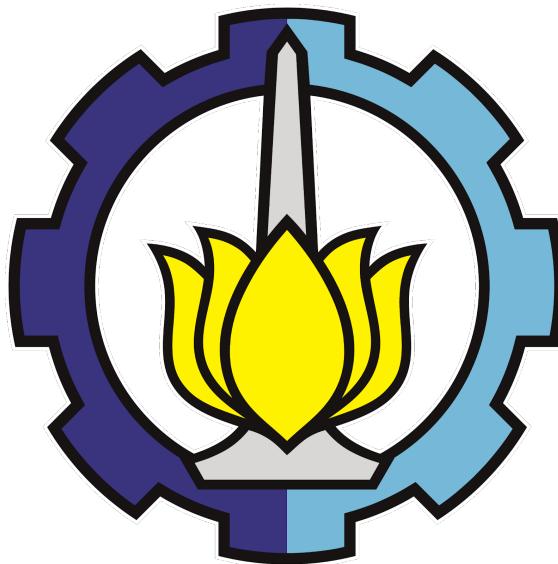


LAPORAN ANALISIS TGP #2

PREDICTIVE MODELLING ANALYTICS



Disusun Oleh:
Kelompok 3 Kelas PMA (C)

Anggota Kelompok:

Bayu Siddhi Mukti	(5026211021)
Wanda Armadianti	(5026211039)
Zahrina Candrakanti	(5026211100)
Alif Destiano	(5026211176)

DEPARTEMEN SISTEM INFORMASI
FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
2023

DAFTAR ISI

DAFTAR ISI.....	2
DETAIL DATASET.....	3
ANALISIS.....	4
1. Import Library.....	4
2. Data Loading.....	5
3. Analisis Univariat.....	5
a. Informasi Umum Data.....	6
b. Statistik Data.....	6
c. Histogram.....	7
d. Bar Chart.....	8
e. Box Plot.....	10
4. Analisis Bivariat.....	12
a. Scatter Plot.....	12
b. Stacked Bar Chart.....	13
c. Korelasi.....	16
5. Preprocessing.....	17
a. Mengatur Tipe Data.....	17
b. Mengecek Nilai Null.....	18
c. Mengecek Nilai Unik.....	18
d. Mengecek Nilai Unknown.....	19
e. Menangani Nilai Unknown.....	20
f. Mengecek Nilai Duplikat.....	21
g. Menangani Outliers.....	21
h. Melakukan Label Encoding.....	22
6. Data Balancing.....	25
7. Data Splitting.....	26
8. Feature Scaling.....	26
9. Model Klasifikasi.....	27
a. Decision Tree.....	27
b. K-Nearest Neighbors.....	30
c. Naive Bayes.....	34
d. Support Vector Machine.....	37
e. Neural Network.....	40
f. Logistic Regression.....	43
g. Random Forest.....	47
10. Model Selection.....	51
11. Backward Selection.....	51
a. Decision Tree.....	51
b. K-Nearest Neighbors.....	52
c. Naive Bayes.....	53
d. Support Vector Machine.....	54
e. Neural Network.....	55
f. Logistic Regression.....	56
g. Random Forest.....	56
12. Kesimpulan.....	57

DETAIL DATASET

Dataset Kampanye Pemasaran Langsung
(*Direct Marketing*) Deposito Bank Berjangka

Jumlah Baris Data : 45211
Rentang Waktu : Mei 2008 - November 2010
Jumlah Variabel : 17

Variabel Independen

1. age : numeric = umur
2. job : categorical = jenis pekerjaan
3. marital : categorical = jenis status pernikahan
4. education : categorical = jenis tingkat pendidikan
5. default : binary = apakah memiliki kewajiban yang gagal bayar?
6. balance : numeric = saldo tabungan (euro)
7. housing : binary = apakah memiliki kredit rumah?
8. loan : binary = apakah memiliki kredit pribadi?
9. contact : categorical = jenis kontak komunikasi
10. day : numeric = tanggal kontak terakhir
11. month : categorical = bulan kontak terakhir
12. duration : numeric = durasi kontak terakhir (detik)
13. campaign : numeric = jumlah kontak ke klien pada kampanye saat ini
14. pdays : numeric = jumlah hari terlewat setelah kontak terakhir ke klien pada kampanye sebelumnya
15. previous : numeric = jumlah kontak ke klien pada kampanye sebelumnya
16. poutcome : categorical = jenis hasil dari kampanye sebelumnya

Variabel Dependen

17. subscribe : binary = apakah klien berlangganan deposito berjangka?

Link Google Colab: [**GOOGLE COLAB**](#)

ANALISIS

1. Import Library

Sebelum memulai berbagai proses pengolahan data untuk file yang bernama (Data utk TGP#1.csv) , dilakukan import library seperti berikut.

```
[ ] import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.impute import SimpleImputer
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.compose import make_column_transformer
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import StratifiedShuffleSplit
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
```

Gambar 1. Import Library

Dari beberapa library di atas terdapat 8 library penting yang akan digunakan dalam model selection dan backward selection diantaranya yaitu :

- a. **from sklearn.tree import DecisionTreeClassifier**, merupakan library ‘DecisionTreeClassifier’ yang diambil dari scikit learn, berfungsi sebagai pembuat algoritma pohon membuat keputusan model klasifikasi decision tree termasuk dengan parameter pengukurnya
- b. **from sklearn.neighbors import KNeighborsClassifier**, merupakan library ‘KNeighborsClassifier’ yang diambil dari scikit learn, berfungsi sebagai algoritma klasifikasi yang digunakan dalam pembelajaran mesin untuk mengelompokkan data berdasarkan kedekatan dengan tetangga terdekat untuk model K-Nearest Neighbors termasuk dengan parameter pengukurnya
- c. **from sklearn.naive_bayes import GaussianNB**, merupakan library ‘GaussianNB’ yang diambil dari scikit learn, berfungsi sebagai memodelkan hubungan antara fitur dalam data dan label kelasnya untuk model Naive Bayes termasuk dengan parameter pengukurnya
- d. **from sklearn.svm import SVC**, merupakan library ‘Support Vector Classifier’ yang diambil dari scikit learn, berfungsi sebagai klasifikasi dan regresi dalam machine learning untuk model Support Vector Machine termasuk dengan parameter pengukurnya
- e. **from sklearn.neural_network import MLPClassifier**, merupakan library ‘Multi-Layer Perceptron Classifier’ yang diambil dari scikit learn, berfungsi sebagai algoritma pembelajaran khusus seperti backpropagation untuk model Neural Network termasuk dengan parameter pengukurnya

- f. **from sklearn.linear_model import LogisticRegression**, merupakan library ‘LogisticRegression’ yang diambil dari scikit learn, berfungsi sebagai algoritma berfokus pada klasifikasi biner, yang berarti memprediksi apakah suatu instance/data termasuk ke dalam satu dari dua kelas yang mungkin untuk model Logistic regression termasuk dengan parameter pengukurnya
- g. **from sklearn.ensemble import RandomForestClassifier**, merupakan library ‘RandomForestClassifier’ yang diambil dari scikit learn, berfungsi sebagai algoritma pembelajaran mesin yang termasuk dalam kategori ensemble learning untuk model Random Forest termasuk dengan parameter pengukurnya
- h. **from mlxtend.feature_selection import SequentialFeatureSelector as SFS**, merupakan library ‘mlxtend’ digunakan untuk melakukan seleksi fitur secara berurutan (sequential feature selection) menggunakan backward feature selection dalam rangka meningkatkan performa model.

2. Data Loading

Setelah menjalankan import library, langkah berikutnya yaitu melakukan pemrosesan mengubah data / file (Data utk TGP#1.csv) yang telah dimasukkan pada folder menjadi data frame dengan kode phyton sebagai berikut:

```
# Import data dari CSV menjadi data frame
df = pd.read_csv('Data utk TGP #1.csv', sep = ';')
df
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	subscribe
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no
...	
45206	51	technician	married	tertiary	no	825	no	no	cellular	17	nov	977	3	-1	0	unknown	yes
45207	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	456	2	-1	0	unknown	yes
45208	72	retired	married	secondary	no	5715	no	no	cellular	17	nov	1127	5	184	3	success	yes
45209	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	508	4	-1	0	unknown	no
45210	37	entrepreneur	married	secondary	no	2971	no	no	cellular	17	nov	361	2	188	11	other	no

45211 rows × 17 columns

Gambar 2. Hasil Informasi Import CSV ke Data Frame

3. Analisis Univariat

Analisis univariat adalah pendekatan statistik yang berfokus pada pemahaman dan deskripsi masing-masing variabel dalam suatu dataset. Analisis univariat ini meliputi analisis informasi umum data, statistik data, dan visualisasi data seperti histogram, *bar chart*, dan *box plot*. Analisis univariat membantu dalam mengidentifikasi pola atau tren dalam data tunggal dan memahami karakteristik inti dari variabel tersebut.

a. Informasi Umum Data

```
[ ] df.info()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 45211 entries, 0 to 45210  
Data columns (total 17 columns):  
 #   Column      Non-Null Count  Dtype    
---  --          --          --          --  
 0   age         45211 non-null   int64  
 1   job          45211 non-null   object  
 2   marital      45211 non-null   object  
 3   education    45211 non-null   object  
 4   default      45211 non-null   object  
 5   balance      45211 non-null   int64  
 6   housing      45211 non-null   object  
 7   loan          45211 non-null   object  
 8   contact      45211 non-null   object  
 9   day           45211 non-null   int64  
 10  month         45211 non-null   object  
 11  duration     45211 non-null   int64  
 12  campaign     45211 non-null   int64  
 13  pdays         45211 non-null   int64  
 14  previous     45211 non-null   int64  
 15  poutcome     45211 non-null   object  
 16  subscribe    45211 non-null   object  
dtypes: int64(7), object(10)  
memory usage: 5.9+ MB
```

Gambar 3. Informasi Umum Data

Gambar 2 adalah informasi umum data yang ditampilkan untuk memeriksa tipe data dan mendeteksi apakah ada nilai yang hilang. Langkah awal ini penting dalam eksplorasi data untuk memahami karakteristik data sebelum melanjutkan dengan analisis yang lebih mendalam atau pemrosesan data. Informasi yang diberikan oleh script Python ‘df.info()’ meliputi jumlah entri, jumlah kolom, tipe data, serta jumlah nilai non-null dalam setiap kolom, dan juga mengukur penggunaan memori oleh DataFrame tersebut.

b. Statistik Data

	age	balance	day	duration	campaign	pdays	previous
count	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000	45211.000000
mean	40.936210	1362.272058	15.806419	258.163080	2.763841	40.197828	0.580323
std	10.618762	3044.765829	8.322476	257.527812	3.098021	100.128746	2.303441
min	18.000000	-8019.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	72.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	448.000000	16.000000	180.000000	2.000000	-1.000000	0.000000
75%	48.000000	1428.000000	21.000000	319.000000	3.000000	-1.000000	0.000000
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	871.000000	275.000000

Gambar 4. Statistik Data

Gambar 3 memberikan ringkasan statistik untuk setiap kolom numerik. Informasi ini berguna untuk memahami karakteristik data, termasuk jumlah data, rata-rata, standar deviasi, nilai minimum, kuartil, serta nilai maksimum dalam setiap kolom. Ringkasan ini merupakan langkah yang esensial dalam eksplorasi data sebelum melanjutkan dengan analisis lebih mendalam atau pengambilan keputusan berdasarkan data.

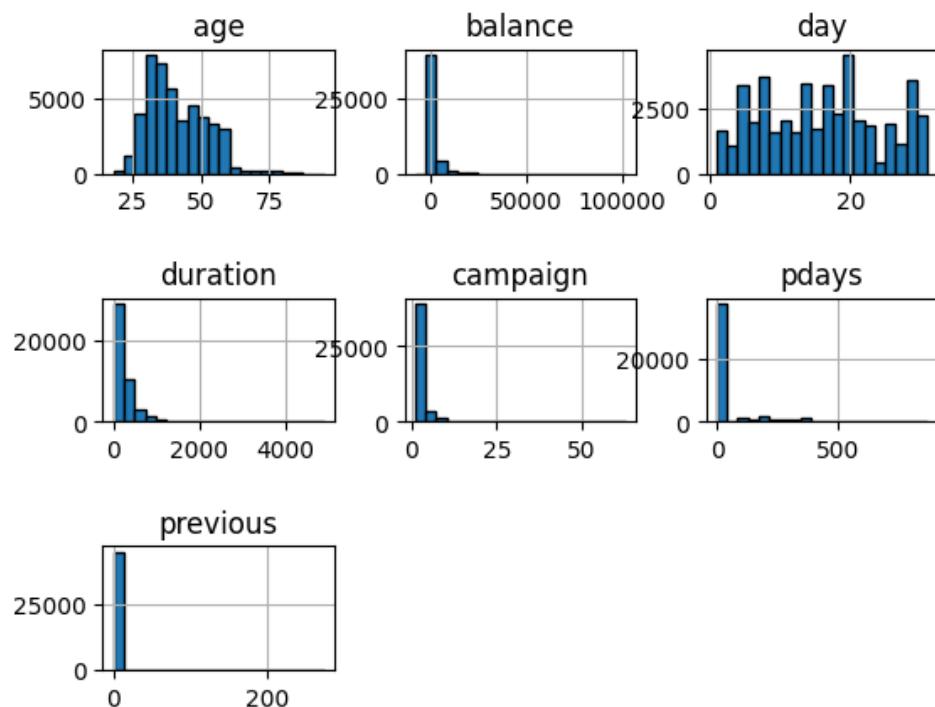
c. Histogram

Histogram digunakan untuk menggambarkan data numerik dalam bentuk grafis. Selain itu, histogram juga berfungsi untuk menunjukkan sebaran frekuensi atau kejadian nilai-nilai dalam variabel tersebut. Berikut ini adalah script Python yang digunakan untuk menampilkan histogram untuk variabel numerik:

```
# Menampilkan histogram seluruh variabel numerikal
plt.figure(figsize=(10, 10))
df.hist(bins=20, edgecolor='k')
plt.subplots_adjust(hspace=1)
plt.show()
```

Gambar 5. Script Python Histogram Variabel Numerik

Setelah script dijalankan, didapatkan tampilan histogram sebagai berikut.



Gambar 6. Histogram Variabel Numerik

Gambar 6 merupakan visualisasi masing-masing histogram variabel numerik, dimana sumbu-y menampilkan frekuensi data. Berikut merupakan ringkasan analisisnya.

Tabel 1. Hasil Analisis Histogram

Variabel	Analisis
age	Mean sedikit lebih besar dari median, sehingga histogram sedikit menunjukkan kemiringan ke kanan (right skewed).
balance	Mean jauh lebih besar dari median, sehingga histogram menunjukkan kemiringan kuat ke kanan (right skewed).
day	Tidak terlalu jauh perbedaan antara mean dan median, sehingga

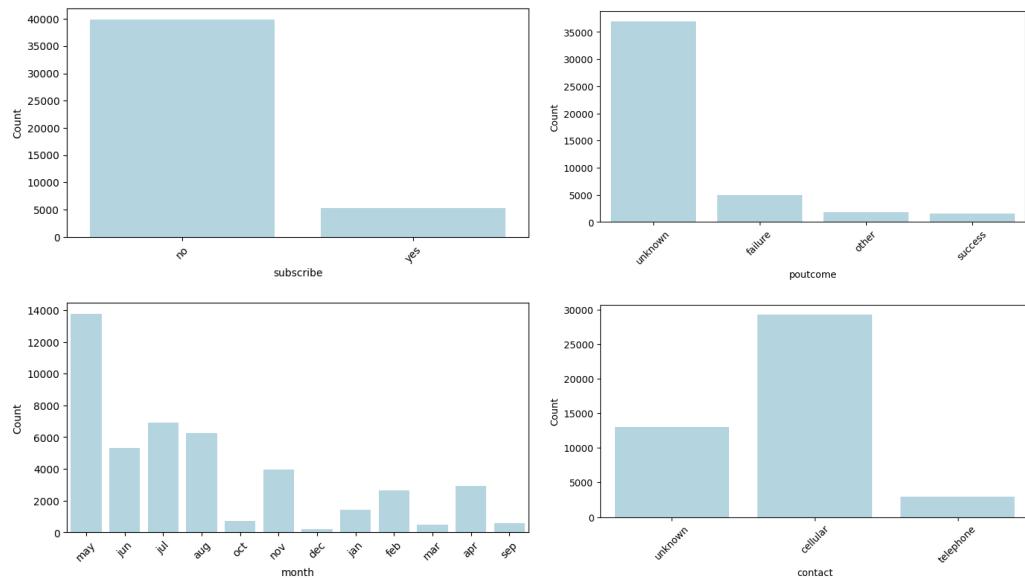
	histogram menunjukkan sedikit atau tanpa kemiringan yang signifikan.
duration	Mean jauh lebih besar dari median, sehingga histogram menunjukkan kemiringan kuat ke kanan (right skewed).
campaign	Mean sedikit lebih besar dari median, sehingga histogram sedikit menunjukkan kemiringan ke kanan (right skewed).
pdays	Mean jauh lebih besar dari median, sehingga histogram menunjukkan kemiringan kuat ke kanan (right skewed).
previous	Mean jauh lebih besar dari median, sehingga histogram menunjukkan kemiringan kuat ke kanan (right skewed).

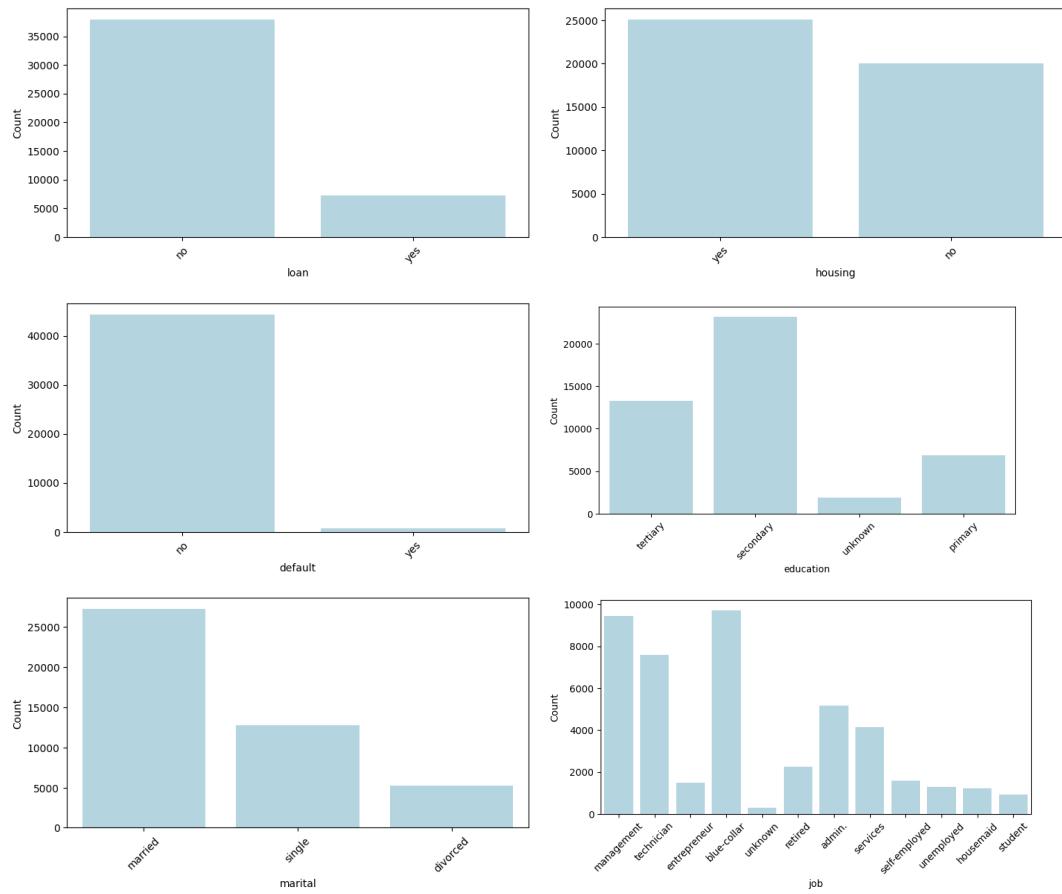
d. Bar Chart

Bar chart digunakan untuk menggambarkan data kategori atau diskrit dalam bentuk batang. Berikut ini adalah script Python yang digunakan untuk menampilkan bar chart pada variabel diskrit.

```
▶ for column in df.select_dtypes(include='object'):
    plt.figure(figsize=(8, 4))
    sns.countplot(x=df[column], data=df, color='lightblue')
    plt.xlabel(f'{column}')
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.show()
```

Gambar 7. Script Python Histogram Variabel Diskrit





Gambar 8. Histogram Variabel Diskrit

Gambar 8 merupakan visualisasi masing-masing histogram variabel diskrit, dimana sumbu-y menampilkan frekuensi data. Berikut adalah ringkasan hasil analisisnya.

Tabel 2. Hasil Analisis Bar Chart

Variabel	Analisis
subscribe	Sebagian besar klien tidak melakukan subscribe
poutcome	Sebagian besar data tidak diketahui poutcome (unknown)
month	Sebagian besar klien terakhir dikontak pada bulan Mei
contact	Sebagian besar klien menggunakan telepon seluler
loan	Sebagian besar klien tidak memiliki kredit pribadi
housing	Sebagian besar klien memiliki kredit rumah
default	Sebagian besar klien tidak memiliki kewajiban gagal bayar
education	Sebagian besar klien menempuh pendidikan hingga sekolah menengah
marital	Sebagian besar klien berstatus menikah
job	Sebagian besar klien berprofesi di bidang blue-collar

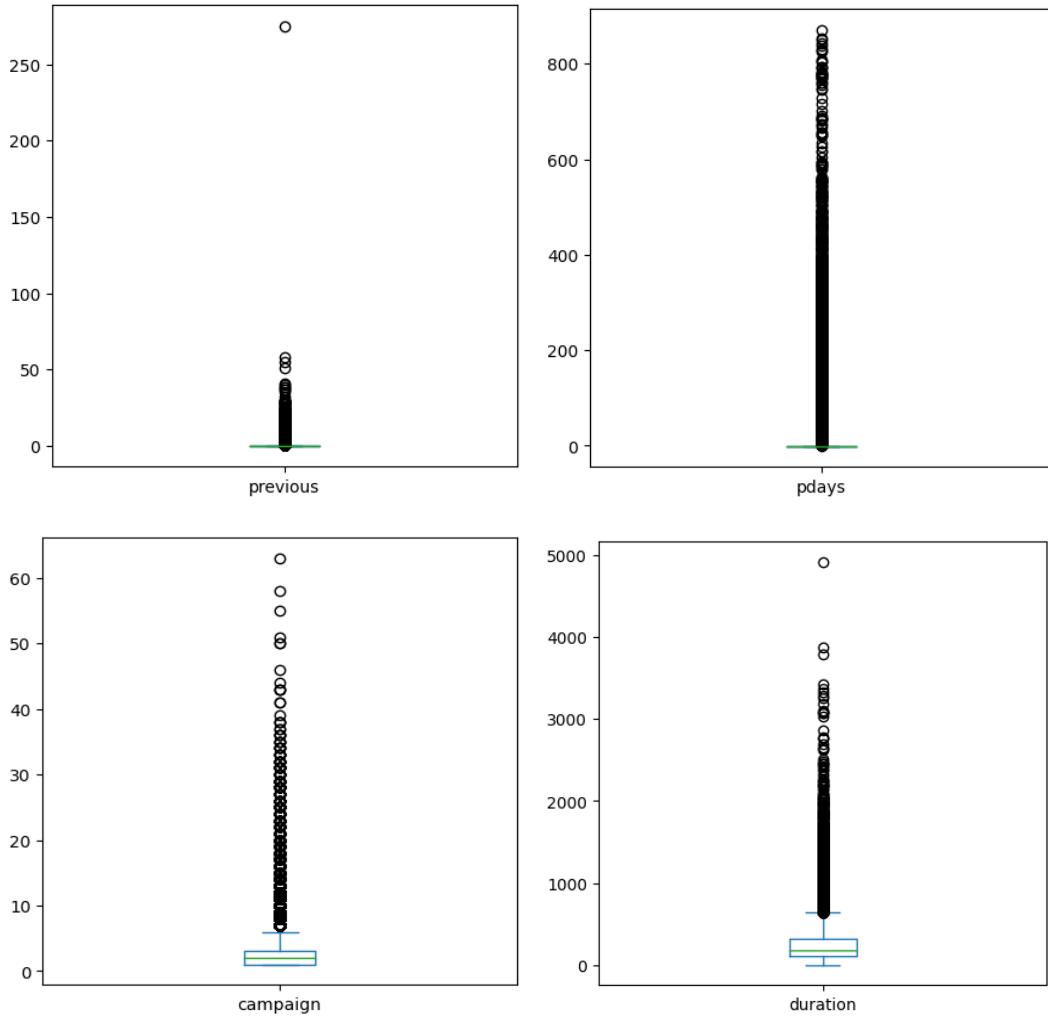
e. Box Plot

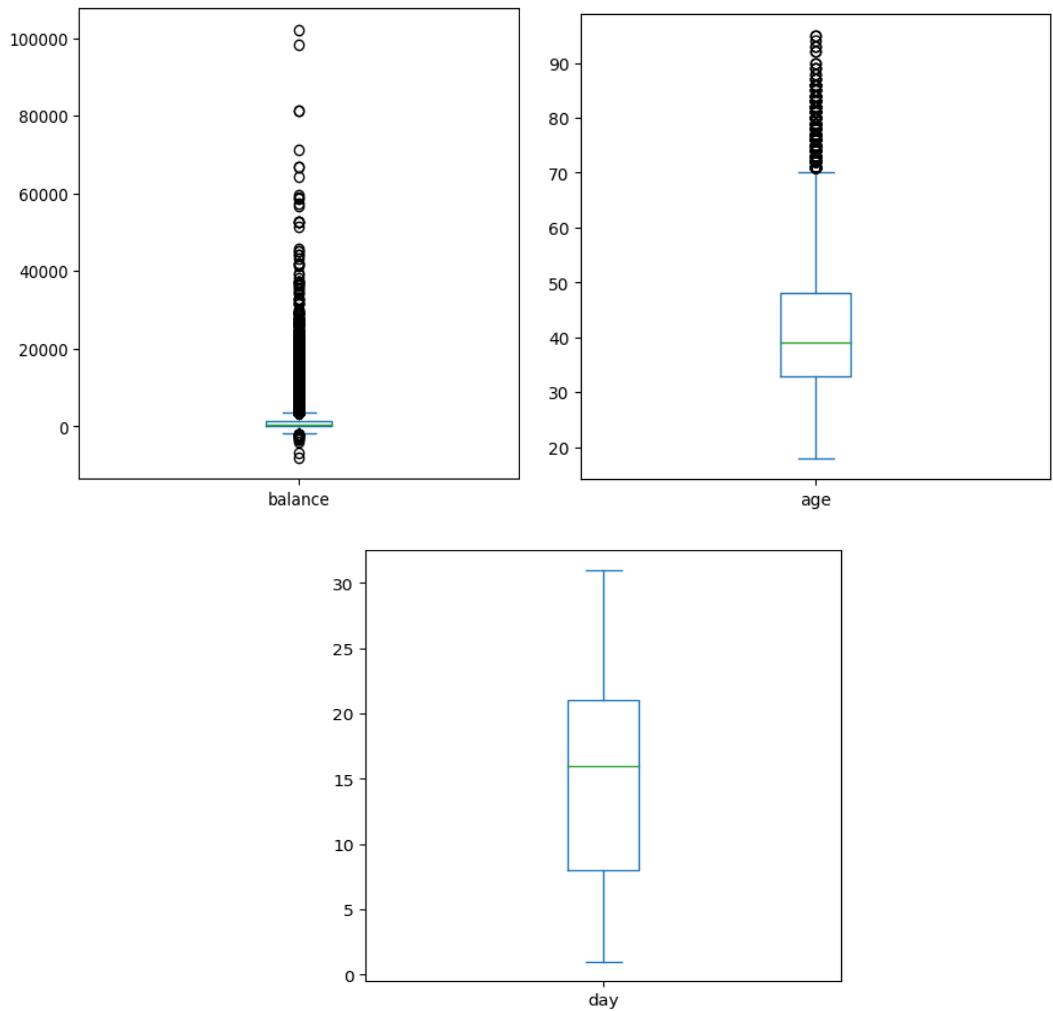
Box plot, yang juga dikenal sebagai diagram kotak, digunakan untuk menunjukkan sebaran dan distribusi data. Berikut ini adalah script Python yang digunakan untuk menampilkan box plot pada variabel numerik.

```
[ ] for column in df.select_dtypes(include='number'):
    plt.figure(figsize = (5, 5))
    df[column].plot(kind='box')
    plt.show()
```

Gambar 9. Script Python Box Plot

Setelah script dijalankan, didapatkan visualisasi box plot sebagai berikut.





Gambar 10. Box Plot

Berdasarkan Gambar 10, ringkasan analisis persebaran data pada masing-masing variabel dapat dilihat pada tabel berikut.

Tabel 3. Hasil Analisis Box Plot

Variabel	Analisis
previous	Persebaran data tidak merata, banyak outlier
pdays	Persebaran data tidak merata, banyak outlier
campaign	Persebaran data tidak merata, banyak outlier
duration	Persebaran data tidak merata, banyak outlier
balance	Persebaran data tidak merata, banyak outlier
age	Persebaran data tidak merata, banyak outlier
day	Persebaran data merata, tidak ada outlier

4. Analisis Bivariat

Analisis bivariate adalah teknik statistik yang digunakan untuk mengeksplorasi hubungan atau asosiasi antara dua variabel dalam sebuah dataset. Tujuan dari analisis bivariate adalah untuk memahami bagaimana perubahan dalam suatu variabel dapat memengaruhi variabel lainnya. Analisis bivariate terdiri dari 2 analisis, yaitu analisis antara setiap pasangan variabel independen dan setiap pasangan variabel independen dengan variabel dependen.

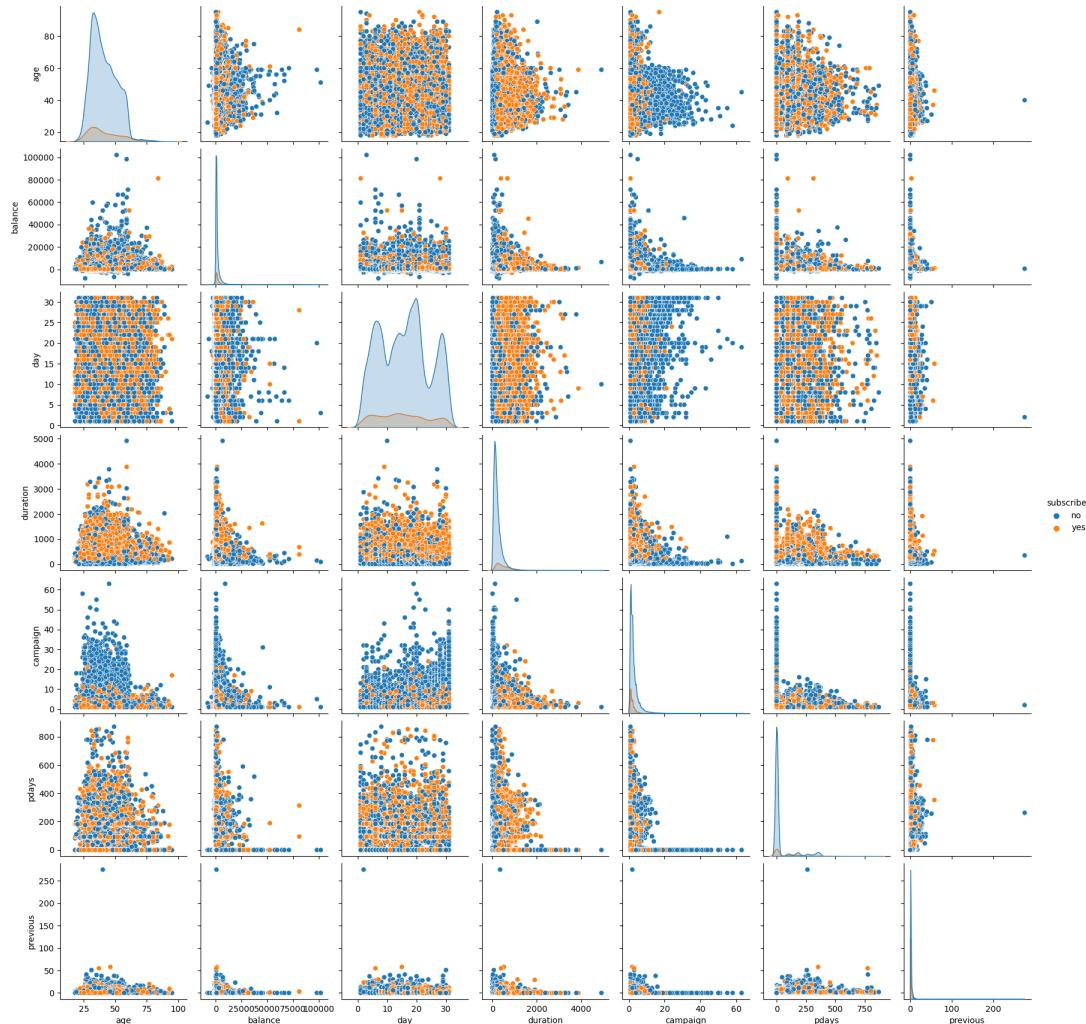
a. Scatter Plot

Berikut ini adalah script Python yang digunakan untuk menampilkan scatter plot pada variabel numerik.

```
[ ] sns.pairplot(data=df, hue='subscribe')
```

Gambar 11. Script Python Scatter Plot

Setelah script dijalankan, didapatkan visualisasi scatter plot sebagai berikut.



Gambar 12. Scatter Plot

b. Stacked Bar Chart

```
[ ] one_hot = pd.get_dummies(df['subscribe'])
df_bivariate = df.join(one_hot)
df_bivariate = df_bivariate.drop(columns = 'subscribe')
df_bivariate = df_bivariate.rename(columns={'yes': 'subscribe', 'no': 'not_subscribe'})

[ ] def stacked_bar_chart_to_subscribe(column, type = ''):
    total_subscribe = df_bivariate[df_bivariate['subscribe'] == 1].groupby(column)[['subscribe']].count().to_frame()
    total_not_subscribe = df_bivariate[df_bivariate['not_subscribe'] == 1].groupby(column)[['not_subscribe']].count()
    subscribe_relationship = total_subscribe.join(total_not_subscribe)

    plt.figure(figsize=(10, 6))

    plt.bar(subscribe_relationship.index, subscribe_relationship['subscribe'],
            label='Subscribe', color='lightcoral')
    plt.bar(subscribe_relationship.index, subscribe_relationship['not_subscribe'],
            label='Not Subscribe', bottom=subscribe_relationship['subscribe'], color='skyblue')

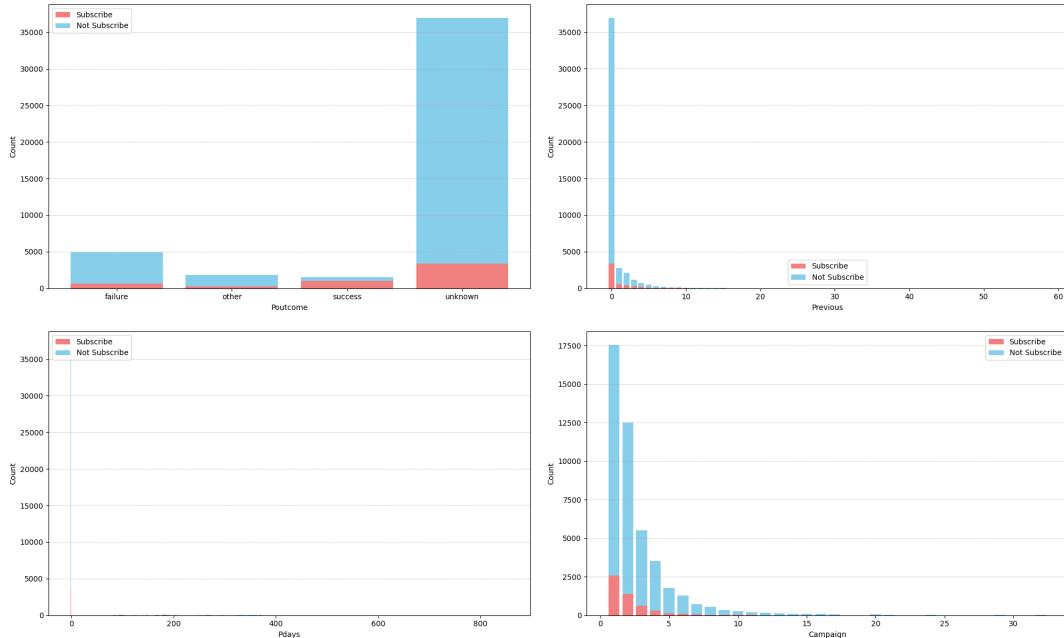
    plt.xlabel(column.capitalize())
    plt.ylabel('Count')
    if type == 'rotate_label':
        plt.xticks(rotation=30)

    plt.grid(True, which='both', linestyle='--', linewidth=0.5, axis='y')
    plt.tight_layout()
    plt.legend()

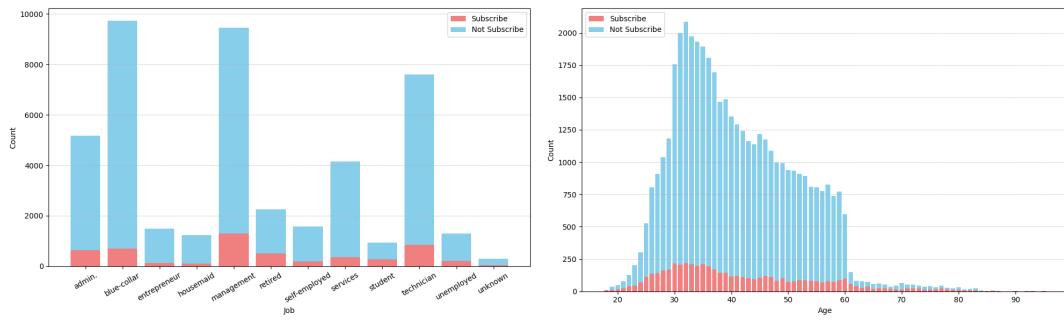
[ ] for column in df.columns:
    if column not in ['subscribe', 'not_subscribe']:
        if column in ['job']:
            stacked_bar_chart_to_subscribe(column, 'rotate_label')
        else:
            stacked_bar_chart_to_subscribe(column)
```

Gambar 13. Script Python Scatter Bar Chart

Setelah script dijalankan, didapatkan visualisasi stacked bar chart sebagai berikut.







Gambar 14. Scatter Bar Chart

Gambar 14 merupakan visualisasi analisis bivariate antara masing-masing variabel independen dan variabel dependen “subscribe”, dimana yang telah melakukan *subscribe* digambarkan dengan *bar chart* merah dan yang belum *subscribe* digambarkan dengan *bar chart* biru. Berikut merupakan hasil analisisnya.

Tabel 4. Hasil Analisis Stacked Bar Chart

Variabel Independen	Hasil Analisis Dengan variabel Dependental “subscribe”
duration	Durasi kontak terakhir tidak memiliki keterkaitan dengan subscribe yang dilakukan klien karena data tidak berkumpul pada rentang durasi tertentu/menyebar, sehingga tidak dapat ditemukan modus data pada durasi tertentu.
month	Klien yang terakhir dikontak pada bulan Mei memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
day	Klien yang terakhir dikontak pada tanggal 30 memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
contact	Klien pengguna telepon seluler memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
loan	Klien yang tidak memiliki kredit pribadi memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
housing	Klien yang tidak memiliki kredit rumah memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
balance	Klien yang memiliki balance 0-500 memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
default	Klien yang tidak memiliki kewajiban gagal bayar memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
education	Klien yang menempuh pendidikan hingga sekolah menengah memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
marital	Klien yang berstatus menikah memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
job	Klien yang profesiya berkaitan dengan bidang manajemen memiliki kecenderungan yang lebih besar untuk melakukan subscribe.

age	Klien dengan rentang usia 30-35 tahun memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
previous	Klien yang tidak di kontak pada kampanye sebelumnya memiliki kecenderungan yang lebih besar untuk melakukan subscribe.
pdays	Jumlah hari terlewat setelah kontak terakhir dengan klien pada kampanye sebelumnya tidak memiliki keterkaitan dengan subscribe yang dilakukan klien karena hanya ada satu data yang dimunculkan, yaitu 0.
campaign	Klien yang hanya dikontak sekali pada kampanye saat ini memiliki kecenderungan yang lebih besar untuk melakukan subscribe.

c. Korelasi

Angka korelasi digunakan untuk mengukur arah (positif atau negatif) dan kekuatan hubungan antara dua variabel, dimana angka korelasi berkisar antara -1 hingga 1. Angka korelasi dapat dibagi menjadi 5 kategori berikut:

- Korelasi Positif (1): Ketika angka korelasinya 1, itu menunjukkan hubungan linear positif sempurna antara dua variabel. Ini berarti ketika satu variabel naik, yang lain juga naik secara linier.
- Korelasi Positif (0 hingga 1): Ketika angka korelasi berada di antara 0 dan 1 (tidak mencapai 1), itu menunjukkan hubungan positif antara dua variabel, tetapi tidak sempurna. Semakin mendekati 1, semakin kuat hubungannya.
- Tidak Ada Korelasi (0): Angka korelasi 0 menunjukkan bahwa tidak ada hubungan linier antara dua variabel. Ini berarti perubahan dalam satu variabel tidak memiliki pengaruh linier pada variabel lainnya.
- Korelasi Negatif (0 hingga -1): Ketika angka korelasi berada di antara 0 dan -1 (tidak mencapai -1), itu menunjukkan hubungan negatif antara dua variabel, tetapi tidak sempurna. Semakin mendekati -1, semakin kuat hubungannya.
- Korelasi Negatif (-1): Ketika angka korelasinya -1, itu menunjukkan hubungan linear negatif sempurna antara dua variabel. Ini berarti ketika satu variabel naik, yang lain turun secara linier.

Kami menghitung angka korelasi antar atribut independen numerik dengan menggunakan script Python di bawah ini.

```
[ ] # Menghitung matriks korelasi untuk semua kolom
correlation_matrix = df.corr()

# Menampilkan matriks korelasi
print(correlation_matrix)

      age   balance    day duration campaign  pdays previous
age  1.000000  0.097783 -0.009120 -0.004648  0.004760 -0.023758  0.001288
balance  0.097783  1.000000  0.004503  0.021560 -0.014578  0.003435  0.016674
day   -0.009120  0.004503  1.000000 -0.030206  0.162490 -0.093044 -0.051710
duration -0.004648  0.021560 -0.030206  1.000000 -0.084570 -0.001565  0.001203
campaign  0.004760 -0.014578  0.162490 -0.084570  1.000000 -0.088628 -0.032855
pdays   -0.023758  0.003435 -0.093044 -0.001565 -0.088628  1.000000  0.454820
previous  0.001288  0.016674 -0.051710  0.001203 -0.032855  0.454820  1.000000
```

Gambar 15. Matriks Korelasi

Dari gambar 15, dapat ditarik kesimpulan bahwa hubungan antar atribut independen sangat lemah atau hampir tidak berhubungan.

5. Preprocessing

a. Mengatur Tipe Data

Untuk memulai tahap preprocessing, maka dilakukan proses mengatur tipe data. Dalam proses ini, setelah data diimpor dan dianalisis sebaran datanya, tipe data setiap data yang ada akan diperiksa menggunakan perintah **info**. Pada data “**Data utk TGP #1.csv**”, ditemukan tipe data object untuk data teks dan tipe data int64 untuk data numerik.

```
# Cek tipe data variabel
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25340 entries, 0 to 25339
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   age         25340 non-null   int64  
 1   job          25340 non-null   object  
 2   marital     25340 non-null   object  
 3   education   25340 non-null   object  
 4   default     25339 non-null   object  
 5   balance     25339 non-null   float64 
 6   housing     25339 non-null   object  
 7   loan         25339 non-null   object  
 8   contact     25339 non-null   object  
 9   day          25339 non-null   float64 
 10  month        25339 non-null   object  
 11  duration    25339 non-null   float64 
 12  campaign    25339 non-null   float64 
 13  pdays       25339 non-null   float64 
 14  previous    25339 non-null   float64 
 15  poutcome    25339 non-null   object  
 16  subscribe   25339 non-null   object  
dtypes: float64(6), int64(1), object(10)
memory usage: 3.3+ MB
```

Gambar 16. Menampilkan Info tipe data setiap variabel

Sesuai permintaan soal, terdapat beberapa kolom yang diharuskan diubah menjadi data kategorikal, yaitu kolom "job", "marital", "education", "contact", "month", dan "poutcome". Perubahan ini dilakukan dengan menggunakan perintah **astype()**.

```
df['job'] = df['job'].astype('category')
df['marital'] = df['marital'].astype('category')
df['education'] = df['education'].astype('category')
df['contact'] = df['contact'].astype('category')
df['month'] = df['month'].astype('category')
df['poutcome'] = df['poutcome'].astype('category')

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25340 entries, 0 to 25339
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype    
--- 
 0   age         25340 non-null   int64    
 1   job          25340 non-null   category 
 2   marital     25340 non-null   category 
 3   education   25340 non-null   category 
 4   default     25339 non-null   object    
 5   balance     25339 non-null   float64  
 6   housing     25339 non-null   object    
 7   loan         25339 non-null   object    
 8   contact     25339 non-null   category 
 9   day          25339 non-null   float64  
 10  month        25339 non-null   category 
 11  duration    25339 non-null   float64  
 12  campaign    25339 non-null   float64  
 13  pdays       25339 non-null   float64  
 14  previous    25339 non-null   float64  
 15  poutcome    25339 non-null   category 
 16  subscribe   25339 non-null   object    
dtypes: category(6), float64(6), int64(1), object(4)
memory usage: 2.3+ MB
```

Gambar 17. Mengubah tipe data

Didapatkan hasil **Dtype** telah untuk kolom "job", "marital", "education", "contact", "month", dan "poutcome" sudah terganti.

b. Mengecek Nilai Null

Agar data yang diproses nantinya akurat dan dapat diproses dengan baik, data akan dicek terlebih dahulu untuk mengetahui apakah terdapat field kosong atau yang bisa disebut missing value. Pengecekan ini dapat dilakukan dengan perintah **isnull().sum()**, **isnull()** sebagai pembaca null sedangkan **sum()** untuk menjumlahkan field berisi **null** di setiap kolom.

```
# Mengecek jumlah data null
jumlah_data_null = df.isnull().sum()
print(f"Jumlah data null di tiap kolom: \n{jumlah_data_null}")

Jumlah data null di tiap kolom:
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
subscribe    0
dtype: int64
```

Gambar 18. Mengecek jumlah data null

Dari hasil pengecekan menggunakan perintah **isnull().sum()**, didapatkan bahwa tidak ada field yang kosong di data tersebut. Hal ini ditunjukkan oleh nilai 0 pada setiap kolom hasil pengecekan.

c. Mengecek Nilai Unik

Untuk melakukan pengecekan nilai unik pada setiap kolom dapat dilakukan dengan membuat sebuah fungsi terlebih dahulu, yang dalam hal ini fungsi dinamakan sebagai **unique_value()**. Fungsi tersebut akan melakukan pengecekan nilai unik di setiap kolom bertipe data **kategorikal**.

```
# Function untuk melihat nilai unique value dari variabel kategorikal
def unique_value(column):
    data = []
    for value in df[column].unique():
        data.append(value)
    print(f"Nilai dari variabel {column} = {data}")

# Menampilkan semua unique value dari semua variabel kategorikal
for column in df.select_dtypes(include='category').columns:
    unique_value(column)

Nilai dari variabel job = ['management', 'technician', 'entrepreneur', 'blue-collar', 'unknown', 'retired', 'admin.', 'services', 'self-employed', 'unemployed', 'housemaid', 'student']
Nilai dari variabel marital = ['married', 'single', 'divorced']
Nilai dari variabel education = ['tertiary', 'secondary', 'unknown', 'primary']
Nilai dari variabel contact = ['unknown', 'cellular', 'telephone']
Nilai dari variabel month = ['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'jan', 'feb', 'mar', 'apr', 'sep']
Nilai dari variabel poutcome = ['unknown', 'failure', 'other', 'success']
```

Gambar 19. Mengecek dan menampilkan Info tipe data setiap variabel

Dari hasil tersebut, dapat diketahui bahwa kolom “job” memiliki 10 nilai unik, kolom “marital” memiliki 3 nilai unik, kolom “education” memiliki 3 nilai unik, kolom “contact” memiliki 2 nilai unik, kolom “month” memiliki 8 nilai unik, dan kolom “poutcome” memiliki 4 nilai unik.

d. Mengecek Nilai Unknown

Setelah mengetahui nilai unik di beberapa kolom, terdapat beberapa kolom yang memiliki nilai unik “Unknown” yang artinya tidak jauh berbeda dengan NULL, diantaranya kolom **job**, **education**, **contact**, **dan poutcome**. Jumlah nilai “Unknown” tersebut dapat diketahui dengan membuat fungsi **check_unknown_value()** yang didalamnya terdapat perulangan untuk membaca nilai “Unknown” di setiap data kategorikal.

```
# Function untuk menghitung jumlah data 'unknown' di setiap variabel categorical
def check_unknown_value():
    length = 0
    for column in df.columns:
        if length < len(column):
            length = len(column)

    print('Total data unknown di setiap variabel categorical:')

    # Menghitung jumlah data 'unknown' di setiap variabel categorical
    for column in df.select_dtypes(include='category').columns:
        unknown = (df[column] == 'unknown').sum()
        total = len(df)
        ratio = unknown/total

        text_column = column
        if len(column) < length:
            for i in range(0, (length - len(column))):
                text_column = text_column + ' '

        text_ratio = f"{unknown}/{total}"
        text_ratio_max = f"{len(df)}/{len(df)}"
        if len(text_ratio) < len(text_ratio_max):
            for i in range(0, (len(text_ratio_max) - len(text_ratio))):
                text_ratio = text_ratio + ' '

        print(f'{text_column}\t = {text_ratio}\t = {ratio}')


check_unknown_value()

Total data unknown di setiap variabel categorical:
job          = 288/45211      = 0.006370131162770122
marital      = 0/45211       = 0.0
education    = 1857/45211     = 0.04107407489327818
contact      = 13020/45211    = 0.28798301298356593
month        = 0/45211       = 0.0
poutcome     = 36959/45211    = 0.8174780473778506
```

Gambar 20. Menghitung jumlah data “Unknown” di setiap variabel categorical

Hasil dari fungsi ini menunjukkan bahwa variabel **contact** dan **poutcome** memiliki jumlah data “Unknown” yang paling banyak, yaitu masing-masing 28% dan 81%. Variabel **job** dan **education** memiliki jumlah data “Unknown” yang paling sedikit, yaitu masing-masing 0,6% dan 4%.

e. Menangani Nilai Unknown

Setelah mendapatkan hasil dari pengecekan Nilai Unknown yang ada pada setiap kolom, langkah selanjutnya yaitu melakukan pengolahan pada kolom yang memiliki nilai Unknown mulai dari menggantikan nilainya dengan modus hingga ada kolom yang dipertahankan. Berikut merupakan penjelasan detail pada kolom yang memiliki unknown yaitu kolom **job**, **education**, **contact**, dan **poutcome** :

- Untuk Kolom **job** dan **education**

karena jumlah unknown pada kolom job dan education tidak terlalu banyak, maka unknown value digantikan nilainya dengan nilai modusnya dengan menggunakan kode phyton sebagai berikut dengan pencarian mode / nilai terbanyak menggunakan **fungsi idxmax** dari kolom job dan education.

```
[ ] # Function untuk mengganti unknown value menjadi data mode-nya
def replace_unknown_to_mode(column):
    mode = df[column].value_counts().idxmax()
    df[column] = df[column].replace(to_replace = 'unknown', value = mode)

# Menangani unknown value pada variabel 'job' dan 'education' (ganti value dengan mode-nya)
replace_unknown_to_mode('job')
replace_unknown_to_mode('education')
check_unknown_value()

👤 Total data unknown di setiap variabel categorical:
job          = 0/45211      = 0.0
marital       = 0/45211      = 0.0
education     = 0/45211      = 0.0
contact       = 13020/45211   = 0.28798301298356593
month         = 0/45211      = 0.0
poutcome      = 36959/45211   = 0.8174780473778506
```

Gambar 21. Unknown Value Job Education

Didapatkan hasil nilai Unknown pada kolom job dan education menjadi tidak ada karena sudah tergantikan dengan nilai modusnya.

- Untuk Kolom **contact**

Banyaknya nilai unknown pada kolom contact tergolong sedang, agar tidak memengaruhi data maka lebih baik dihapus barisnya yang mengandung nilai unknown seperti berikut dengan menggunakan fungsi **dropna** yaitu fungsi yang digunakan untuk menghapus baris yang berisi unknown

```
[ ] df['contact'] = df['contact'].replace('unknown', np.nan)
df = df.dropna()
df = df.reset_index(drop=True)
check_unknown_value()

👤 Total data unknown di setiap variabel categorical:
job          = 0/32191      = 0.0
marital       = 0/32191      = 0.0
education     = 0/32191      = 0.0
contact       = 0/32191      = 0.0
month         = 0/32191      = 0.0
poutcome      = 24009/32191   = 0.7458295796961883
```

Gambar 22. Unknown Value Contact

Maka, didapatkan hasil nilai Unknown pada kolom contact sudah dihilangkan, dan jumlah baris yang semula sebanyak 452111 menjadi 32191.

- Untuk Kolom **poutcome**

Berdasarkan hasil dari pra proses pada check unknown value, poutcome memiliki nilai unknown yang sangat banyak (81 %) yaitu sekitar 36959 (sebelum data unknown di variabel contact dihapus), maka seharusnya kolom poutcome diputuskan untuk dihapus. Namun, agar jumlah kolom variabel independen tidak berubah, yaitu tetap 16 variabel, maka kami memutuskan untuk tetap mempertahankan variabel ini beserta nilai unknown-nya.

f. Mengecek Nilai Duplikat

Setelah Unknown Value sudah ditangani dengan baik, langkah selanjutnya yaitu melakukan pengecekan pada nilai duplikat pada data. Pengecekan dilakukan dengan menggunakan fungsi **duplicated().sum()**, fungsi tersebut mencari nilai duplikat lalu di total berapa jumlah nilai duplikat yang ada pada data. Penjabaran kode dan hasil sebagai berikut.

```
[ ] # Mencari jumlah baris yang nilai nya sama persis (duplikat)
jumlah_baris_duplikat = df.duplicated().sum()
print(f"Jumlah baris data duplikat = {jumlah_baris_duplikat}")

Jumlah baris data duplikat = 0
```

Gambar 23. Duplicate Value

Didapatkan jumlah baris yang duplikat sebanyak 0 baris menandakan bahwa baris yang sama tidak ditemukan / tidak ada pada data.

g. Menangani Outliers

Berdasarkan analisis pada univariate , Agar data memiliki rata-rata yang baik dan normal, outlier atau yang biasa disebut data pencilan harus dihapus. Outlier sendiri adalah nilai yang ekstrem dan terletak jauh dari sebagian besar data, sehingga dapat memengaruhi nilai rata-rata dari data. Maka dari itu, outlier dihapus dengan menggunakan metode z-score dengan threshold=3 sebagai berikut

```
[ ] # Menghapus outlier menggunakan z-score dengan Threshold 3
def remove_outliers_zscore_loop(df, threshold=3):
    outliers = []
    for col in df.select_dtypes(include='number').columns:
        z_scores = np.abs((df[col] - df[col].mean()) / df[col].std())
        outliers.append(z_scores <= threshold)
    outliers = np.all(outliers, axis=0)
    TGP1_no_outliers = df[outliers]
    return TGP1_no_outliers

df_cleaned = remove_outliers_zscore_loop(df)
```

Gambar 24. Metode Z-Score hapus Outlier

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	subscribe
0	27	management	single	secondary	no	35	no	no	cellular	4	jul	255	1	-1	0	unknown	no
1	54	blue-collar	married	primary	no	466	no	no	cellular	4	jul	297	1	-1	0	unknown	no
2	43	blue-collar	married	secondary	no	105	no	yes	cellular	4	jul	668	2	-1	0	unknown	no
3	31	technician	single	secondary	no	19	no	no	telephone	4	jul	65	2	-1	0	unknown	no
4	27	technician	single	secondary	no	126	yes	yes	cellular	4	jul	436	4	-1	0	unknown	no
...
32184	73	retired	married	secondary	no	2850	no	no	cellular	17	nov	300	1	40	8	failure	yes
32185	25	technician	single	secondary	no	505	no	yes	cellular	17	nov	386	2	-1	0	unknown	yes
32186	51	technician	married	tertiary	no	825	no	no	cellular	17	nov	977	3	-1	0	unknown	yes
32187	71	retired	divorced	primary	no	1729	no	no	cellular	17	nov	456	2	-1	0	unknown	yes
32189	57	blue-collar	married	secondary	no	668	no	no	telephone	17	nov	508	4	-1	0	unknown	no

29453 rows × 17 columns

Gambar 25. Hasil Hapus Outlier

Didapatkan setelah penghapusan outlier dengan metode z-score data yang semula berjumlah 32191 menjadi 29453. Penghapusan outlier hanya dilakukan pada variabel yang bernilai numerik.

h. Melakukan Label Encoding

Langkah terakhir pada bagian praproses yaitu label encoding. Label encoding sendiri adalah pemrosesan data yang digunakan untuk mengonversi data kategori atau label menjadi bentuk angka atau bilangan bulat. Dalam Data terdapat beberapa variabel yang masih memiliki tipe category dan object. Pengecekan dilakukan dengan menggunakan fungsi atau perintah **info()** dengan hasil sebagai berikut

```
[ ] df_cleaned.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 29453 entries, 0 to 32189
Data columns (total 17 columns):
 #   Column      Non-Null Count Dtype  
--- 
 0   age         29453 non-null    int64  
 1   job          29453 non-null    category
 2   marital     29453 non-null    category
 3   education   29453 non-null    category
 4   default     29453 non-null    object  
 5   balance     29453 non-null    int64  
 6   housing     29453 non-null    object  
 7   loan         29453 non-null    object  
 8   contact     29453 non-null    category
 9   day          29453 non-null    int64  
 10  month        29453 non-null    category
 11  duration    29453 non-null    int64  
 12  campaign    29453 non-null    int64  
 13  pdays       29453 non-null    int64  
 14  previous    29453 non-null    int64  
 15  poutcome    29453 non-null    category
 16  subscribe   29453 non-null    object  
 dtypes: category(6), int64(7), object(4)
memory usage: 2.9+ MB
```

Gambar 26. Hasil Hapus Outlier

Didapatkan kolom yang masih bertipe kategori yaitu kolom job, marital,education, contact,day,month dan poutcome. Sedangkan untuk kolom yang masih bernilai object yaitu kolom default, housing, loan, dan subscribe. Maka, langkah selanjutnya yaitu mengubah kolom bertipe object menjadi numerik yaitu kolom default, housing, loan, dan subscribe dengan menggunakan pelabelan no = 0, yes = 1 seperti berikut

```
[ ] # Melakukan encoding terhadap variabel binary
df_encoded = df_cleaned.replace({'default' : {'no': 0, 'yes': 1}})
df_encoded = df_encoded.replace({'housing' : {'no': 0, 'yes': 1}})
df_encoded = df_encoded.replace({'loan' : {'no': 0, 'yes': 1}})
df_encoded = df_encoded.replace({'subscribe' : {'no': 0, 'yes': 1}})
```

Gambar 27. Label Encoding untuk Object

Berikutnya mengubah kolom yang bertipe kategorikal ordinal yaitu kolom education dan month menjadi numerikal dengan melabelkan setiap isi unknown hingga tertiary dengan angka 1-4. Sedangkan month dilabelkan dengan 1-12 seperti berikut

```
[ ] # Melakukan encoding terhadap variabel categorical ordinal
df_encoded = df_encoded.replace({'education' : {'unknown': 1, 'primary': 2, 'secondary': 3, 'tertiary': 4}})
df_encoded = df_encoded.replace(
    {'month' : {
        'jan': 1, 'feb': 2, 'mar': 3,
        'apr': 4, 'may': 5, 'jun': 6,
        'jul': 7, 'aug': 8, 'sep': 9,
        'oct': 10, 'nov': 11, 'dec': 12}})

df_encoded['education'] = df_encoded['education'].astype('int64')
df_encoded['month'] = df_encoded['month'].astype('int64')
```

Gambar 28. Label Encoding untuk kategorikal ordinal

Kolom terakhir yang belum diubah menjadi numerik yaitu kolom job, marital, education, contact, dan poutcome. kolom-kolom tersebut dilabeli secara otomatis sebagai berikut

```
[ ] # Melakukan encoding terhadap variabel categorical nominal
df_encoded['job'] = df_encoded['job'].cat.codes
df_encoded['marital'] = df_encoded['marital'].cat.codes
df_encoded['contact'] = df_encoded['contact'].cat.codes
df_encoded['poutcome'] = df_encoded['poutcome'].cat.codes
```

Gambar 29. Label Encoding untuk nominal

Maka, hasil akhir dengan mengubah tipe data menjadi bentuk angka atau integer dapat diperiksa dengan menampilkan kembali data frame yang telah dibersihkan/ di encoding yaitu df_encoded.info()

```
[ ] df_encoded.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 29453 entries, 0 to 32189
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   age         29453 non-null   int64  
 1   job          29453 non-null   int8   
 2   marital     29453 non-null   int8   
 3   education   29453 non-null   int64  
 4   default     29453 non-null   int64  
 5   balance     29453 non-null   int64  
 6   housing     29453 non-null   int64  
 7   loan         29453 non-null   int64  
 8   contact     29453 non-null   int8   
 9   day          29453 non-null   int64  
 10  month        29453 non-null   int64  
 11  duration    29453 non-null   int64  
 12  campaign    29453 non-null   int64  
 13  pdays       29453 non-null   int64  
 14  previous    29453 non-null   int64  
 15  poutcome    29453 non-null   int8   
 16  subscribe   29453 non-null   int64  
dtypes: int64(13), int8(4)
memory usage: 3.3 MB
```

Gambar 30. Menampilkan Info tipe data setiap variabel

```
[ ] df_encoded

   age job marital education default balance housing loan contact day month duration campaign pdays previous poutcome subscribe
0   27   4      2         3      0     35      0   0     0   4     7    255     1   -1     0     3     0
1   54   1      1         2      0     466      0   0     0   4     7    297     1   -1     0     3     0
2   43   1      1         3      0     105      0   1     0   4     7    668     2   -1     0     3     0
3   31   9      2         3      0     19      0   0     1   4     7     65     2   -1     0     3     0
4   27   9      2         3      0     126      1   1     0   4     7    436     4   -1     0     3     0
...
32184 73   5      1         3      0     2850      0   0     0   17    11    300     1   40     8     0     1
32185 25   9      2         3      0     505      0   1     0   17    11    386     2   -1     0     3     1
32186 51   9      1         4      0     825      0   0     0   17    11    977     3   -1     0     3     1
32187 71   5      0         2      0     1729      0   0     0   17    11    456     2   -1     0     3     1
32189 57   1      1         3      0     668      0   0     1   17    11    508     4   -1     0     3     0
29453 rows × 17 columns
```

Gambar 31. Hasil Label Encoding

6. Data Balancing

Sebelum data dipisahkan menjadi train set dan test set hingga digunakan ke dalam model machine learning, data harus diseimbangkan. Data harus diseimbangkan berdasarkan jumlah dari variabel dependen yang menjadi tujuan, yaitu variabel ‘subscribe’.

```
[ ] # Mengecek keseimbangan setiap value label tujuan
df_encoded['subscribe'].value_counts()

0    25562
1    3891
Name: subscribe, dtype: int64
```

Gambar 32. Mengecek Keseimbangan Setiap Nilai pada Variabel Dependen

Berdasarkan hasil pengukuran keseimbangan, didapatkan fakta bahwa data 0 yang merepresentasikan ‘no’ memiliki jumlah 21671 lebih banyak daripada data 1 yang merepresentasikan nilai ‘yes’. Oleh karena itu, data yes (1) harus diseimbangkan sehingga jumlahnya sama menjadi 25562. Sebelum dilakukan penyeimbangan, harus dilakukan proses pemisahan feature (variabel independen/prediktor) dengan label (variabel dependen/target) seperti berikut.

```
[ ] # Memisahkan Feature dan Label
x = df_encoded.drop('subscribe', axis=1) # X = Feature
y = df_encoded['subscribe'] # y = Label

x.reset_index(drop=True, inplace=True)
y.reset_index(drop=True, inplace=True)
```

Gambar 33. Proses Memisahkan Feature dengan Label ke Dalam Dua Dataframe Berbeda

Setelah feature dan label telah dipisah, selanjutnya proses penyeimbangan dapat dilakukan. Penyeimbangan dilakukan dengan menggunakan instance dari RandomOverSampler() milik library imblearn, dengan memasukkan dataframe feature dan label sebagai input untuk dilakukan penyeimbangan.

```
[ ] # Menyeimbangkan value label tujuan
balancer = RandomOverSampler(random_state=42)
X_balanced, y_balanced = balancer.fit_resample(x, y)
y_balanced.value_counts()

0    25562
1    25562
Name: subscribe, dtype: int64
```

Gambar 34. Proses Menyeimbangkan Value Berdasarkan Nilai Variabel Dependen

Hasilnya, didapatkan data feature dan label yang telah diseimbangkan, sehingga sekarang nilai ‘no’ (0) dan ‘yes’ (1) telah seimbang menjadi 25562 baris data.

7. Data Splitting

Sebelum dimasukkan ke dalam model klasifikasi, data harus dipisahkan/dipotong menjadi train set dan test set. Ada 4 dataset yang akan dihasilkan, yaitu X_train dan y_train, serta X_test dan y_test. Pemisahan dilakukan dengan menggunakan test_size = 30%, yaitu 30% dari data akan dialokasikan ke dataset uji (X_test dan y_test), sedangkan 70% sisanya akan digunakan sebagai dataset pelatihan (X_train dan y_train). Data pelatihan (X_train dan y_train) akan digunakan untuk melatih model, dan data uji (X_test dan y_test) akan digunakan untuk menguji kinerja model tersebut.

```
[ ] # Melakukan splitting pada data X_balanced dan y_balanced
X_train, X_test, y_train, y_test = train_test_split(X_balanced,
                                                    y_balanced,
                                                    test_size=0.3,
                                                    random_state=42,
                                                    stratify=y_balanced)
```

Gambar 35. Proses Memotong Dataset menjadi Train Set dan Test Set

8. Feature Scaling

Proses terakhir sebelum data dipakai ke dalam model klasifikasi adalah melakukan proses feature scaling. Feature scaling digunakan untuk menyamakan skala pada data numerik di dataset agar nilainya tidak terlalu jauh dibandingkan data lainnya. Variabel apa saja yang perlu dilakukan feature scaling dapat dievaluasi dari nilai persebaran nilainya.

```
[ ] X_balanced.describe()
```

	age	job	marital	education	default	balance	housing	loan	contact	day
count	51124.000000	51124.000000	51124.000000	51124.000000	51124.000000	51124.000000	51124.000000	51124.000000	51124.000000	51124.000000
mean	40.786773	4.558231	1.214576	3.247183	0.013066	1287.652746	0.419020	0.130604	0.076579	15.451354
std	11.441988	3.203159	0.625284	0.645377	0.113560	1870.495440	0.493404	0.336970	0.265924	8.375688
min	18.000000	0.000000	0.000000	2.000000	0.000000	-8019.000000	0.000000	0.000000	0.000000	1.000000
25%	32.000000	1.000000	1.000000	3.000000	0.000000	126.000000	0.000000	0.000000	0.000000	8.000000
50%	38.000000	4.000000	1.000000	3.000000	0.000000	550.000000	0.000000	0.000000	0.000000	15.000000
75%	49.000000	7.000000	2.000000	4.000000	0.000000	1711.000000	1.000000	0.000000	0.000000	21.000000
max	74.000000	10.000000	2.000000	4.000000	1.000000	10995.000000	1.000000	1.000000	1.000000	31.000000

Gambar 36. Persebaran Nilai Setiap Variabel

Berdasarkan sebaran nilainya, didapatkan informasi bahwa variabel ‘age’, ‘balance’, ‘duration’, dan ‘pdays’ memiliki sebaran nilai yang jauh dibandingkan nilai dari variabel lainnya. Sehingga keempat variabel ini perlu dilakukan scaling. Berdasarkan bentuk histogram dari keempat variabel tersebut di bagian Analisis Univariat, didapatkan informasi bahwa bentuk sebaran dari variabel ‘age’ mengikuti bentuk distribusi normal, dan tidak untuk ketiga variabel lainnya. Sehingga variabel ‘age’ akan dilakukan Standard Scaling menggunakan StandardScaler dari Scikit-Learn, sedangkan ketiga variabel lainnya akan dilakukan Min-Max Scaling menggunakan MinMaxScaler yang juga dari Scikit-Learn.

```

scaler = StandardScaler()

X_train['age'] = scaler.fit_transform(X_train[['age']])
X_test['age'] = scaler.transform(X_test[['age']])

scaler = MinMaxScaler()

X_train['balance'] = scaler.fit_transform(X_train[['balance']])
X_test['balance'] = scaler.transform(X_test[['balance']])

X_train['duration'] = scaler.fit_transform(X_train[['duration']])
X_test['duration'] = scaler.transform(X_test[['duration']])

X_train['pdays'] = scaler.fit_transform(X_train[['pdays']])
X_test['pdays'] = scaler.transform(X_test[['pdays']])

```

Gambar 37. Melakukan Proses Feature Scaling

	age	job	marital	education	default	balance	housing	loan	contact	day
count	3.578600e+04	35786.000000	35786.000000	35786.000000	35786.000000	35786.000000	35786.000000	35786.000000	35786.000000	35786.000000
mean	-1.417670e-16	4.562315	1.215391	3.246214	0.013134	0.489219	0.418599	0.131225	0.076594	15.446068
std	1.000014e+00	3.200049	0.625877	0.644613	0.113849	0.098071	0.493336	0.337650	0.265950	8.376368
min	-1.991412e+00	0.000000	0.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	-7.696023e-01	1.000000	1.000000	3.000000	0.000000	0.428211	0.000000	0.000000	0.000000	8.000000
50%	-2.459696e-01	4.000000	1.000000	3.000000	0.000000	0.450563	0.000000	0.000000	0.000000	15.000000
75%	7.140239e-01	7.000000	2.000000	4.000000	0.000000	0.511413	1.000000	0.000000	0.000000	21.000000
max	2.895827e+00	10.000000	2.000000	4.000000	1.000000	1.000000	1.000000	1.000000	1.000000	31.000000

Gambar 38. Persebaran Nilai Setiap Variabel Setelah Dilakukan Feature Scaling pada Empat Variabel

Dengan melakukan feature scaling pada keempat variabel tersebut, maka seluruh variabel telah memiliki skala yang tidak terlalu jauh berbeda.

9. Model Klasifikasi

a. Decision Tree

Model klasifikasi yang pertama yaitu model klasifikasi Decision Tree. Model ini menggunakan class `DecisionTreeClassifier()` dari library Scikit-Learn. Berikut ini merupakan parameter-parameter yang kami gunakan dalam model klasifikasi Decision Tree:

- `criterion='gini'`, yaitu parameter yang menentukan metode perhitungan pemisahan node selama pembentukan tree. 'gini' mengacu pada Gini Impurity.
- `splitter='best'`, yaitu parameter yang digunakan untuk menentukan kriteria pemisahan node. 'best' berarti pemisahan node akan dipilih berdasarkan kriteria yang paling baik (contoh, Gini impurity yang paling rendah).
- `max_depth=None`, yaitu kedalaman maksimum yang diizinkan untuk tree. Nilai None berarti tree akan diperluas sampai semua leaf berisi data yang unik berjumlah kurang dari `min_samples_split`.
- `min_samples_split=2`, yaitu jumlah sampel minimum yang diperlukan untuk membagi node.
- `min_samples_leaf=1`, yaitu jumlah sampel minimum yang diperlukan untuk membuat leaf.

- max_features=None, yaitu jumlah fitur maksimum yang ikut dipertimbangkan saat membuat pemisahan terbaik. None berarti semua fitur akan dipertimbangkan.

Parameter-parameter tersebut dipilih karena dapat menghasilkan nilai F1-Score dan AUC-ROC yang lebih tinggi dari parameter-parameter lainnya setelah dilakukan beberapa kali percobaan. Berikut ini merupakan script Python yang digunakan untuk pembuatan instance Decision Tree:

```

kfold_set = pd.DataFrame(index=range(1, 11), columns=['train', 'test'])

dtree = DecisionTreeClassifier(criterion='gini',
                                splitter='best',
                                max_depth=None,
                                min_samples_split=2,
                                min_samples_leaf=1,
                                max_features=None)

dtree_roc_auc_scores = []
dtree_f1_scores = []
fold_num = 1

for train_index, test_index in skf.split(X_train, y_train):
    train_features = X_train.iloc[train_index]
    test_features = X_train.iloc[test_index]

    train_labels = y_train.iloc[train_index]
    test_labels = y_train.iloc[test_index]

    dtree.fit(train_features, train_labels)
    pred_labels = dtree.predict(test_features)

    f1 = f1_score(test_labels, pred_labels)
    roc_auc = roc_auc_score(test_labels, pred_labels)

    dtree_f1_scores.append(f1)
    dtree_roc_auc_scores.append(roc_auc)

    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'test'] = test_index

    print(f"F1-score for fold {fold_num}: ", f1)
    print(f"ROC-AUC score for fold {fold_num}: ", roc_auc, '\n')

    fold_num += 1

```

Gambar 39. Model Klasifikasi Decision Tree

Selanjutnya, Decision Tree akan melatih modelnya menggunakan data train set yang telah dibagi menjadi 10 fold oleh K-Fold Cross Validation. F1-Score dan AUC-ROC dari setiap fold dihitung untuk mengetahui kinerja dari model Decision Tree yang telah dilatih menggunakan setiap fold. Berikut ini merupakan F1-Score dan AUC-ROC dari model Decision Tree yang telah dilatih menggunakan setiap fold.

```

F1-score for fold 1: 0.9427359490986214
ROC-AUC score for fold 1: 0.9396630869591015

F1-score for fold 2: 0.9482666666666666
ROC-AUC score for fold 2: 0.9458083383557495

F1-score for fold 3: 0.9476481530693595
ROC-AUC score for fold 3: 0.9449711302153758

F1-score for fold 4: 0.9519487453283503
ROC-AUC score for fold 4: 0.9496936586401692

F1-score for fold 5: 0.9482666666666667
ROC-AUC score for fold 5: 0.9457816388794339

F1-score for fold 6: 0.9488545551411828
ROC-AUC score for fold 6: 0.9463401419600226

F1-score for fold 7: 0.9483542948889483
ROC-AUC score for fold 7: 0.9460592509782001

F1-score for fold 8: 0.9436358825086002
ROC-AUC score for fold 8: 0.9404695360536612

F1-score for fold 9: 0.947481243301179
ROC-AUC score for fold 9: 0.9452207937395193

F1-score for fold 10: 0.9461333333333333
ROC-AUC score for fold 10: 0.9435438792621578

```

Gambar 40. F1-Score dan AUC-ROC dari 10 Fold Decision Tree

Setelah didapatkan F1-Score dan AUC-ROC dari setiap fold, langkah selanjutnya yaitu mengurutkan fold berdasarkan nilai rerata antara F1-Score dan AUC-ROC yang paling baik. Tujuannya untuk mengetahui nomor fold yang bisa menghasilkan nilai terbaik.

```

[ ] # Mengurutkan fold dari yang memiliki nilai rata-rata F1 dan AUC-ROC terbesar ke terkecil
fold_number = []
fold_score_mean = []

for i in range(0, len(dtree_roc_auc_scores)):
    number = i + 1
    score_mean = (dtree_roc_auc_scores[i] + dtree_f1_scores[i]) / 2
    fold_number.append(number)
    fold_score_mean.append(score_mean)

fsm = pd.DataFrame({'fold_number': fold_number, 'fold_score_mean': fold_score_mean})
fsm_sort = fsm.sort_values(by='fold_score_mean', ascending=False).reset_index(drop=True)
fsm_sort

```

	fold_number	fold_score_mean
0	4	0.950821
1	6	0.947597
2	7	0.947207
3	2	0.947038
4	5	0.947024
5	9	0.946351
6	3	0.946310
7	10	0.944839
8	8	0.942053
9	1	0.941200

Gambar 41. Rerata F1-Score dan AUC-ROC Decision Tree Per Fold

```
[ ] best_fold = fsm_sort.loc[0, 'fold_number']
print(f"Model klasifikasi Decission Tree terbaik menggunakan fold ke-{best_fold}")

Model klasifikasi Decission Tree terbaik menggunakan fold ke-4
```

Gambar 42. Fold Terbaik Decision Tree

Setelah diurutkan, didapatkan fold yang paling baik dalam melatih Decision Tree adalah fold nomor 4 dengan rerata F1-Score dan AUC-ROC sebesar 0,950821. Model Decision Tree yang telah dilatih dengan fold tersebut selanjutnya akan digunakan untuk memprediksi nilai F1-Score dan AUC-ROC pada test set (X_{test} dan y_{test}) dan didapatkan hasil yang tidak jauh berbeda dengan hasil dari train set.

```
▶ X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

dtree.fit(X_best_fold, y_best_fold)
y_pred = dtree.predict(X_test)

f1_test = f1_score(y_test, y_pred)
auc roc test = roc_auc_score(y_test, y_pred)

print("F1-Score: {f1_test}")
print("AUC-ROC Score: {auc roc test}")
```

🕒 F1-Score: 0.9457431636386184
AUC-ROC Score: 0.942952144999348

Gambar 43. Hasil Prediksi Decision Tree pada Test Set

Setelah mendapatkan informasi mengenai F1-Score dan AUC-ROC semua fold train set serta dari test set, maka informasi tersebut dimasukkan ke dataframe yang merekap keseluruhan hasil dari setiap model klasifikasi Decision Tree seperti pada gambar di bawah ini.

```
[ ] # Mendapatkan mean dan standar deviasi dari seluruh F1-Score dan ROC-AUC
roc_auc_mean = np.mean(dtree_roc_auc_scores)
roc_auc_std = np.std(dtree_roc_auc_scores)
f1_mean = np.mean(dtree_f1_scores)
f1_std = np.std(dtree_f1_scores)
best_fold = fsm_sort.loc[0, 'fold_number']

scores.loc['Decision Tree'] = [f1_mean, roc_auc_mean, f1_std, roc_auc_std, best_fold, f1_test, auc roc test]
scores
```

	F1-Score Mean	ROC-AUC Mean	F1-Score STD	ROC-AUC STD	Best Fold Num	F1-Score Test	ROC-AUC Test
Decision Tree	0.947333	0.944755	0.002505	0.002771	4	0.945743	0.942952
KNN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Naive Bayes	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SVM	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Neural Networks	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Logistic Regression	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Random Forest	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Gambar 44. Rekap Score Decision Tree

b. K-Nearest Neighbors

Model klasifikasi yang pertama yaitu model klasifikasi K-Nearest Neighbors. Model ini menggunakan class KNeighborsClassifier() dari library Scikit-Learn. Berikut ini merupakan parameter-parameter yang kami gunakan dalam model klasifikasi K-Nearest Neighbors:

- `n_neighbors=3`, yaitu parameter yang menentukan jumlah tetangga terdekat yang akan digunakan dalam prediksi. Dalam contoh ini, model K-Nearest Neighbors akan mempertimbangkan 3 tetangga terdekat.
- `weights='distance'`, berarti bahwa tetangga yang lebih dekat akan memiliki pengaruh yang lebih besar dalam prediksi daripada yang lebih jauh.
- `metric='minkowski'`, yaitu metrik yang digunakan untuk mengukur jarak antara titik data.
- `p=2`, yaitu parameter yang menentukan jenis metrik yang akan digunakan. Dalam hal ini, `p=2` mengacu pada metrik Euclidean, yang digunakan untuk mengukur jarak antara titik data.

Parameter-parameter tersebut dipilih karena dapat menghasilkan nilai F1-Score dan AUC-ROC yang lebih tinggi dari parameter-parameter lainnya setelah dilakukan beberapa kali percobaan. Berikut ini merupakan script Python yang digunakan untuk pembuatan instance K-Nearest Neighbors:

```
[ ] kfold_set = pd.DataFrame(index=range(1, 11), columns=['train', 'test'])

knn = KNeighborsClassifier(n_neighbors=3,
                           weights='distance',
                           metric='minkowski',
                           p=2)

knn_roc_auc_scores = []
knn_f1_scores = []
fold_num = 1

for train_index, test_index in skf.split(X_train, y_train):
    train_features = X_train.iloc[train_index]
    test_features = X_train.iloc[test_index]

    train_labels = y_train.iloc[train_index]
    test_labels = y_train.iloc[test_index]

    knn.fit(train_features, train_labels)
    pred_labels = knn.predict(test_features)

    f1 = f1_score(test_labels, pred_labels)
    roc_auc = roc_auc_score(test_labels, pred_labels)

    knn_f1_scores.append(f1)
    knn_roc_auc_scores.append(roc_auc)

    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'test'] = test_index

    print(f"F1-score for fold {fold_num}: ", f1)
    print(f"ROC-AUC score for fold {fold_num}: ", roc_auc, '\n')

    fold_num += 1
```

Gambar 45. Model Klasifikasi K-Nearest Neighbors

Selanjutnya, K-Nearest Neighbors akan melatih modelnya menggunakan data train set yang telah dibagi menjadi 10 fold oleh K-Fold Cross Validation. F1-Score dan AUC-ROC dari setiap fold dihitung untuk mengetahui kinerja dari model K-Nearest Neighbors yang telah dilatih menggunakan setiap fold. Berikut ini merupakan F1-Score dan AUC-ROC dari model K-Nearest Neighbors yang telah dilatih menggunakan setiap fold.

```

F1-score for fold 1: 0.9207741935483871
ROC-AUC score for fold 1: 0.914245029369424

F1-score for fold 2: 0.919659002841643
ROC-AUC score for fold 2: 0.9131270863845162

F1-score for fold 3: 0.9174832387828777
ROC-AUC score for fold 3: 0.9106129637667809

F1-score for fold 4: 0.9137577002053389
ROC-AUC score for fold 4: 0.9060943506406313

F1-score for fold 5: 0.9140805334701204
ROC-AUC score for fold 5: 0.9063735241122814

F1-score for fold 6: 0.9124967924044136
ROC-AUC score for fold 6: 0.9046972341840734

F1-score for fold 7: 0.924954557257855
ROC-AUC score for fold 7: 0.9192286193404136

F1-score for fold 8: 0.9147843942505133
ROC-AUC score for fold 8: 0.907210732252655

F1-score for fold 9: 0.9115681233933162
ROC-AUC score for fold 9: 0.9038569032979319

F1-score for fold 10: 0.9118400820092261
ROC-AUC score for fold 10: 0.9038569032979319

```

Gambar 46. F1-Score dan AUC-ROC dari 10 Fold K-Nearest Neighbors

Setelah didapatkan F1-Score dan AUC-ROC dari setiap fold, langkah selanjutnya yaitu mengurutkan fold berdasarkan nilai rerata antara F1-Score dan AUC-ROC yang paling baik. Tujuannya untuk mengetahui nomor fold yang bisa menghasilkan nilai terbaik.

```

[ ] # Mengurutkan fold dari yang memiliki nilai rata-rata F1 dan AUC-ROC terbesar ke terkecil
fold_number = []
fold_score_mean = []

for i in range(0, len(knn_roc_auc_scores)):
    number = i + 1
    score_mean = (knn_roc_auc_scores[i] + knn_f1_scores[i]) / 2
    fold_number.append(number)
    fold_score_mean.append(score_mean)

fsm = pd.DataFrame({'fold_number': fold_number, 'fold_score_mean': fold_score_mean})
fsm_sort = fsm.sort_values(by='fold_score_mean', ascending=False).reset_index(drop=True)
fsm_sort

```

	fold_number	fold_score_mean
0	7	0.922092
1	1	0.917510
2	2	0.916393
3	3	0.914048
4	8	0.910998
5	5	0.910227
6	4	0.909926
7	6	0.908597
8	10	0.907848
9	9	0.907713

Gambar 47. Rerata F1-Score dan AUC-ROC K-Nearest Neighbors Per Fold

```
[ ] best_fold = fsm_sort.loc[0, 'fold_number']
print(f"Model klasifikasi K-Nearest Neighbors terbaik menggunakan fold ke-{best_fold}")

Model klasifikasi K-Nearest Neighbors terbaik menggunakan fold ke-7
```

Gambar 48. Fold Terbaik K-Nearest Neighbors

Setelah diurutkan, didapatkan fold yang paling baik dalam melatih K-Nearest Neighbors adalah fold nomor 7 dengan rerata F1-Score dan AUC-ROC sebesar 0,922092. Model K-Nearest Neighbors yang telah dilatih dengan fold tersebut selanjutnya akan digunakan untuk memprediksi nilai F1-Score dan AUC-ROC pada test set (X_{test} dan y_{test}) dan didapatkan hasil yang tidak jauh berbeda dengan hasil dari train set.

```
[ ] X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

knn.fit(X_best_fold, y_best_fold)
y_pred = knn.predict(X_test)
f1_test = f1_score(y_test, y_pred)
auc_roc_test = roc_auc_score(y_test, y_pred)

print(f"F1-Score: {f1_test}")
print(f"AUC-ROC Score: {auc_roc_test}")

F1-Score: 0.9152501651750856
AUC-ROC Score: 0.9080062589646629
```

Gambar 49. Hasil Prediksi K-Nearest Neighbors pada Test Set

Setelah mendapatkan informasi mengenai F1-Score dan AUC-ROC semua fold train set serta dari test set, maka informasi tersebut dimasukkan ke dataframe yang merekap keseluruhan hasil dari setiap model klasifikasi K-Nearest Neighbors seperti pada gambar di bawah ini.

```
[ ] # Mendapatkan mean dan standar deviasi dari seluruh F1-Score dan ROC-AUC
roc_auc_mean = np.mean(knn_roc_auc_scores)
roc_auc_std = np.std(knn_roc_auc_scores)
f1_mean = np.mean(knn_f1_scores)
f1_std = np.std(knn_f1_scores)
best_fold = fsm_sort.loc[0, 'fold_number']

scores.loc['KNN'] = [f1_mean, roc_auc_mean, f1_std, roc_auc_std, best_fold, f1_test, auc_roc_test]
scores
```

	F1-Score Mean	ROC-AUC Mean	F1-Score STD	ROC-AUC STD	Best Fold Num	F1-Score Test	ROC-AUC Test
Decision Tree	0.947333	0.944755	0.002505	0.002771	4	0.945743	0.942952
KNN	0.91614	0.90893	0.004216	0.004916	7	0.91525	0.908006
Naive Bayes	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SVM	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Neural Networks	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Logistic Regression	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Random Forest	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Gambar 50. Rekap Score K-Nearest Neighbors

c. Naive Bayes

Model klasifikasi berikutnya yaitu model klasifikasi Naive Bayes. Model Naive Bayes sendiri adalah model klasifikasi yang mengasumsikan bahwa setiap fitur dalam dataset adalah independen secara kondisional terhadap kelas target. Berikut merupakan pemanggilan class Naive Bayes dengan mengimpor GaussianNB(). Selanjutnya Naive Bayes akan melatih modelnya menggunakan data train set yang telah dibagi menjadi 10 fold oleh K-Fold Cross Validation. F1-Score dan AUC-ROC dari setiap fold dihitung untuk mengetahui kinerja dari model Naive Bayes yang telah dilatih oleh setiap fold.

```
[ ] kfolds_set = pd.DataFrame(index=range(1, 11), columns=['train', 'test'])

nbc = GaussianNB()

nbc_roc_auc_scores = []
nbc_f1_scores = []
fold_num = 1

for train_index, test_index in skf.split(X_train, y_train):
    train_features = X_train.iloc[train_index]
    test_features = X_train.iloc[test_index]

    train_labels = y_train.iloc[train_index]
    test_labels = y_train.iloc[test_index]

    nbc.fit(train_features, train_labels)
    pred_labels = nbc.predict(test_features)

    f1 = f1_score(test_labels, pred_labels)
    roc_auc = roc_auc_score(test_labels, pred_labels)

    nbc_f1_scores.append(f1)
    nbc_roc_auc_scores.append(roc_auc)

    kfolds_set.at[fold_num, 'train'] = train_index
    kfolds_set.at[fold_num, 'test'] = test_index

    print(f"F1-score for fold {fold_num}: ", f1)
    print(f"ROC-AUC score for fold {fold_num}: ", roc_auc, '\n')

    fold_num += 1
```

Gambar 51. Model Klasifikasi Naive Bayes

Berikut ini merupakan F1-Score dan AUC-ROC dari setiap fold.

```

F1-score for fold 1: 0.7675257731958763
ROC-AUC score for fold 1: 0.7479978515509118

F1-score for fold 2: 0.7538779731127199
ROC-AUC score for fold 2: 0.734026530848044

F1-score for fold 3: 0.7650130548302871
ROC-AUC score for fold 3: 0.7485527634738673

F1-score for fold 4: 0.762519359834796
ROC-AUC score for fold 4: 0.7429219844424806

F1-score for fold 5: 0.759467758444217
ROC-AUC score for fold 5: 0.7373311765569229

F1-score for fold 6: 0.7555669311492192
ROC-AUC score for fold 6: 0.733140139461826

F1-score for fold 7: 0.7522032141005702
ROC-AUC score for fold 7: 0.732811626607043

F1-score for fold 8: 0.756141711921386
ROC-AUC score for fold 8: 0.7364449413079933

F1-score for fold 9: 0.7666494579246257
ROC-AUC score for fold 9: 0.7473448854108441

F1-score for fold 10: 0.7631103074141048
ROC-AUC score for fold 10: 0.7437115707098938

```

Gambar 52. F1-Score dan AUC-ROC dari 10 Fold Naive Bayes

Setelah didapatkan F1-Score dan AUC-ROC dari setiap fold, langkah selanjutnya yaitu mengurutkan fold berdasarkan nilai rerata antara F1-Score dan AUC-ROC yang paling baik. Tujuannya untuk mengetahui nomor fold yang bisa menghasilkan nilai terbaik.

```

[ ] # Mengurutkan fold dari yang memiliki nilai rata-rata F1 dan AUC-ROC terbesar ke terkecil
fold_number = []
fold_score_mean = []

for i in range(0, len(nbc_roc_auc_scores)):
    number = i + 1
    score_mean = (nbc_roc_auc_scores[i] + nbc_f1_scores[i]) / 2
    fold_number.append(number)
    fold_score_mean.append(score_mean)

fsm = pd.DataFrame({'fold_number': fold_number, 'fold_score_mean': fold_score_mean})
fsm_sort = fsm.sort_values(by='fold_score_mean', ascending=False).reset_index(drop=True)
fsm_sort

```

	fold_number	fold_score_mean
0	1	0.757762
1	9	0.756997
2	3	0.756783
3	10	0.753411
4	4	0.752721
5	5	0.748399
6	8	0.746293
7	6	0.744354
8	2	0.743952
9	7	0.742507

Gambar 53. Mengurutkan Fold dari Rerata F1-Score dan AUC-ROC tertinggi

```
[ ] best_fold = fsm_sort.loc[0, 'fold_number']
print(f"Model klasifikasi Naive Bayes terbaik menggunakan fold ke-{best_fold}")

Model klasifikasi Naive Bayes terbaik menggunakan fold ke-1
```

Gambar 54. Hasil Fold Terpilih Naive Bayes

Setelah diurutkan, didapatkan fold yang paling baik dalam melatih Naive Bayes yaitu fold nomor 1 dengan rerata F1-Score dan AUC-ROC sebesar 0.757762. Model Naive Bayes yang telah dilatih dengan fold tersebut selanjutnya akan digunakan untuk memprediksi nilai F1-Score dan AUC-ROC pada test set (X_{test} dan y_{test}). Didapatkan hasil yang tidak jauh berbeda dengan hasil dari train set.

```
[ ] X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

nbc.fit(X_best_fold, y_best_fold)
y_pred = nbc.predict(X_test)
f1_test = f1_score(y_test, y_pred)
auc_roc_test = roc_auc_score(y_test, y_pred)

print(f"F1-Score: {f1_test}")
print(f"AUC-ROC Score: {auc_roc_test}")
```

```
F1-Score: 0.7564025489960321
AUC-ROC Score: 0.7358195331855523
```

Gambar 55. Hasil Prediksi Naive Bayes pada Test Set

Setelah mendapatkan informasi mengenai F1-Score dan AUC-ROC semua fold train set serta dari test set, maka informasi tersebut dimasukkan ke dataframe yang merekap keseluruhan hasil dari setiap model klasifikasi. Karena pada bagian ini menggunakan model klasifikasi Naive Bayes, maka informasi Mean dan Standar Deviasi masuk ke baris Naive Bayes seperti berikut.

```
[ ] # Mendapatkan mean dan standar deviasi dari seluruh F1-Score dan ROC-AUC
roc_auc_mean = np.mean(nbc_roc_auc_scores)
roc_auc_std = np.std(nbc_roc_auc_scores)
f1_mean = np.mean(nbc_f1_scores)
f1_std = np.std(nbc_f1_scores)
best_fold = fsm_sort.loc[0, 'fold_number']

scores.loc['Naive Bayes'] = [f1_mean, roc_auc_mean, f1_std, roc_auc_std, best_fold, f1_test, auc_roc_test]
scores
```

	F1-Score Mean	ROC-AUC Mean	F1-Score STD	ROC-AUC STD	Best Fold Num	F1-Score Test	ROC-AUC Test
Decision Tree	0.947042	0.94442	0.002308	0.002587	6	0.944689	0.941779
KNN	0.91614	0.90893	0.004216	0.004916	7	0.91525	0.908006
Naive Bayes	0.760208	0.740428	0.005241	0.006047	1	0.756403	0.73582
SVM	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Neural Networks	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Logistic Regression	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Random Forest	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Gambar 56. Memasukkan Rekap Hasil Naive Bayes ke Dalam Dataframe Rekap Score

d. Support Vector Machine

Model klasifikasi berikutnya yaitu model klasifikasi Support Vector Machine. Model Support Vector Machine sendiri adalah algoritma pembelajaran mesin yang digunakan untuk tugas klasifikasi dan regresi. Berikut merupakan pemanggilan class Support Vector Machine dengan mengimpor SVC() menggunakan parameter C=1.0 yang diartikan sebagai bobot yang seimbang antara mencapai margin maksimum dan mengurangi kesalahan klasifikasi pada data pelatihan dan parameter dual=false yang diartikan sebagai formulasi primal untuk masalah SVM .

Selanjutnya Support Vector Machine akan melatih modelnya menggunakan data train set yang telah dibagi menjadi 10 fold oleh K-Fold Cross Validation. F1-Score dan AUC-ROC dari setiap fold dihitung untuk mengetahui kinerja dari model Support vector Machine yang telah dilatih oleh setiap fold.

```
[ ] kfolds_set = pd.DataFrame(index=range(1, 11), columns=['train', 'test'])

# svm = SVC(kernel='linear')
svm = LinearSVC(C=1.0, dual=False)

svm_f1_scores = []
svm_roc_auc_scores = []
fold_num = 1

for train_index, test_index in skf.split(X_train, y_train):
    train_features = X_train.iloc[train_index]
    test_features = X_train.iloc[test_index]

    train_labels = y_train.iloc[train_index]
    test_labels = y_train.iloc[test_index]

    svm.fit(train_features, train_labels)
    pred_labels = svm.predict(test_features)

    f1 = f1_score(test_labels, pred_labels)
    roc_auc = roc_auc_score(test_labels, pred_labels)

    svm_f1_scores.append(f1)
    svm_roc_auc_scores.append(roc_auc)

    kfolds_set.at[fold_num, 'train'] = train_index
    kfolds_set.at[fold_num, 'test'] = test_index

    print(f"F1-score for fold {fold_num}: ", f1)
    print(f"ROC-AUC score for fold {fold_num}: ", roc_auc, '\n')

    fold_num += 1
```

Gambar 57. Model Klasifikasi Support Vector Machine

Berikut ini merupakan F1-Score dan AUC-ROC dari setiap fold.

```

F1-score for fold 1: 0.7705799151343705
ROC-AUC score for fold 1: 0.7733970165286934

F1-score for fold 2: 0.7798920761147402
ROC-AUC score for fold 2: 0.7834545999606535

F1-score for fold 3: 0.7762039660056658
ROC-AUC score for fold 3: 0.77926418741471

F1-score for fold 4: 0.7786259541984735
ROC-AUC score for fold 4: 0.7812271454044112

F1-score for fold 5: 0.765237020316027
ROC-AUC score for fold 5: 0.7675356227223473

F1-score for fold 6: 0.765704584040747
ROC-AUC score for fold 6: 0.7686540341191203

F1-score for fold 7: 0.7751277683134583
ROC-AUC score for fold 7: 0.7786472889882615

F1-score for fold 8: 0.7662227259846981
ROC-AUC score for fold 8: 0.7694242593627725

F1-score for fold 9: 0.7760683760683761
ROC-AUC score for fold 9: 0.7803242034656231

F1-score for fold 10: 0.7810344827586208
ROC-AUC score for fold 10: 0.78703186137507

```

Gambar 58. F1-Score dan AUC-ROC dari 10 Fold Support Vector Machine

Setelah didapatkan F1-Score dan AUC-ROC dari setiap fold, langkah selanjutnya yaitu mengurutkan fold berdasarkan nilai rerata antara F1-Score dan AUC-ROC yang paling baik. Tujuannya untuk mengetahui nomor fold yang bisa menghasilkan nilai terbaik.

```

[ ] # Mengurutkan fold dari yang memiliki nilai rata-rata F1 dan AUC-ROC terbesar ke terkecil
fold_number = []
fold_score_mean = []

for i in range(0, len(svm_roc_auc_scores)):
    number = i + 1
    score_mean = (svm_roc_auc_scores[i] + svm_f1_scores[i]) / 2
    fold_number.append(number)
    fold_score_mean.append(score_mean)

fsm = pd.DataFrame({'fold_number': fold_number, 'fold_score_mean': fold_score_mean})
fsm_sort = fsm.sort_values(by='fold_score_mean', ascending=False).reset_index(drop=True)
fsm_sort

```

fold_number	fold_score_mean
0	0.784033
1	0.781673
2	0.779927
3	0.778196
4	0.777734
5	0.776888
6	0.771988
7	0.767823
8	0.767179
9	0.766386

Gambar 59. Mengurutkan Fold dari Rerata F1-Score dan AUC-ROC tertinggi

```
[ ] best_fold = fsm_sort.loc[0, 'fold_number']
print(f"Model klasifikasi Support Vector Machine terbaik menggunakan fold ke-{best_fold}")

Model klasifikasi Support Vector Machine terbaik menggunakan fold ke-10
```

Gambar 60. Hasil Fold Terpilih Support Vector Machine

Setelah diurutkan, didapatkan fold yang paling baik dalam melatih Support Vector Machine yaitu fold nomor 10 dengan rerata F1-Score dan AUC-ROC sebesar 0.784033. Model Support Vector Machine yang telah dilatih dengan fold tersebut selanjutnya akan digunakan untuk memprediksi nilai F1-Score dan AUC-ROC pada test set (X_{test} dan y_{test}). Didapatkan hasil yang tidak jauh berbeda dengan hasil dari train set.

```
[ ] X_best_fold = X_train.iloc[kfold_set['train']].loc[best_fold]
y_best_fold = y_train.iloc[kfold_set['train']].loc[best_fold]

svm.fit(X_best_fold, y_best_fold)
y_pred = svm.predict(X_test)
f1_test = f1_score(y_test, y_pred)
auc_roc_test = roc_auc_score(y_test, y_pred)

print(F1-Score: {f1_test})
print(AUC-ROC Score: {auc_roc_test})
```

F1-Score: 0.7692812602559895
AUC-ROC Score: 0.7708306167688095

Gambar 61. Hasil Prediksi Support Vector Machine pada Test Set

Setelah mendapatkan informasi mengenai F1-Score dan AUC-ROC semua fold train set serta dari test set, maka informasi tersebut dimasukkan ke dataframe yang merekap keseluruhan hasil dari setiap model klasifikasi. Karena pada bagian ini menggunakan model klasifikasi Support Vector Machine, maka informasi Mean dan Standar Deviasi masuk ke baris Support Vector Machine seperti berikut.

```
[ ] # Mendapatkan mean dan standar deviasi dari seluruh F1-Score dan ROC-AUC
roc_auc_mean = np.mean(svm_roc_auc_scores)
roc_auc_std = np.std(svm_roc_auc_scores)
f1_mean = np.mean(svm_f1_scores)
f1_std = np.std(svm_f1_scores)
best_fold = fsm_sort.loc[0, 'fold_number']

scores.loc['SVM'] = [f1_mean, roc_auc_mean, f1_std, roc_auc_std, best_fold, f1_test, auc_roc_test]
```

	F1-Score Mean	ROC-AUC Mean	F1-Score STD	ROC-AUC STD	Best Fold Num	F1-Score Test	ROC-AUC Test
Decision Tree	0.94733	0.944755	0.002505	0.002771	4	0.945743	0.942952
KNN	0.91614	0.90893	0.004216	0.004916	7	0.91525	0.908006
Naive Bayes	0.760208	0.740428	0.005241	0.006047	1	0.756403	0.73582
SVM	0.77347	0.776896	0.005753	0.006392	10	0.769281	0.770831
Neural Networks	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Logistic Regression	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Random Forest	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Gambar 62. Memasukkan Rekap Hasil Support Vector Machine ke Dalam Dataframe Rekap Score

e. Neural Network

Model klasifikasi berikutnya yaitu model klasifikasi Neural Network. Di bawah ini merupakan kode pembuatan instance Neural Network dengan menggunakan class MLPClassifier() dari library Scikit-Learn. Ada berbagai parameter di dalam Neural Network milik MLPClassifier() yang dapat digunakan. Pada pembuatan instance Neural Network ini, kami mencoba menggunakan 1 hidden layer dengan 100 node/neuron di dalamnya. Activation function yang digunakan pada hidden layer tersebut adalah ‘relu’, dengan jenis learning rate ‘constant’ dan bernilai 0,001.

Selanjutnya Neural Network akan melatih modelnya menggunakan data train set yang telah dibagi menjadi 10 fold oleh K-Fold Cross Validation. F1-Score dan AUC-ROC dari setiap fold dihitung untuk mengetahui kinerja dari model Neural Network yang telah dilatih menggunakan setiap fold.

Kami telah mencoba menggunakan jenis activation function dan jenis learning rate yang berbeda, namun memberikan hasil yang lebih rendah.

```
[ ] kfold_set = pd.DataFrame(index=range(1, 11), columns=['train', 'test'])

nn = MLPClassifier(hidden_layer_sizes=(100),
                    activation='relu',
                    learning_rate='constant',
                    learning_rate_init=0.001,
                    max_iter=500,
                    early_stopping=True)

nn_roc_auc_scores = []
nn_f1_scores = []
fold_num = 1

for train_index, test_index in skf.split(X_train, y_train):
    train_features = X_train.iloc[train_index]
    test_features = X_train.iloc[test_index]

    train_labels = y_train.iloc[train_index]
    test_labels = y_train.iloc[test_index]

    nn.fit(train_features, train_labels)
    pred_labels = nn.predict(test_features)

    f1 = f1_score(test_labels, pred_labels)
    roc_auc = roc_auc_score(test_labels, pred_labels)

    nn_f1_scores.append(f1)
    nn_roc_auc_scores.append(roc_auc)

    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'test'] = test_index

    print(f"F1-score for fold {fold_num}: ", f1)
    print(f"ROC-AUC score for fold {fold_num}: ", roc_auc, '\n')

    fold_num += 1
```

Gambar 63. Model Klasifikasi Neural Network

Berikut ini merupakan F1-Score dan AUC-ROC dari model Neural Network yang telah dilatih menggunakan setiap fold.

```
F1-score for fold 1: 0.8330134357005757
ROC-AUC score for fold 1: 0.829846111088558

F1-score for fold 2: 0.8425391591096455
ROC-AUC score for fold 2: 0.8399041629323831

F1-score for fold 3: 0.8395265620699146
ROC-AUC score for fold 3: 0.8371096177446906

F1-score for fold 4: 0.8506811989100816
ROC-AUC score for fold 4: 0.846877566506678

F1-score for fold 5: 0.830461367426348
ROC-AUC score for fold 5: 0.82955991143893

F1-score for fold 6: 0.8327572454134539
ROC-AUC score for fold 6: 0.8242384403758537

F1-score for fold 7: 0.8423349699945445
ROC-AUC score for fold 7: 0.8384572386808273

F1-score for fold 8: 0.8222709818979703
ROC-AUC score for fold 8: 0.8188932364449412

F1-score for fold 9: 0.840595399188092
ROC-AUC score for fold 9: 0.8353828954723309

F1-score for fold 10: 0.8343291689299295
ROC-AUC score for fold 10: 0.8295136948015651
```

Gambar 64. F1-Score dan AUC-ROC dari 10 Fold Neural Network

Setelah didapatkan F1-Score dan AUC-ROC dari setiap fold, langkah selanjutnya yaitu mengurutkan fold berdasarkan nilai rerata antara F1-Score dan AUC-ROC yang paling baik. Tujuannya untuk mengetahui nomor fold yang bisa menghasilkan nilai terbaik.

```
[ ] # Mengurutkan fold dari yang memiliki nilai rata-rata F1 dan AUC-ROC terbesar ke terkecil
fold_number = []
fold_score_mean = []

for i in range(0, len(nn_roc_auc_scores)):
    number = i + 1
    score_mean = (nn_roc_auc_scores[i] + nn_f1_scores[i]) / 2
    fold_number.append(number)
    fold_score_mean.append(score_mean)

fsm = pd.DataFrame({'fold_number': fold_number, 'fold_score_mean': fold_score_mean})
fsm_sort = fsm.sort_values(by='fold_score_mean', ascending=False).reset_index(drop=True)
fsm_sort
```

	fold_number	fold_score_mean
0	4	0.848779
1	2	0.841222
2	7	0.840396
3	3	0.838318
4	9	0.837989
5	10	0.831921
6	1	0.831430
7	5	0.830011
8	6	0.828498
9	8	0.820582

Gambar 65. Mengurutkan Fold dari Rerata F1-Score dan AUC-ROC tertinggi

```
[ ] best_fold = fsm_sort.loc[0, 'fold_number']
print(f"Model klasifikasi Neural Network terbaik menggunakan fold ke-{best_fold}")
```

Model klasifikasi Neural Network terbaik menggunakan fold ke-4

Gambar 66. Hasil Fold Terpilih Neural Network

Setelah diurutkan, didapatkan fold yang paling baik dalam melatih Neural Network adalah fold nomor 4 dengan rerata F1-Score dan AUC-ROC sebesar 0,848779. Model Neural Network yang telah dilatih dengan fold tersebut selanjutnya akan digunakan untuk memprediksi nilai F1-Score dan AUC-ROC pada test set (X_test dan y_test). Didapatkan hasil yang tidak jauh berbeda dengan hasil dari train set.

```
[ ] X_best_fold = X_train.iloc[kfold_set['train']].loc[best_fold]
y_best_fold = y_train.iloc[kfold_set['train']].loc[best_fold]
```

```
nn.fit(X_best_fold, y_best_fold)
y_pred = nn.predict(X_test)
f1_test = f1_score(y_test, y_pred)
auc_roc_test = roc_auc_score(y_test, y_pred)

print(f"F1-Score: {f1_test}")
print(f"AUC-ROC Score: {auc_roc_test}")
```

F1-Score: 0.8341856302735873
AUC-ROC Score: 0.8308775590037815

Gambar 67. Hasil Prediksi Neural Network pada Test Set

Setelah mendapatkan informasi mengenai F1-Score dan AUC-ROC semua fold train set serta dari test set, maka informasi tersebut dimasukkan ke dataframe yang merekap keseluruhan hasil dari setiap model klasifikasi. Karena pada bagian ini menggunakan model klasifikasi Neural Network, maka informasi mean dan standar deviasi masuk ke baris Neural Network seperti berikut.

```
[ ] # Mendapatkan mean dan standar deviasi dari seluruh F1-Score dan ROC-AUC
roc_auc_mean = np.mean(nn_roc_auc_scores)
roc_auc_std = np.std(nn_roc_auc_scores)
f1_mean = np.mean(nn_f1_scores)
f1_std = np.std(nn_f1_scores)
best_fold = fsm_sort.loc[0, 'fold_number']

scores.loc['Neural Networks'] = [f1_mean, roc_auc_mean, f1_std, roc_auc_std, best_fold, f1_test, auc_roc_test]
scores
```

	F1-Score Mean	ROC-AUC Mean	F1-Score STD	ROC-AUC STD	Best Fold Num	F1-Score Test	ROC-AUC Test
Decision Tree	0.947333	0.944755	0.002505	0.002771	4	0.945743	0.942952
KNN	0.91614	0.90893	0.004216	0.004916	7	0.91525	0.908006
Naive Bayes	0.760208	0.740428	0.005241	0.006047	1	0.756403	0.73582
SVM	0.77347	0.776896	0.005753	0.006392	10	0.769281	0.770831
Neural Networks	0.836851	0.832978	0.007524	0.007761	4	0.834186	0.830878
Logistic Regression	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Random Forest	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Gambar 68. Memasukkan Rekap Hasil Neural Network ke Dalam Dataframe Rekap Score

f. Logistic Regression

Model klasifikasi selanjutnya yaitu model klasifikasi Logistic Regression. Di bawah ini merupakan kode pembuatan instance Logistic Regression dengan menggunakan class LogisticRegression() dari library Scikit-Learn. Berikut ini merupakan parameter-parameter yang kami gunakan dalam model klasifikasi Logistic Regression:

- max_iter=300, yang berarti jumlah iterasi maksimum yang dilakukan oleh solver selama pelatihan.
- dual=false yang diartikan sebagai formulasi primal untuk masalah Logistic Regression.
- penalty='L2', yang merupakan tipe regularisasi yang digunakan dalam model.
- C=1.0 yang diartikan sebagai bobot yang seimbang antara mencapai margin maksimum dan mengurangi kesalahan klasifikasi pada train set.
- solver='lbfgs', yaitu algoritma pelatihan yang digunakan dalam model.
- class_weight=None, pada kasus ini semua kelas dianggap sama pentingnya dan diberi bobot yang sama.
- verbose=0, yang berarti tingkat keluaran selama pelatihan.
- multi_class='auto', yang menandakan bahwa model akan memilih strategi klasifikasi yang sesuai berdasarkan jenis data masukan.

Parameter-parameter tersebut dipilih karena dapat menghasilkan nilai F1-Score dan AUC-ROC yang lebih tinggi dari parameter-parameter lainnya setelah dilakukan setelah beberapa kali kami lakukan percobaan. Selanjutnya, Logistic Regression akan melatih modelnya menggunakan data train set yang telah dibagi menjadi 10 fold oleh K-Fold Cross Validation. F1-Score dan AUC-ROC dari setiap fold dihitung untuk mengetahui kinerja dari model Logistic Regression yang telah dilatih oleh setiap fold.

```

fold_set = pd.DataFrame(index=range(1, 11), columns=['train', 'test'])

logreg = LogisticRegression(max_iter=300,
                            dual=False,
                            penalty='l2', # Tipe regularisasi L2
                            C=1.0, # Kekuatan regularisasi (semakin kecil semakin kuat)
                            solver='lbfgs', # Algoritma pelatihan
                            class_weight=None, # Menangani ketidakseimbangan kelas
                            verbose=0, # Tingkat keluaran selama pelatihan
                            multi_class='auto') # Pendekatan multi-kelas (one-vs-rest)

logreg_roc_auc_scores = []
logreg_f1_scores = []
fold_num = 1

for train_index, test_index in skf.split(X_train, y_train):
    train_features = X_train.iloc[train_index]
    test_features = X_train.iloc[test_index]

    train_labels = y_train.iloc[train_index]
    test_labels = y_train.iloc[test_index]

    logreg.fit(train_features, train_labels)
    pred_labels = logreg.predict(test_features)

    f1 = f1_score(test_labels, pred_labels)
    roc_auc = roc_auc_score(test_labels, pred_labels)

    logreg_f1_scores.append(f1)
    logreg_roc_auc_scores.append(roc_auc)

    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'test'] = test_index

    print(f"F1-score for fold {fold_num}: ", f1)
    print(f"ROC-AUC score for fold {fold_num}: ", roc_auc, '\n')

    fold_num += 1

```

Gambar 69. Model Klasifikasi Logistic Regression

Berikut ini merupakan F1-Score dan AUC-ROC dari model Logistic Regression yang telah dilatih menggunakan setiap fold.

```

F1-score for fold 1: 0.770316027088036
ROC-AUC score for fold 1: 0.772559496113743

F1-score for fold 2: 0.7815054976036085
ROC-AUC score for fold 2: 0.7834566297454025

F1-score for fold 3: 0.7781531531531533
ROC-AUC score for fold 3: 0.7798247202800479

F1-score for fold 4: 0.7806795843864083
ROC-AUC score for fold 4: 0.781784087112116

F1-score for fold 5: 0.7650365373805509
ROC-AUC score for fold 5: 0.7664168990509976

F1-score for fold 6: 0.7663288288288289
ROC-AUC score for fold 6: 0.7680938135283593

F1-score for fold 7: 0.7777463993222252
ROC-AUC score for fold 7: 0.7800447177193963

F1-score for fold 8: 0.7673169352558666
ROC-AUC score for fold 8: 0.7699832308552264

F1-score for fold 9: 0.7762039660056658
ROC-AUC score for fold 9: 0.7792062604807155

F1-score for fold 10: 0.7815366972477062
ROC-AUC score for fold 10: 0.7870318613750699

```

Gambar 70. F1-Score dan AUC-ROC dari 10 Fold Logistic Regression

Setelah didapatkan F1-Score dan AUC-ROC dari setiap fold, langkah selanjutnya yaitu mengurutkan fold berdasarkan nilai rerata antara F1-Score dan AUC-ROC yang paling baik. Tujuannya untuk mengetahui nomor fold yang bisa menghasilkan nilai terbaik.

```

# Mengurutkan fold dari yang memiliki nilai rata-rata F1 dan AUC-ROC terbesar ke terkecil
fold_number = []
fold_score_mean = []

for i in range(0, len(logreg_roc_auc_scores)):
    number = i + 1
    score_mean = (logreg_roc_auc_scores[i] + logreg_f1_scores[i]) / 2
    fold_number.append(number)
    fold_score_mean.append(score_mean)

fsm = pd.DataFrame({'fold_number': fold_number, 'fold_score_mean': fold_score_mean})
fsm_sort = fsm.sort_values(by='fold_score_mean', ascending=False).reset_index(drop=True)
fsm_sort

```

	fold_number	fold_score_mean
0	10	0.784284
1	2	0.782481
2	4	0.781232
3	3	0.778989
4	7	0.778896
5	9	0.777705
6	1	0.771438
7	8	0.768650
8	6	0.767211
9	5	0.765727

Gambar 71. Rerata F1-Score dan AUC-ROC Logistic Regression Per Fold

```
best_fold = fsm_sort.loc[0, 'fold_number']
print(f"Model klasifikasi Logistic Regression terbaik menggunakan fold ke-{best_fold}")

Model klasifikasi Logistic Regression terbaik menggunakan fold ke-10
```

Gambar 72. Fold Terbaik Logistic Regression

Setelah diurutkan, didapatkan fold yang paling baik dalam melatih Logistic Regression adalah fold nomor 10 dengan rerata F1-Score dan AUC-ROC sebesar 0,784284. Model Logistic Regression yang telah dilatih dengan fold tersebut selanjutnya akan digunakan untuk memprediksi nilai F1-Score dan AUC-ROC pada test set (X_{test} dan y_{test}) dan didapatkan hasil yang tidak jauh berbeda dengan hasil dari train set.

```
X_best_fold = X_train.iloc[kfold_set['train']].loc[best_fold]
y_best_fold = y_train.iloc[kfold_set['train']].loc[best_fold]

logreg.fit(X_best_fold, y_best_fold)
y_pred = logreg.predict(X_test)
f1_test = f1_score(y_test, y_pred)
auc_roc_test = roc_auc_score(y_test, y_pred)

print(F1-Score: {f1_test})
print(AUC-ROC Score: {auc_roc_test})
```

F1-Score: 0.7712623390207229
AUC-ROC Score: 0.7718737775459643

Gambar 73. Hasil Prediksi Logistic Regression pada Test Set

Setelah mendapatkan informasi mengenai F1-Score dan AUC-ROC semua fold train set serta dari test set, maka informasi tersebut dimasukkan ke dataframe yang merekap keseluruhan hasil dari setiap model klasifikasi Logistic Regression seperti pada gambar di bawah ini.

```

# Mendapatkan mean dan standar deviasi dari seluruh F1-Score dan ROC-AUC
roc_auc_mean = np.mean(logreg_roc_auc_scores)
roc_auc_std = np.std(logreg_roc_auc_scores)
f1_mean = np.mean(logreg_f1_scores)
f1_std = np.std(logreg_f1_scores)
best_fold = fsm_sort.loc[0, 'fold_number']

scores.loc['Logistic Regression'] = [f1_mean, roc_auc_mean, f1_std, roc_auc_std, best_fold, f1_test, auc_roc_test]
scores

```

	F1-Score Mean	ROC-AUC Mean	F1-Score STD	ROC-AUC STD	Best Fold Num	F1-Score Test	ROC-AUC Test
Decision Tree	0.947333	0.944755	0.002505	0.002771	4	0.945743	0.942952
KNN	0.91614	0.90893	0.004216	0.004916	7	0.91525	0.908006
Naive Bayes	0.760208	0.740428	0.005241	0.006047	1	0.756403	0.73582
SVM	0.77347	0.776896	0.005753	0.006392	10	0.769281	0.770831
Neural Networks	0.836851	0.832978	0.007524	0.007761	4	0.834186	0.830878
Logistic Regression	0.774482	0.77684	0.006237	0.006688	10	0.771262	0.771874
Random Forest	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Gambar 74. Rekap Score Logistic Regression

g. Random Forest

Model klasifikasi terakhir yaitu model klasifikasi Random Forest. Di bawah ini merupakan kode pembuatan instance Random Forest dengan menggunakan class RandomForestClassifier() dari library Scikit-Learn. Berikut ini merupakan parameter-parameter yang kami gunakan dalam model klasifikasi Random Forest:

- n_estimators=100, yang berarti jumlah decision tree yang dibangun dalam model.
- criterion='gini' yang diartikan sebagai kriteria yang digunakan untuk memilih fitur terbaik untuk membagi node.
- max_depth=None, yang merupakan kedalaman maksimum dari setiap decision tree. None berarti node akan terus dibagi sampai semua leaf menjadi murni atau sampai semua leaf berisi kurang dari min_samples_split
- min_samples_split=2 yang berarti jumlah minimum sampel yang diperlukan untuk membagi node.
- min_samples_leaf=1 yang diartikan sebagai jumlah minimum sampel yang diperlukan untuk menjadi leaf node.
- max_features='sqrt', yaitu jumlah fitur yang dipertimbangkan saat mencari fitur terbaik untuk membagi node. max_features yang digunakan adalah 'sqrt' yang artinya akar kuadrat dari jumlah total fitur dalam data.

Parameter-parameter tersebut dipilih karena dapat menghasilkan nilai F1-Score dan AUC-ROC yang lebih tinggi dari parameter-parameter lainnya setelah dilakukan beberapa kali percobaan. Selanjutnya, Random Forest akan melatih modelnya menggunakan data train set yang telah dibagi menjadi 10 fold oleh K-Fold Cross Validation. F1-Score dan AUC-ROC dari setiap fold dihitung untuk mengetahui kinerja dari model Random Forest yang telah dilatih oleh setiap fold.

```

fold_set = pd.DataFrame(index=range(1, 11), columns=['train', 'test'])

rf = RandomForestClassifier(n_estimators=100,
                           criterion='gini',
                           max_depth=None,
                           min_samples_split=2,
                           min_samples_leaf=1,
                           max_features='sqrt')

rf_roc_auc_scores = []
rf_f1_scores = []
fold_num = 1

for train_index, test_index in skf.split(X_train, y_train):
    train_features = X_train.iloc[train_index]
    test_features = X_train.iloc[test_index]

    train_labels = y_train.iloc[train_index]
    test_labels = y_train.iloc[test_index]

    rf.fit(train_features, train_labels)
    pred_labels = rf.predict(test_features)

    f1 = f1_score(test_labels, pred_labels)
    roc_auc = roc_auc_score(test_labels, pred_labels)

    rf_f1_scores.append(f1)
    rf_roc_auc_scores.append(roc_auc)

    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'test'] = test_index

    print(f"F1-score for fold {fold_num}: ", f1)
    print(f"ROC-AUC score for fold {fold_num}: ", roc_auc, '\n')

    fold_num += 1

```

Gambar 75. Model Klasifikasi Random Forest

Berikut ini merupakan F1-Score dan AUC-ROC dari model Random Forest yang telah dilatih menggunakan setiap fold.

```

F1-score for fold 1: 0.9633225458468178
ROC-AUC score for fold 1: 0.9620107047724925

F1-score for fold 2: 0.9627630868861305
ROC-AUC score for fold 2: 0.9614517332800385

F1-score for fold 3: 0.9580870499731327
ROC-AUC score for fold 3: 0.9564236441818562

F1-score for fold 4: 0.9614451334591534
ROC-AUC score for fold 4: 0.9600346312505661

F1-score for fold 5: 0.954983922829582
ROC-AUC score for fold 5: 0.9530476437321808

F1-score for fold 6: 0.9617044228694714
ROC-AUC score for fold 6: 0.960314116996793

F1-score for fold 7: 0.9600647598488936
ROC-AUC score for fold 7: 0.9586361095584125

F1-score for fold 8: 0.9543991416309012
ROC-AUC score for fold 8: 0.9524874231414198

F1-score for fold 9: 0.9598490972783615
ROC-AUC score for fold 9: 0.9583566238121856

F1-score for fold 10: 0.9598707244815513
ROC-AUC score for fold 10: 0.9583566238121857

```

Gambar 76. F1-Score dan AUC-ROC dari 10 Fold Random Forest

Setelah didapatkan F1-Score dan AUC-ROC dari setiap fold, langkah selanjutnya yaitu mengurutkan fold berdasarkan nilai rerata antara F1-Score dan AUC-ROC yang paling baik. Tujuannya untuk mengetahui nomor fold yang bisa menghasilkan nilai terbaik.

fold_number	fold_score_mean
0	0.962667
1	0.962107
2	0.961009
3	0.960740
4	0.959350
5	0.959114
6	0.959103
7	0.957255
8	0.954016
9	0.953443

Gambar 77. Rerata FI-Score dan AUC-ROC Random Forest Per Fold

```

best_fold = fsm_sort.loc[0, 'fold_number']
print(f"Model klasifikasi Random Forest terbaik menggunakan fold ke-{best_fold}")

Model klasifikasi Random Forest terbaik menggunakan fold ke-1

```

Gambar 78. Fold Terbaik Random Forest

Setelah diurutkan, didapatkan fold yang paling baik dalam melatih Random Forest adalah fold nomor 1 dengan rerata F1-Score dan AUC-ROC sebesar 0,962667. Model Random Forest yang telah dilatih dengan fold tersebut selanjutnya akan digunakan untuk memprediksi nilai F1-Score dan AUC-ROC pada test set (X_{test} dan y_{test}) dan didapatkan hasil yang tidak jauh berbeda dengan hasil dari train set.

```

X_best_fold = X_train.iloc[kfold_set['train']].loc[best_fold]
y_best_fold = y_train.iloc[kfold_set['train']].loc[best_fold]

rf.fit(X_best_fold, y_best_fold)
y_pred = rf.predict(X_test)
f1_test = f1_score(y_test, y_pred)
auc_roc_test = roc_auc_score(y_test, y_pred)

print("F1-Score: " + str(f1_test))
print("AUC-ROC Score: " + str(auc_roc_test))

```

F1-Score: 0.9573507275464125
AUC-ROC Score: 0.9556656669709219

Gambar 79. Hasil Prediksi Random Forest pada Test Set

Setelah mendapatkan informasi mengenai F1-Score dan AUC-ROC semua fold train set serta dari test set, maka informasi tersebut dimasukkan ke dataframe yang merekap keseluruhan hasil dari setiap model klasifikasi Random Forest seperti pada gambar di bawah ini.

```

# Mendapatkan mean dan standar deviasi dari seluruh F1-Score dan ROC-AUC
roc_auc_mean = np.mean(rf_roc_auc_scores)
roc_auc_std = np.std(rf_roc_auc_scores)
f1_mean = np.mean(rf_f1_scores)
f1_std = np.std(rf_f1_scores)
best_fold = fsm_sort.loc[0, 'fold_number']

scores.loc['Random Forest'] = [f1_mean, roc_auc_mean, f1_std, roc_auc_std, best_fold, f1_test, auc_roc_test]
scores

```

	F1-Score Mean	ROC-AUC Mean	F1-Score STD	ROC-AUC STD	Best Fold Num	F1-Score Test	ROC-AUC Test
Decision Tree	0.947333	0.944755	0.002505	0.002771	4	0.945743	0.942952
KNN	0.91614	0.90893	0.004216	0.004916	7	0.91525	0.908006
Naive Bayes	0.760208	0.740428	0.005241	0.006047	1	0.756403	0.73582
SVM	0.77347	0.776896	0.005753	0.006392	10	0.769281	0.770831
Neural Networks	0.836851	0.832978	0.007524	0.007761	4	0.834186	0.830878
Logistic Regression	0.774482	0.77684	0.006237	0.006688	10	0.771262	0.771874
Random Forest	0.959649	0.958112	0.002875	0.003089	1	0.957351	0.955666

Gambar 80. Rekap Score Random Forest

10. Model Selection

Berdasarkan dataframe rekap hasil kinerja score setiap model klasifikasi, kita dapat menentukan model klasifikasi terbaik di antara 6 model klasifikasi tersebut. Dengan mengurutkan dataframe tersebut dari nilai rata-rata F1-Score dan rata-rata AUC-ROC tertinggi ke terendah, maka didapatkan bahwa model klasifikasi terbaik adalah **Random Forest** dengan menggunakan **fold nomor ke-1**, yang berhasil mendapatkan nilai F1-Score dan AUC-ROC di sekitar nilai 0,95.

```
[ ] # Mendapatkan fold dengan nilai rata-rata skor F1 dan AUC-ROC tertinggi
scores_sort = scores.sort_values(
    by=['F1-Score Mean', 'ROC-AUC Mean', 'F1-Score Test', 'ROC-AUC Test'],
    ascending=[False, False, False, False])
```

```
scores_sort
```

	F1-Score Mean	ROC-AUC Mean	F1-Score STD	ROC-AUC STD	Best Fold Num	F1-Score Test	ROC-AUC Test
Random Forest	0.959649	0.958112	0.002875	0.003089	1	0.957351	0.955666
Decision Tree	0.947333	0.944755	0.002505	0.002771	4	0.945743	0.942952
KNN	0.91614	0.90893	0.004216	0.004916	7	0.91525	0.908006
Neural Networks	0.836851	0.832978	0.007524	0.007761	4	0.834186	0.830878
Logistic Regression	0.774482	0.77684	0.006237	0.006688	10	0.771262	0.771874
SVM	0.77347	0.776896	0.005753	0.006392	10	0.769281	0.770831
Naive Bayes	0.760208	0.740428	0.005241	0.006047	1	0.756403	0.73582

Gambar 81. Rekap Score Semua Model Klasifikasi

Lalu didapatkan juga nomor fold terbaik yang digunakan untuk melatih setiap model klasifikasi lainnya, seperti yang terlihat di kolom Best Fold Num.

11. Backward Selection

a. Decision Tree

Pertama, kami melakukan *backward selection* pada model Decision Tree yang telah dilatih menggunakan *fold* terbaiknya, yaitu *fold* ke-4. Kami mencoba mengurangi *feature/variabel* dari yang berjumlah 16 hingga 10. Berikut ini merupakan proses *backward selection* pada model Decision Tree.

```

[ ] dtree_backward_selection = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'AUC-ROC Score', 'Mean', 'Features'])

best_fold = scores_sort.loc['Decision Tree', 'Best Fold Num']
X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

for i in range(16, 9, -1):
    sbs = SFS(dtree, k_features=i, forward=False, floating=False, cv=0)
    sbs.fit(X_best_fold, y_best_fold)

    X_train_sfs = sbs.transform(X_best_fold)
    X_test_sfs = sbs.transform(X_test)
    dtree.fit(X_train_sfs, y_best_fold)
    y_pred = dtree.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred)
    mean = (f1 + auc_roc) / 2
    features = str(sbs.k_feature_names_)
    dtree_backward_selection.loc[i] = [f1, auc_roc, mean, features]

print(f"F1-score dengan {i} feature: {f1}")

F1-score dengan 16 feature: 0.9442553981315349
F1-score dengan 15 feature: 0.9427917620137299
F1-score dengan 14 feature: 0.9439732557419674
F1-score dengan 13 feature: 0.941067457375834
F1-score dengan 12 feature: 0.9400615384615384
F1-score dengan 11 feature: 0.9327046720960136
F1-score dengan 10 feature: 0.9272782688204816

```

Gambar 82. Backward Selection Decision Tree

Dari hasil tersebut, tidak ada perubahan yang signifikan pada nilai F1-Score dan AUC-ROC-nya ketika jumlah *feature* dikurangi dari 16 hingga 10. Namun terdapat kecenderungan nilai F1-Score dan AUC-ROC menjadi turun ketika banyaknya *feature* berkurang. Nilai tertinggi muncul ketika jumlah *feature*nya 16. Artinya, model Decision Tree bekerja paling optimal ketika tidak ada *feature* yang dikurangi. Berikut ini merupakan ringkasan hasilnya.

	F1-Score	AUC-ROC Score	Mean	Features
10	0.927278	0.922219	0.924749	('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'duration')
11	0.932705	0.928348	0.930526	('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'duration')
12	0.940062	0.936498	0.93828	('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration')
13	0.941067	0.937802	0.939434	('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign')
14	0.943973	0.940996	0.942485	('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays')
15	0.942792	0.939692	0.941242	('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous')
16	0.944255	0.941257	0.942756	('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')

Gambar 83. Hasil Backward Selection Decision Tree

b. K-Nearest Neighbors

Selanjutnya, kami melakukan *backward selection* pada model K-Nearest Neighbors yang telah dilatih menggunakan *fold* terbaiknya, yaitu *fold* ke-7. Kami mencoba mengurangi *feature*/variabel dari yang berjumlah 16 hingga 10. Berikut ini merupakan proses *backward selection* pada model K-Nearest Neighbors.

```

[ ] knn_backward_selection = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'AUC-ROC Score', 'Mean', 'Features'])

best_fold = scores_sort.loc['KNN', 'Best Fold Num']
X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

for i in range(16, 9, -1):
    sbs = SFS(knn, k_features=i, forward=False, floating=False, cv=0)
    sbs.fit(X_best_fold, y_best_fold)

    X_train_sfs = sbs.transform(X_best_fold)
    X_test_sfs = sbs.transform(X_test)
    knn.fit(X_train_sfs, y_best_fold)
    y_pred = knn.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred)
    mean = (f1 + auc_roc) / 2
    features = str(sbs.k_feature_names_)
    knn_backward_selection.loc[i] = [f1, auc_roc, mean, features]

print("F1-score dengan {i} feature: {f1}")

F1-score dengan 16 feature: 0.9152501651750856
F1-score dengan 15 feature: 0.9140423489892628
F1-score dengan 14 feature: 0.9119827792394164
F1-score dengan 13 feature: 0.9122660127982777
F1-score dengan 12 feature: 0.9179360587633212
F1-score dengan 11 feature: 0.9012002601549105
F1-score dengan 10 feature: 0.9042591054692134

```

Gambar 84. Backward Selection K-Nearest Neighbors

Dari hasil tersebut, tidak ada perubahan yang signifikan pada nilai F1-Score dan AUC-ROC-nya ketika jumlah *feature* dikurangi dari 16 hingga 10. Namun nilai tertinggi bisa didapatkan ketika jumlah *feature*nya 12. Artinya, model K-Nearest Neighbors bekerja paling optimal ketika mengurangi 4 *feature*, yaitu “campaign”, “pdays”, “previous”, dan “poutcome”. Berikut ini merupakan ringkasan hasilnya.

	F1-Score	AUC-ROC Score	Mean	Features
10	0.904259	0.894771	0.899515	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘contact’, ‘day’, ‘month’)
11	0.9012	0.891055	0.896128	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’)
12	0.917936	0.911136	0.914536	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’)
13	0.912266	0.904355	0.908311	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’)
14	0.911983	0.904029	0.908006	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’)
15	0.914042	0.906572	0.910307	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’)
16	0.91525	0.908006	0.911628	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)

Gambar 85. Hasil Backward Selection K-Nearest Neighbors

c. Naive Bayes

Berikutnya, kami mencoba melakukan backward selection pada model Naive Bayes yang telah dilatih menggunakan fold terbaiknya, yaitu fold ke-1. Kami mencoba mengurangi feature/variabel dari berjumlah 16 hingga 10.

```
[ ] nbc_backward_selection = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'AUC-ROC Score', 'Mean', 'Features'])

best_fold = scores_sort.loc['Naive Bayes', 'Best Fold Num']
X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

for i in range(16, 9, -1):
    sbs = SFS(nbc, k_features=i, forward=False, floating=False, cv=0)
    sbs.fit(X_best_fold, y_best_fold)

    X_train_sfs = sbs.transform(X_best_fold)
    X_test_sfs = sbs.transform(X_test)
    nbc.fit(X_train_sfs, y_best_fold)
    y_pred = nbc.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred)
    mean = (f1 + auc_roc) / 2
    features = str(sbs.k_feature_names_)
    nbc_backward_selection.loc[i] = [f1, auc_roc, mean, features]

print(f"F1-score dengan {i} feature: {f1}")
```

Gambar 86. Backward Selection untuk Naive Bayes

Didapatkan hasil tidak ada perubahan signifikan ketika mengubah jumlah feature dari 16 hingga 10 pada dataset ini menggunakan model Naive Bayes pada nilai F1-Score dan AUC-ROC-nya. Nilai tertinggi didapatkan ketika jumlah feature sama dengan 11, yaitu dengan membuang variabel ‘job’, ‘default’, ‘balance’, ‘contact’, dan ‘month’.

	F1-Score	AUC-ROC Score	Mean	Features
10	0.755067	0.75577	0.755419	(‘age’, ‘marital’, ‘education’, ‘housing’, ‘loan’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
11	0.756447	0.756161	0.756304	(‘age’, ‘marital’, ‘education’, ‘housing’, ‘loan’, ‘day’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
12	0.756209	0.75414	0.754675	(‘age’, ‘job’, ‘marital’, ‘education’, ‘housing’, ‘loan’, ‘day’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
13	0.751852	0.751076	0.751464	(‘age’, ‘job’, ‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘day’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
14	0.75125	0.750228	0.750739	(‘age’, ‘job’, ‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
15	0.749466	0.74749	0.748478	(‘age’, ‘job’, ‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
16	0.756403	0.73582	0.746111	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)

Gambar 87. Hasil Backward Selection pada Naive Bayes

d. Support Vector Machine

Selanjutnya, kami mencoba melakukan backward selection pada model Support Vector Machine yang telah dilatih menggunakan fold terbaiknya, yaitu fold ke-10. Kami mencoba mengurangi feature/variabel dari berjumlah 16 hingga 10.

```
[ ] svm_backward_selection = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'AUC-ROC Score', 'Mean', 'Features'])

best_fold = scores_sort.loc['SVM', 'Best Fold Num']
X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

for i in range(16, 9, -1):
    sbs = SFS(svm, k_features=i, forward=False, floating=False, cv=0)
    sbs.fit(X_best_fold, y_best_fold)

    X_train_sfs = sbs.transform(X_best_fold)
    X_test_sfs = sbs.transform(X_test)
    svm.fit(X_train_sfs, y_best_fold)
    y_pred = svm.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred)
    mean = (f1 + auc_roc) / 2
    features = str(sbs.k_feature_names_)
    svm_backward_selection.loc[i] = [f1, auc_roc, mean, features]

print(f"F1-score dengan {i} feature: {f1}")
```

Gambar 88. Backward Selection untuk Support Vector Machine

Didapatkan hasil tidak ada perubahan signifikan ketika mengubah jumlah feature dari 16 hingga 10 pada dataset ini menggunakan model Support Vector Machine pada nilai F1-Score dan AUC-ROC-nya. Nilai tertinggi didapatkan ketika jumlah feature sama dengan 11, yaitu dengan membuang variabel ‘age’, ‘job’, ‘default’, ‘contact’, dan ‘month’.

	F1-Score	AUC-ROC Score	Mean	Features
10	0.768116	0.770505	0.76931	(‘marital’, ‘education’, ‘housing’, ‘loan’, ‘day’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
11	0.770629	0.772917	0.771773	(‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘day’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
12	0.76915	0.771483	0.770316	(‘age’, ‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘day’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
13	0.76917	0.770765	0.769968	(‘age’, ‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
14	0.769685	0.771157	0.770421	(‘age’, ‘job’, ‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
15	0.769483	0.770896	0.770189	(‘age’, ‘job’, ‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
16	0.769281	0.770831	0.770056	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)

Gambar 89. Hasil Backward Selection pada Support Vector Machine

e. Neural Network

Selanjutnya, kami mencoba melakukan backward selection pada model Neural Network yang telah dilatih menggunakan fold terbaiknya, yaitu fold ke-7. Kami mencoba mengurangi feature/variabel dari berjumlah 16 hingga 10.

```
nn_backward_selection = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'AUC-ROC Score', 'Mean', 'Features'])

best_fold = scores_sort.loc['Neural Networks', 'Best Fold Num']
X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

for i in range(16, 9, -1):
    sbs = SFS(nn, k_features=i, forward=False, floating=False, cv=0)
    sbs.fit(X_best_fold, y_best_fold)

    X_train_sfs = sbs.transform(X_best_fold)
    X_test_sfs = sbs.transform(X_test)
    nn.fit(X_train_sfs, y_best_fold)
    y_pred = nn.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred)
    mean = (f1 + auc_roc) / 2
    features = str(sbs.k_feature_names_)
    nn_backward_selection.loc[i] = [f1, auc_roc, mean, features]
```

Gambar 90. Backward Selection untuk Neural Network

Didapatkan hasil tidak ada perubahan signifikan ketika mengubah jumlah feature dari 16 hingga 10 pada dataset ini menggunakan model Neural Network pada nilai F1-Score dan AUC-ROC-nya. Nilai tertinggi didapatkan ketika jumlah feature sama dengan 13, yaitu dengan membuang variabel ‘job’, ‘pdays’, dan ‘previous’.

	F1-Score	AUC-ROC Score	Mean	Features
10	0.828226	0.821294	0.82476	(‘marital’, ‘default’, ‘balance’, ‘housing’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘poutcome’)
11	0.83241	0.826444	0.829427	(‘age’, ‘marital’, ‘education’, ‘housing’, ‘loan’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘previous’, ‘poutcome’)
12	0.838883	0.835311	0.837097	(‘age’, ‘education’, ‘default’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘poutcome’)
13	0.838934	0.83655	0.837742	(‘age’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘poutcome’)
14	0.836264	0.832442	0.834353	(‘age’, ‘job’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘pdays’, ‘previous’, ‘poutcome’)
15	0.826708	0.824358	0.825533	(‘age’, ‘job’, ‘marital’, ‘education’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)
16	0.838014	0.836419	0.837217	(‘age’, ‘job’, ‘marital’, ‘education’, ‘default’, ‘balance’, ‘housing’, ‘loan’, ‘contact’, ‘day’, ‘month’, ‘duration’, ‘campaign’, ‘pdays’, ‘previous’, ‘poutcome’)

Gambar 91. Hasil Backward Selection pada Neural Network

f. Logistic Regression

Berikutnya, kami mencoba melakukan backward selection pada model Logistic Regression yang telah dilatih menggunakan fold terbaiknya, yaitu fold ke-10. Kami mencoba mengurangi feature/variabel dari berjumlah 16 hingga 10.

```
logreg_backward_selection = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'AUC-ROC Score', 'Mean', 'Features'])

best_fold = scores_sort.loc['Logistic Regression', 'Best Fold Num']
X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

for i in range(16, 9, -1):
    logreg = LogisticRegression(max_iter=500)
    sbs = SFS(logreg, k_features=i, forward=False, floating=False, cv=0)
    sbs.fit(X_best_fold, y_best_fold)

    X_train_sfs = sbs.transform(X_best_fold)
    X_test_sfs = sbs.transform(X_test)
    logreg.fit(X_train_sfs, y_best_fold)
    y_pred = logreg.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred)
    mean = (f1 + auc_roc) / 2
    features = str(sbs.k_feature_names_)
    logreg_backward_selection.loc[i] = [f1, auc_roc, mean, features]

print("F1-score dengan {} feature: {}".format(i, f1))
```

Gambar 92. Backward Selection untuk Logistic Regression

Didapatkan hasil tidak ada perubahan signifikan ketika mengubah jumlah feature dari 16 hingga 10 pada dataset ini menggunakan model Logistic Regression pada nilai F1-Score dan AUC-ROC-nya. Nilai tertinggi didapatkan ketika jumlah feature sama dengan 15, dimana model Logistic Regression bekerja paling optimal ketika hanya mengurangi 1 *feature*, yaitu “job”. Berikut ini merupakan ringkasan hasilnya.

logreg_backward_selection				
	F1-Score	AUC-ROC Score	Mean	Features
10	0.768547	0.769135	0.768841	('education', 'balance', 'housing', 'loan', 'day', 'month', 'duration', 'campaign', 'previous', 'poutcome')
11	0.771199	0.771483	0.771341	('education', 'balance', 'housing', 'loan', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')
12	0.771481	0.771809	0.771645	('education', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')
13	0.770979	0.771352	0.771166	('education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')
14	0.771451	0.771809	0.77163	('age', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')
15	0.772175	0.772591	0.772383	('age', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')
16	0.771262	0.771874	0.771568	('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')

Gambar 93. Hasil Backward Selection pada Logistic Regression

g. Random Forest

Terakhir, kami mencoba melakukan backward selection pada model Random Forest yang telah dilatih menggunakan fold terbaiknya, yaitu fold ke-1. Kami mencoba mengurangi feature/variabel dari berjumlah 16 hingga 10.

```

rf_backward_selection = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'AUC-ROC Score', 'Mean', 'Features'])

best_fold = scores_sort.loc['Random Forest', 'Best Fold Num']
X_best_fold = X_train.iloc[kfold_set['train'].loc[best_fold]]
y_best_fold = y_train.iloc[kfold_set['train'].loc[best_fold]]

for i in range(16, 9, -1):
    sbs = SFS(rf, k_features=i, forward=False, floating=False, cv=0)
    sbs.fit(X_best_fold, y_best_fold)

    X_train_sfs = sbs.transform(X_best_fold)
    X_test_sfs = sbs.transform(X_test)
    rf.fit(X_train_sfs, y_best_fold)
    y_pred = rf.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred)
    mean = (f1 + auc_roc) / 2
    features = str(sbs.k_feature_names_)
    rf_backward_selection.loc[i] = [f1, auc_roc, mean, features]

print(f"F1-score dengan {i} feature: {f1}")

```

Gambar 94. Backward Selection untuk Random Forest

Didapatkan hasil tidak ada perubahan signifikan ketika mengubah jumlah feature dari 16 hingga 10 pada dataset ini menggunakan model Random Forest pada nilai F1-Score dan AUC-ROC-nya. Nilai tertinggi didapatkan ketika jumlah feature sama dengan 15, dimana model Random Forest bekerja paling optimal ketika hanya mengurangi 1 *feature*, yaitu “poutcome”. Berikut ini merupakan ringkasan hasilnya.

rf_backward_selection			Features	
	F1-Score	AUC-ROC Score	Mean	
10	0.948587	0.946147	0.947367	(age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'day', 'duration')
11	0.94971	0.947386	0.948548	(age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'duration')
12	0.953781	0.951819	0.9528	(age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration')
13	0.955645	0.95384	0.954742	(age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign')
14	0.957934	0.956252	0.957093	(age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays')
15	0.958268	0.956644	0.957456	(age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous')
16	0.956146	0.954362	0.955254	(age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')

Gambar 95. Hasil Backward Selection pada Random Forest

12. Kesimpulan

Analisis model klasifikasi terbaik yang dilakukan pada *dataset* “Kampanye Pemasaran Langsung (*Direct Marketing*) Deposito Bank Berjangka” dimulai dengan melakukan analisis univariat dan bivariat untuk mengetahui karakteristik seluruh data dan variabel. Sebelum menggunakan data ke model klasifikasi, data perlu dibersihkan/diperbaiki di dalam proses *preprocessing data*. Mulai dari pembersihan hingga akhirnya melakukan *encoding* pada seluruh variabel bukan numerik. Agar pembelajaran model klasifikasi bisa optimal, data harus diseimbangkan berdasarkan jumlah variabel dependen yang menjadi tujuannya, kemudian memisahkan/memotong data menjadi *train set* dan *test set* dengan perbandingan 70% dan 30%, serta menyamakan skala seluruh variabel numerik agar nilainya sebanding dengan variabel lainnya. Dengan begitu, data sudah siap untuk digunakan oleh setiap model klasifikasi.

Setiap model klasifikasi akan dilatih menggunakan data *train set* yang telah dibagi menjadi 10 *fold* menggunakan K-Fold Cross Validation. Nilai F1-Score dan AUC-ROC dihitung untuk mengetahui kinerja dari model klasifikasi yang telah dilatih. Berdasarkan analisis ini, didapatkan fakta bahwa model klasifikasi **Random Forest** memberikan

kinerja yang paling baik dengan rerata nilai F1-Score dan rerata nilai AUC-ROC yang dihasilkan, baik saat *di-train* maupun *di-test*, selalu menyentuh angka minimal 0,95 dengan nilai simpangan baku yang kecil.

Terakhir, dengan menggunakan semua model klasifikasi dengan *fold* terbaiknya, kami melakukan *backward selection* untuk menentukan variabel independen apa saja yang dapat dipertahankan untuk menghasilkan kinerja model yang lebih optimal. Didapatkan hasil yaitu pengurangan variabel independen, dari semula 16 dan berkurang bertahap hingga menjadi 10, tidak memberikan pengaruh yang signifikan terhadap kinerja setiap model klasifikasi. Namun terdapat beberapa model klasifikasi yang kinerjanya cenderung menurun ketika jumlah variabel independen semakin berkurang.