculate_precision` digunakan untuk menghitung nilai *precision* per kelas serta rata-rata *precision* untuk semua kelas berdasarkan metrik yang dihitung sebelumnya. Fungsi ini menerima input berupa *dictionary metrics* yang dihasilkan dari fungsi `calculate_metrics()`, yang berisi nilai TP dan FP untuk setiap kelas. *Precision* untuk setiap kelas dihitung dengan rumus $TP / (TP + FP)$, dan jika jumlah $TP + FP$ sama dengan nol, *precision* diatur menjadi nol. Nilai *precision* per kelas disimpan dalam *dictionary precision_per_class*, sementara rata-rata *precision* dihitung dengan menjumlahkan semua *precision* dan membaginya dengan jumlah kelas. Fungsi ini mengembalikan *dictionary precision* per kelas serta rata-rata *precision* untuk seluruh kelas.

Fungsi `calculate_recall` digunakan untuk menghitung nilai *recall* per kelas serta rata-rata *recall* untuk semua kelas berdasarkan metrik yang dihitung sebelumnya. Fungsi ini menerima input berupa *dictionary metrics* yang dihasilkan dari fungsi `calculate_metrics()`, yang berisi nilai TP dan FN untuk setiap kelas. *Recall* dihitung dengan rumus $TP / (TP + FN)$, dan jika jumlah $TP + FN$ sama dengan nol, *recall* diatur menjadi nol. Nilai *recall* per kelas disimpan dalam *dictionary recall_per_class*, sementara rata-rata *recall* dihitung dengan menjumlahkan semua *recall* dan membaginya dengan jumlah kelas. Fungsi ini mengembalikan *dictionary recall* per kelas serta rata-rata *recall* untuk seluruh kelas.

Fungsi `calculate_f1_score` digunakan untuk menghitung nilai F1-score per kelas serta rata-rata F1-score untuk seluruh kelas berdasarkan metrik yang dihitung sebelumnya. Fungsi ini menerima input berupa *dictionary metrics* yang dihasilkan dari fungsi `calculate_metrics()`, yang berisi nilai TP, FP, dan FN untuk setiap kelas. F1-score dihitung dengan rumus $2 * (precision * recall) / (precision + recall)$, dengan *precision* dan *recall* dihitung terlebih dahulu. Jika jumlah *precision* dan *recall* adalah nol, F1-score diatur menjadi nol. Nilai F1-score per kelas disimpan dalam *dictionary f1_per_class*, sementara rata-rata F1-score dihitung dengan menjumlahkan semua F1-score dan membaginya dengan jumlah kelas. Fungsi ini mengembalikan *dictionary F1-score* per kelas serta rata-rata F1-score untuk seluruh kelas.

Fungsi `calculate_accuracy_score` digunakan untuk menghitung nilai *accuracy score* rata-rata berdasarkan metrik yang dihitung sebelumnya. Fungsi ini menerima input berupa *dictionary metrics*, yang berisi nilai TP, FP, FN, dan TN untuk setiap kelas. Untuk setiap kelas, *accuracy* dihitung dengan rumus $(TP + TN) / (TP + FP + FN + TN)$, yang menunjukkan proporsi prediksi yang benar (baik positif maupun negatif) dibandingkan dengan total prediksi. Nilai *accuracy* per kelas dijumlahkan, dan rata-rata *accuracy* dihitung dengan membaginya dengan jumlah kelas. Fungsi ini mengembalikan rata-rata *accuracy* untuk seluruh kelas.

Fungsi `cm_plot` digunakan untuk menampilkan visualisasi *confusion matrix* dalam bentuk heatmap menggunakan `seaborn`. Fungsi ini menerima tiga parameter utama: `cm` yang berisi *confusion matrix* hasil prediksi model (bisa dalam bentuk normalisasi atau tidak), `true_labels` yang merupakan label sebenarnya dari data, dan `predictions` yang merupakan hasil prediksi model. Fungsi ini juga menggunakan `sns.heatmap` untuk memplot *confusion matrix* dengan anotasi untuk setiap elemen, serta menampilkan label pada sumbu X dan Y berdasarkan kelas target yang unik. Outputnya adalah visualisasi heatmap yang menggambarkan hubungan antara label yang sebenarnya dan hasil prediksi.

4.8.2 Persiapan Variabel

4.8.2.1 Kode

```
# Label asli
y_true = data_uji["Churn"].tolist()

# Label prediksi Decision Tree
y_pred_dt = hasil_prediksi_dt

# Label prediksi Decision Tree
y_pred_svm = hasil_prediksi_svm

# Label prediksi Random Forest
y_pred_rf = hasil_prediksi_rf

# Pastikan label sudah diketahui (misalnya dari data latih)
labels = np.unique(data_latih["Churn"]).tolist()

# Hitung confusion matrix
conf_matrix_dt = confusion_matrix_manual(y_true, y_pred_dt, labels)

conf_matrix_svm = confusion_matrix_manual(y_true, y_pred_svm, labels)

conf_matrix_rf = confusion_matrix_manual(y_true, y_pred_rf, labels)
```

```
# Hitung metrik evaluasi
metrics_dt = calculate_metrics(conf_matrix_dt, labels)

metrics_svm = calculate_metrics(conf_matrix_svm, labels)

metrics_rf = calculate_metrics(conf_matrix_rf, labels)
```

4.8.3 Evaluasi *Decision Tree*

4.8.3.1 Kode

```
precision_dt, avg_precision_dt = calculate_precision(metrics_dt)
recall_dt, avg_recall_dt = calculate_recall(metrics_dt)
f1_dt, avg_f1_dt = calculate_f1_score(metrics_dt)
accuracy_dt = calculate_accuracy_score(metrics_dt)

print("Accuracy :", accuracy_dt)
print("Precision per kelas:", precision_dt)
print("Recall per kelas:", recall_dt)
print("F1-score per kelas:", f1_dt)
print("\nRata-rata: ")
print(f"Precision:      {avg_precision_dt:.2f},      Recall:      {avg_recall_dt:.2f},      F1: {avg_f1_dt:.2f}")

# Cetak hasil
print("Confusion Matrix DT:")
print(conf_matrix_dt)

norm_cm_dt = normalize_confusion_matrix(conf_matrix_dt)

cm_plot(norm_cm_dt, y_true, y_pred_dt)

#Perbandingan dengan DecisionTreeClassifier dari library scikit-learn
from sklearn.metrics import accuracy_score, classification_report

# Misalnya: data_latih dan data_uji sudah dibagi seperti ini
X_latih = data_latih.drop("Churn", axis=1)
y_latih = data_latih["Churn"]
X_uji = data_uji.drop("Churn", axis=1)
y_uji = data_uji["Churn"].tolist()
```

```
from sklearn.tree import DecisionTreeClassifier

# Membuat model Decision Tree
model = DecisionTreeClassifier(random_state=42)
model.fit(X_latih, y_latih)

# Prediksi data uji
y_pred_sklearn_dt = model.predict(X_uji)

# Evaluasi hasil prediksi
print("Akurasi:", accuracy_score(y_uji, y_pred_sklearn_dt))
print("\nLaporan klasifikasi:\n", classification_report(y_uji, y_pred_sklearn_dt))

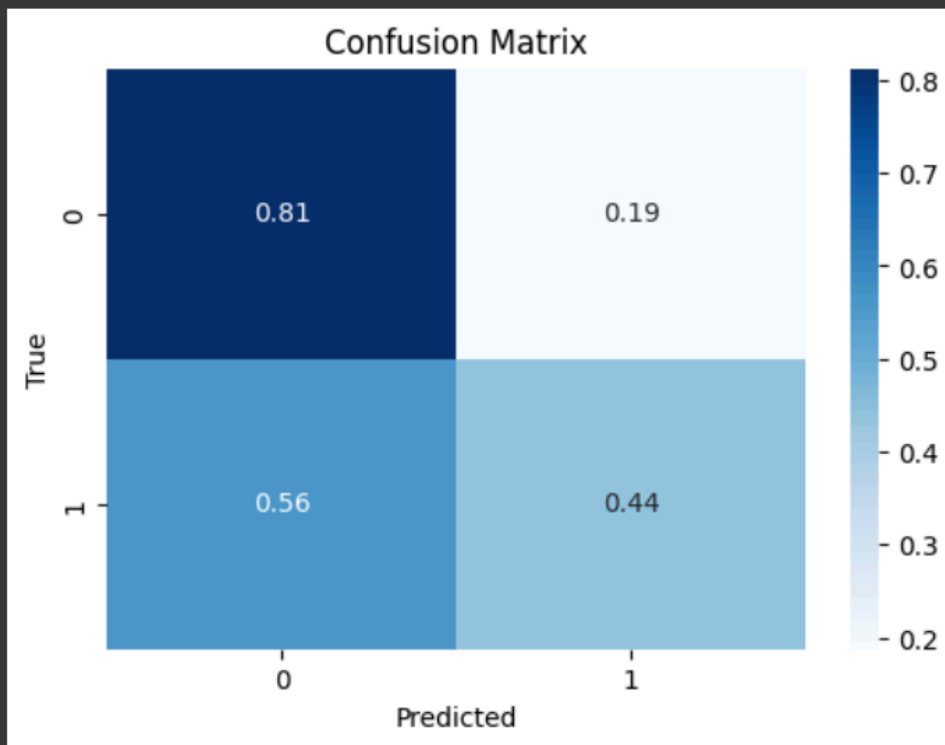
# Confusion matrix
cm = confusion_matrix(y_uji, y_pred_sklearn_dt)
print(cm, "\n")
```

4.8.3.2 Output

```
⇒ Accuracy : 0.7139815471965933
Precision per kelas: {0: np.float64(0.800952380952381), 1: np.float64(0.4596100278551532)}
Recall per kelas: {0: np.float64(0.81256038647343), 1: np.float64(0.4411764705882353)}
F1-score per kelas: {0: np.float64(0.8067146282973622), 1: np.float64(0.4502046384720328)}

Rata-rata:
Precision: 0.63, Recall: 0.63, F1: 0.63
```

```
⇒ Confusion Matrix DT:
[[841 194]
 [209 165]]
```



Akurasi: 0.7253371185237757

Laporan klasifikasi:

	precision	recall	f1-score	support
0	0.82	0.81	0.81	1035
1	0.48	0.50	0.49	374
accuracy			0.73	1409
macro avg	0.65	0.65	0.65	1409
weighted avg	0.73	0.73	0.73	1409

4.8.4 Evaluasi SVM

4.8.4.1 Kode

```
precision_svm, avg_precision_svm = calculate_precision(metrics_svm)
recall_svm, avg_recall_svm = calculate_recall(metrics_svm)
f1_svm, avg_f1_svm = calculate_f1_score(metrics_svm)
accuracy_svm = calculate_accuracy_score(metrics_svm)

print("Accuracy :", accuracy_svm)
print("Precision per kelas:", precision_svm)
```

```
print("Recall per kelas:", recall_svm)
print("F1-score per kelas:", f1_svm)
print("\nRata-rata: ")
print(f"Precision:    {avg_precision_svm:.2f},    Recall:    {avg_recall_svm:.2f},    F1:
{avg_f1_svm:.2f}")

# Cetak hasil
print("Confusion Matrix SVM:")
print(conf_matrix_svm)

norm_cm_svm = normalize_confusion_matrix(conf_matrix_svm)

cm_plot(norm_cm_svm, y_true, y_pred_svm)

from sklearn.svm import SVC
# Inisialisasi dan latih model SVM
svm_model = SVC(kernel='linear', random_state=101)
svm_model.fit(X_latih, y_latih)

# Prediksi data uji
y_pred_sklearn_svm = svm_model.predict(X_uji)

# Evaluasi hasil prediksi
print("Akurasi SVM sklearn:", accuracy_score(y_uji, y_pred_sklearn_svm))
print("\nLaporan    klasifikasi    SVM    sklean:\n",    classification_report(y_uji,
y_pred_sklearn_svm))

# confusion matrix
cm = confusion_matrix(y_uji, y_pred_sklearn_svm)

print(cm, "\n")
```

4.8.4.2 Output

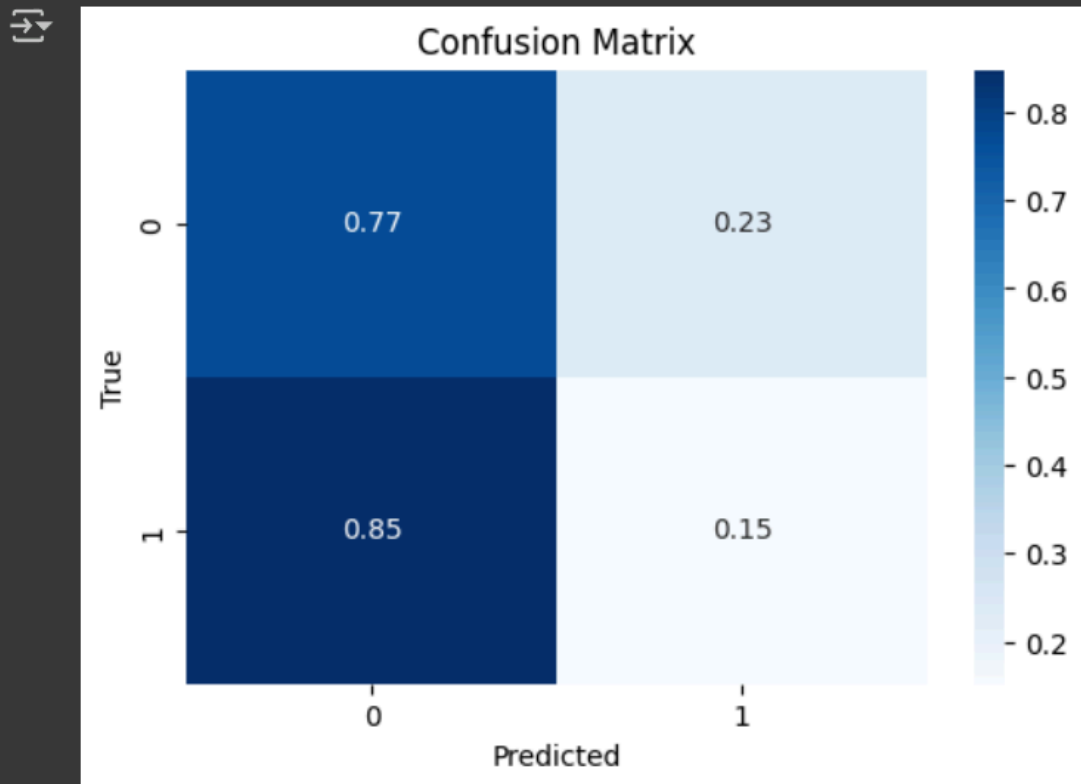
```
Accuracy : 0.602555003548616
Precision per kelas: {0: np.float64(0.7141568981064021), 1: np.float64(0.19)}
Recall per kelas: {0: np.float64(0.7652173913043478), 1: np.float64(0.15240641711229946)}
F1-score per kelas: {0: np.float64(0.7388059701492538), 1: np.float64(0.16913946587537093)}

Rata-rata:
Precision: 0.45, Recall: 0.46, F1: 0.45
```

```

[→] Confusion Matrix SVM:
      [[792 243]
       [317  57]]

```



Perbandingan dengan SVM Classifier dari scikit-learn


```

[→] Akurasi SVM sklearn: 0.7877927608232789

```

Laporan klasifikasi SVM sklearn:

	precision	recall	f1-score	support
0	0.84	0.88	0.86	1035
1	0.62	0.53	0.57	374
accuracy			0.79	1409
macro avg	0.73	0.70	0.71	1409
weighted avg	0.78	0.79	0.78	1409



```
[[913 122]
 [177 197]]
```

4.8.5 Evaluasi *Random Forest*

4.8.5.1 Kode

```
precision_rf, avg_precision_rf = calculate_precision(metrics_rf)
recall_rf, avg_recall_rf = calculate_recall(metrics_rf)
f1_rf, avg_f1_rf = calculate_f1_score(metrics_rf)
accuracy_rf = calculate_accuracy_score(metrics_rf)


print("Accuracy :", accuracy_rf)
print("Precision per kelas:", precision_rf)
print("Recall per kelas:", recall_rf)
print("F1-score per kelas:", f1_rf)
print("\nRata-rata: ")
print(f"Precision: {avg_precision_rf:.2f}, Recall: {avg_recall_rf:.2f}, F1: {avg_f1_rf:.2f}")

# Cetak hasil
print("Confusion Matrix RT:")
print(conf_matrix_rf)

norm_cm_rf = normalize_confusion_matrix(conf_matrix_rf)

cm_plot(norm_cm_rf, y_true, y_pred_rf)
```

4.8.5.2 Output



```
Accuracy : 0.7821149751596878
Precision per kelas: {0: np.float64(0.8297101449275363), 1: np.float64(0.6098360655737705)}
Recall per kelas: {0: np.float64(0.885024154589372), 1: np.float64(0.49732620320855614)}
F1-score per kelas: {0: np.float64(0.8564749883122955), 1: np.float64(0.5478645066273932)}

Rata-rata:
Precision: 0.72, Recall: 0.69, F1: 0.70
```

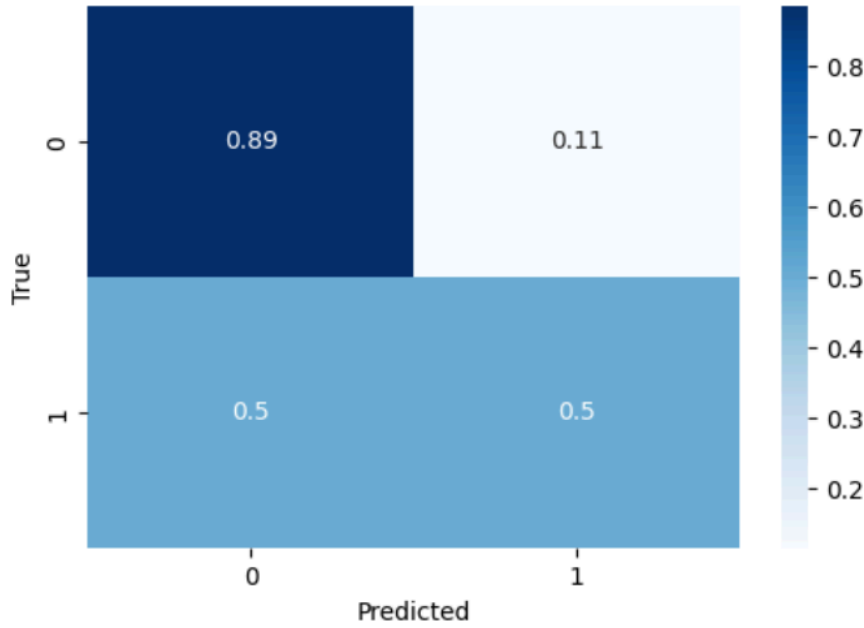


Confusion Matrix RT:

```
[[916 119]  
 [188 186]]
```



Confusion Matrix



BAB V

PEMBAHASAN & HASIL

5.1 Analisis Data

Ditemukan bahwa distribusi kelas pada data tidak seimbang, di mana kelas ‘Churn’ (1) hanya sebesar 26.54% dan kelas ‘Tidak Churn’ (0) mendominasi dengan 73.46% dari total data. Ketidakseimbangan ini memengaruhi kinerja model karena model cenderung lebih bermain aman dengan memprediksi kelas mayoritas, sehingga akurasi model bisa tampak tinggi, tetapi sebenarnya tidak mencerminkan kinerja yang baik dalam mendeteksi churn.

5.2 Fokus Evaluasi

Karena Churn berarti potensi kehilangan pelanggan dan pendapatan, maka model yang dirancang harus mampu mengenali pelanggan yang berisiko churn secara akurat dan seimbang. Oleh karena itu, metrik evaluasi utama difokuskan pada:

- *Recall* kelas 1: Kemampuan model dalam mengenali pelanggan *churn* (*True Positive Rate*).
- *Presisi* kelas 1: Ketepatan model dalam memprediksi *churn*.
- *F1-score* kelas 1: Harmoni antara *precision* dan *recall*.

5.3 Analisis Kinerja Model

Model	Akurasi	Precision_0	Precision_1	Recall_0	Recall_1	F1_0	F1_1	Avg_Precision	Avg_Recall	Avg_F1
Random Forest (Scikit-Learn)	0,7821	0,8297	0,6098	0,885	0,4973	0,8564	0,5478	0,72	0,69	0,7
SVM (Scikit-Learn)	0,7879	0,84	0,62	0,88	0,53	0,86	0,57	0,73	0,7	0,71
Decision Tree (Scikit-Learn)	0,7253	0,82	0,48	0,81	0,5	0,81	0,49	0,65	0,65	0,65
Decision Tree (Manual)	0,7139	0,8009	0,4596	0,8125	0,4411	0,8067	0,4502	63	0,63	0,63
SVM (Manual)	0,6025	0,7141	0,19	0,7652	0,1524	0,7388	0,1691	0,45	0,46	0,45

Evaluasi performa dilakukan terhadap lima model machine learning: Random Forest (Scikit-Learn), SVM (Scikit-Learn), Decision Tree (Scikit-Learn), Decision Tree (Manual), dan SVM (Manual). Fokus utama evaluasi adalah kemampuan model dalam mengenali pelanggan yang akan churn, karena hal ini berdampak langsung pada potensi kehilangan pendapatan perusahaan. Oleh karena itu, metrik utama yang digunakan dalam penilaian adalah *Recall_1*, *Precision_1*, dan *F1_1*, yaitu *recall*, *precision*, dan *f1-score* khusus untuk kelas

5.3.1 SVM Scikit-Learn

Model ini menyeimbangkan kemampuan mengenali pelanggan churn dengan ketepatan prediksi. Meskipun nilai *recall_1* tidak terlalu tinggi, yaitu 0,53 (yang berarti masih ada pelanggan churn yang lolos dari deteksi), nilai *precision_1*-nya relatif lebih baik dibanding model lain, dan nilai *F1_1* tertinggi menunjukkan bahwa model ini tetap memiliki keseimbangan terbaik antara mendeteksi churn dan menghindari kesalahan prediksi churn palsu.

5.3.2 Random Forest

Random Forest menjadi alternatif kedua yang cukup stabil, meskipun performa terhadap kelas churn sedikit di bawah SVM. Nilai precision_1 dan recall_1 yang cukup seimbang menghasilkan F1-score kelas 1 yang cukup baik dibanding beberapa model lain. Random Forest juga menunjukkan nilai recall_0 sebesar 0,885, yang berarti sangat akurat dalam mengenali pelanggan yang tidak churn. Ini menjadikan model ini aman digunakan dalam konteks di mana kesalahan memprediksi pelanggan tetap (non-churn) sebagai churn harus diminimalkan.

5.3.3 Decision Tree Scikit-Learn

Nilai Recall_1 dan Precision_1 berada di level menengah, yang artinya model cukup baik dalam mengenali pelanggan churn tetapi masih menghasilkan prediksi positif palsu dalam jumlah yang signifikan. Dengan nilai F1_1 sebesar 0,49, model ini lebih baik dari model manual, namun tidak seefisien SVM atau Random Forest dalam menangani ketidakseimbangan kelas. Model Decision Tree (Scikit-Learn) menunjukkan performa yang cukup seimbang, namun secara umum masih berada di bawah dua model sebelumnya

5.3.4 Decision Tree Manual

Model Decision Tree (Manual) merupakan hasil implementasi pohon keputusan tanpa menggunakan pustaka pembelajaran mesin seperti Scikit-Learn. Secara performa, model ini menunjukkan hasil yang relatif rendah. Akurasinya hanya mencapai 0,7198, dan nilai Precision_1 serta Recall_1 masing-masing hanya sebesar 0,46 dan 0,45. Nilai F1_1 juga rendah, yaitu 0,45, menandakan bahwa model tidak mampu mengenali pelanggan yang berpotensi churn secara efektif. Rata-rata metrik F1 keseluruhan pun hanya berada di angka 0,45. Hasil ini mencerminkan bahwa model mengalami kesulitan dalam menangani data yang tidak seimbang, di mana kelas churn lebih sedikit dibanding kelas non-churn. Meski tidak direkomendasikan untuk implementasi nyata, model ini tetap memiliki nilai edukatif karena mampu menunjukkan proses pembentukan pohon keputusan dari awal, termasuk perhitungan entropi dan information gain. Dengan demikian, model ini lebih tepat digunakan sebagai sarana pembelajaran konsep dasar machine learning daripada sebagai solusi bisnis.

5.3.5 SVM Manual

Model SVM (Manual) adalah implementasi Support Vector Machine yang dibangun dari nol tanpa pustaka pembelajaran mesin. Sayangnya, performa model ini adalah yang paling rendah dibandingkan semua model lain dalam penelitian ini. Akurasinya hanya mencapai 0,6083, dan Precision_1 serta Recall_1 sangat rendah, yaitu masing-masing 0,19 dan 0,15. Hal ini menghasilkan nilai F1_1 sebesar 0,16, yang berarti model hampir tidak mampu mengidentifikasi pelanggan churn dengan baik. Sebagian besar prediksi churn yang dihasilkan adalah salah (false positive tinggi), dan banyak pelanggan yang sebenarnya churn tidak terdeteksi (false negative tinggi). Rata-rata skor F1 yang hanya sebesar 0,45 menunjukkan bahwa model ini tidak layak digunakan dalam konteks prediksi nyata. Kinerja

rendah ini kemungkinan besar disebabkan oleh kurangnya teknik optimasi margin, pemilihan kernel, dan penyesuaian parameter penting yang tersedia di pustaka seperti Scikit-Learn. Meski begitu, membangun model SVM secara manual tetap memberikan pemahaman mendalam tentang prinsip kerja algoritma ini, termasuk konsep margin, support vector, dan pemisahan kelas secara linier.

5.4 Analisis Performa Model Manual vs Scikit-Learn

Hasil percobaan menunjukkan bahwa model dari library Scikit-Learn lebih unggul dibandingkan model manual, baik dari sisi akurasi, precision, recall, maupun F1-score. Perbedaan ini muncul karena beberapa faktor utama:

1. Optimasi Algoritma

Scikit-learn mengimplementasikan algoritma dengan efisiensi tinggi, banyak di antaranya dibangun di atas pustaka numerik seperti NumPy dan SciPy yang ditulis dalam bahasa C. Hal ini memberikan performa yang jauh lebih cepat dibanding implementasi manual dalam Python murni (Garreta, R., & Moncecchi, G., 2013).

2. Penanganan Data Tidak Seimbang

Scikit-learn mendukung metode penyesuaian bobot kelas (class weight adjustment) dan parameter lain yang memungkinkan model untuk lebih memperhatikan kelas minoritas seperti churn (Sharma & Sharma, 2021). Hal ini penting untuk menangani ketidakseimbangan data yang sering terjadi dalam kasus nyata. Sebaliknya, model yang dibangun secara manual umumnya tidak memiliki fleksibilitas ini, sehingga cenderung menghasilkan prediksi bias terhadap kelas mayoritas.

3. Akurasi dan Ketepatan

Algoritma yang dibangun menggunakan Scikit-learn cenderung memberikan performa yang lebih akurat karena telah melalui banyak pengujian dan optimasi (Sharma & Sharma, 2021). Ini sejalan dengan hasil evaluasi pada penelitian ini, di mana model Scikit-learn menunjukkan nilai F1 rata-rata (Avg_F1) lebih tinggi, yaitu sekitar 0,70, dibandingkan model manual yang hanya mencapai 0,45 untuk SVM dan 0,63 untuk Decision Tree.

4. Keterbatasan Fitur

Model manual sering kali tidak menyertakan fitur penting seperti, pemilihan *hyperparameter* otomatis, atau pemrosesan data lanjutan yang dibantu oleh library. Penggunaan *library* seperti Scikit-learn memungkinkan implementasi fitur penting seperti validasi silang (*cross-validation*), pemilihan *hyperparameter* otomatis, dan integrasi preprocessing lanjutan (Ahmad et al., 2020). Hal ini membuat pipeline model menjadi lebih efisien dan andal dibanding implementasi dari nol secara manual yang lebih rentan kesalahan dan memakan waktu.

BAB VI

KESIMPULAN & SARAN

6.1 Kesimpulan

Hasil penugasan ini menunjukkan bahwa ketidakseimbangan kelas dalam dataset pelanggan berdampak signifikan terhadap efektivitas model prediktif, khususnya dalam mengidentifikasi pelanggan yang berpotensi berhenti berlangganan (*churn*). Oleh karena itu, metrik evaluasi seperti recall, precision, dan F1-score pada kelas minoritas (*churn*) menjadi krusial untuk mengukur keberhasilan model secara menyeluruh.

Dari lima model yang diuji, SVM dari Scikit-Learn terbukti paling seimbang dalam mengenali dan memprediksi *churn*, dengan nilai F1 tertinggi di antara semua model. Random Forest juga memberikan hasil yang cukup baik dan stabil, terutama dalam meminimalkan kesalahan prediksi pada pelanggan yang tidak *churn*. Sementara itu, model Decision Tree dari Scikit-Learn menempati posisi menengah dengan performa yang masih layak, meskipun tidak seoptimal dua model sebelumnya.

Sebaliknya, model yang dibangun secara manual, baik Decision Tree maupun SVM manual menghasilkan kinerja yang jauh lebih rendah. Hal ini menunjukkan adanya keterbatasan dari pendekatan manual dalam menangani kompleksitas data dan kebutuhan optimasi model, terutama tanpa dukungan *library* seperti Scikit-Learn yang menyediakan fitur penyeimbangan kelas, pemilihan hyperparameter otomatis, dan efisiensi komputasi.

Secara keseluruhan, penugasan ini menegaskan pentingnya penggunaan *library* pembelajaran mesin yang telah teruji dalam pengembangan model prediksi, terutama ketika menangani data yang tidak seimbang dan berskala besar. Meskipun model manual memberikan wawasan konseptual yang berharga, penerapannya dalam konteks bisnis nyata masih belum direkomendasikan karena keterbatasan akurasi dan efisiensinya.

6.2 Saran

Untuk pengembangan lebih lanjut, disarankan untuk menguji dataset dengan menggunakan algoritma supervised learning lainnya, seperti *Gradient Boost*, *XGBoost*, atau *Naive Bayes* untuk dibandingkan performanya terhadap model yang telah diuji sebelumnya. Penambahan algoritma ini dapat memperluas wawasan mengenai efektivitas berbagai pendekatan dalam mendeteksi pelanggan *churn* dan meningkatkan peluang mendapatkan model dengan performa lebih optimal, khususnya dalam menghadapi data yang tidak seimbang. Selain itu, penggunaan teknik ensemble atau kombinasi beberapa model juga layak dieksplorasi untuk meningkatkan akurasi prediksi model.

STUDI PUSTAKA

- Lebanov, L., Tedone, L., Ghiasvand, A., dan Paull, B. 2020. Random forests machine learning applied to gas chromatography–mass spectrometry derived average mass spectrum data sets for classification and characterisation of essential oils. *Talanta*.208:1-12
- Sarker, I. H. 2021. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science*.2(160):1-21.
- Sharma, V. (2022). A study on data scaling methods for machine learning. *International Journal for Global Academic & Scientific Research*, 1(1), 31-42.
- Song, Y.-Y., & Lu, Y. (2015). Decision tree methods: applications for classification and prediction. *Shanghai Archives of Psychiatry*, 27(2), 130–135.
- Awad, M., & Khanna, R. (2015). Support Vector Machines for Classification. In *Efficient Learning Machines* (pp. 39–66).
- Arina, F., & Ulfah, M. (2022). Analisa survival untuk mengurangi customer churn pada perusahaan telekomunikasi. *Journal Industrial Services*, 8(1), 59–62.
- Dhangar, K., & Anand, P. (2021). A review on customer churn prediction using machine learning approach. *International Journal of Innovations in Engineering Research and Technology*, 8(5), 193–201.
- Zhao, S. (2023). Customer churn prediction based on the decision tree and random forest model. *BCP Business & Management*, 44, 339–344.
- Garreta, R., & Moncecchi, G. (2013). *Learning scikit-learn: machine learning in python* (Vol. 2013). Birmingham: Packt Publishing.
- Sharma, S., & Sharma, D. (2021). A Review on Machine Learning Algorithms using Scikit-learn. *International Journal of Engineering Research & Technology (IJERT)*, 10(06), 114–119.
- Ahmad, I., Basher, M., Iqbal, M. J., & Rahim, A. (2020). Machine Learning Techniques for Data Mining: A Comparative Study. *IEEE Access*, 8, 60501–60516.