

# **TUGAS GROUP PROJECT #2**

## **LAPORAN KLASIFIKASI**



Kelas :  
**Predictive Modeling dan Analytics (C)**

Anggota Kelompok :

1. Anugra Salaza	5026201003
2. Nadila Nur Sholekah	5026211041
3. Maharani Putri Efendi	5026211095

**Kelompok 12**

**DEPARTEMEN SISTEM INFORMASI**  
**INSTITUT TEKNOLOGI SEPULUH NOPEMBER SURABAYA**  
**2023**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>1</b>
<b>A. Deskripsi Data.....</b>	<b>2</b>
1. Penjelasan Singkat.....	2
2. Variabel.....	2
<b>B. Praproses Data.....</b>	<b>3</b>
1. Prerequisites (Importing Dependables).....	3
2. Data Preparation.....	4
2.1. Cek Missing Value.....	4
2.2. Pengecekan Tiap Kolom.....	5
2.3. Pengecekan Outlier.....	6
2.4. Penghapusan Outlier.....	8
2.5. Categorical Encoding.....	9
3. Data Balancing.....	10
4. Data Splitting.....	10
5. Feature Scaling.....	11
<b>C. Model Klasifikasi.....</b>	<b>12</b>
1. Logistic Regression.....	12
2. Decision Tree.....	14
3. Random Forest.....	17
4. KNN.....	20
5. Naive Bayes.....	23
6. SVM.....	26
7. Neural Network.....	28
<b>D. Backward Selection.....</b>	<b>29</b>
1. Logistic Regression.....	29
2. Decision Tree.....	30
3. Random Forest.....	31
4. KNN.....	31
5. Naive Bayes.....	32
6. SVM.....	33
7. Neural Network.....	33
<b>E. Kesimpulan.....</b>	<b>34</b>
<b>F. Daftar Pustaka.....</b>	<b>35</b>

## A. Deskripsi Data

### 1. Penjelasan Singkat

Dataset yang digunakan dalam TGP #2 adalah data terkait kampanye direct marketing produk deposito bank berjangka dari suatu institusi perbankan di Portugal dari Mei 2008 hingga November 2010. Dataset ini memiliki 45.211 baris dengan 16 atribut (variabel independent) dan 1 atribut target (Variabel dependen). Dimana data yang digunakan adalah data yaitu data mengenai kampanye pemasaran langsung (direct marketing) produk deposito bank berjangka dari sebuah institusi perbankan di Portugal. Kampanye dilakukan melalui panggilan telepon di mana seorang klien mungkin harus dihubungi lebih dari sekali. Kampanye melalui pemasaran langsung ini ditujukan untuk memprediksi apakah seorang klien yang dihubungi akan membeli produk deposito berjangka tersebut atau tidak.

### 2. Variabel

Berikut merupakan penjelasan terkait variabel-variabel yang ada di dataset :

#### # Bank client data:

1. **age** (numeric)
2. **job**: type of job (categorical:  
"admin.", "unknown", "unemployed", "management", "housemaid", "entrepreneur",  
"student", "bluecollar", "self-employed", "retired", "technician", "services")
3. **marital**: marital status (categorical: "married", "divorced", "single"; note:  
"divorced" means divorced or widowed)
4. **education** (categorical: "unknown", "secondary", "primary", "tertiary")
5. **default**: has credit in default? (binary: "yes", "no")
6. **balance**: average yearly balance, in euros (numeric)
7. **housing**: has housing loan? (binary: "yes", "no")
8. **loan**: has personal loan? (binary: "yes", "no")

#### # Attribute related with the last contact of the current campaign:

9. **contact**: contact communication type (categorical:  
"unknown", "telephone", "cellular")
10. **day**: last contact day of the month (numeric)
11. **month**: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov",  
"dec")
12. **duration**: last contact duration, in seconds (numeric)

#### # other attributes:

13. **campaign**: number of contacts performed during this campaign and for this client (numeric, includes last contact)
14. **pdays**: number of days that passed by after the client was last contacted from a previous campaign (numeric, -1 means client was not previously contacted)
15. **previous**: number of contacts performed before this campaign and for this client (numeric)
16. **poutcome**: outcome of the previous marketing campaign (categorical:  
"unknown", "other", "failure", "success")

#### #Output variable (desired target):

17. **subscribe**: has the client subscribed a term deposit? (binary: "yes", "no")

## B. Praproses Data

### 1. Prerequisites (Importing Dependables)

Pada tahap pertama dalam melakukan metode klasifikasi pada *python* adalah dengan melakukan *import* dari beberapa library yang akan digunakan.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import KFold, StratifiedKFold, RepeatedKFold, RepeatedStratifiedKFold, ShuffleSplit, StratifiedShuffleSplit
from sklearn.pipeline import Pipeline
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.impute import SimpleImputer
from sklearn.compose import make_column_transformer
from sklearn.metrics import accuracy_score
from sklearn.pipeline import make_pipeline

# Load libraries
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.metrics import accuracy_score # Import scikit-learn metrics module for accuracy calculation
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.feature_selection import SelectKBest
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.model_selection import cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.compose import ColumnTransformer
```

Gambar 1. Code untuk Import Library

Kode pada gambar 1 adalah kode *Python* yang digunakan untuk melakukan analisis data dan pemodelan klasifikasi dengan menggunakan beberapa pustaka yang umum digunakan dalam ilmu data, seperti Pandas, NumPy, Matplotlib, Seaborn, dan berbagai modul dari pustaka scikit-learn.

1. **`import pandas as pd`**: Digunakan untuk mengimpor pustaka Pandas dan mengaliasnya sebagai 'pd', yang digunakan untuk manipulasi dan analisis data.
2. **`import numpy as np`**: Digunakan untuk mengimpor pustaka NumPy dan mengaliasnya sebagai 'np', yang digunakan untuk operasi numerik.
3. **`import matplotlib.pyplot as plt`**: Digunakan untuk mengimpor modul 'pyplot' dari pustaka Matplotlib, yang digunakan untuk membuat visualisasi data.
4. **`import seaborn as sns`**: Digunakan untuk mengimpor pustaka Seaborn, yang digunakan untuk meningkatkan visualisasi data.
5. **`from sklearn.model\_selection import ...`**: Digunakan untuk mengimpor berbagai modul dari scikit-learn (sklearn), yang digunakan untuk pemodelan dan evaluasi klasifikasi. Modul yang diimpor mencakup fungsi pembagian data, oversampling, pencarian parameter, metrik evaluasi, pemrosesan data, dan lainnya.
6. **`from imblearn.over\_sampling import RandomOverSampler`**: Digunakan untuk mengimpor modul dari pustaka imbalanced-learn (imblearn) yang digunakan untuk menangani ketidakseimbangan kelas dalam dataset dengan teknik oversampling.
7. **`from sklearn.preprocessing import StandardScaler, OneHotEncoder`**: Ini mengimpor modul untuk pra-pemrosesan data, seperti penskalaan fitur (StandardScaler) dan encode one-hot (OneHotEncoder).

8. ``from sklearn.metrics import ConfusionMatrixDisplay``: Digunakan untuk mengimpor modul yang digunakan untuk menampilkan matriks kebingungan dengan visual yang lebih baik.
9. ``from sklearn.impute import SimpleImputer``: Digunakan untuk mengimpor modul untuk mengatasi data yang hilang atau tidak lengkap dengan teknik imputasi sederhana.
10. ``from sklearn.compose import make_column_transformer``: Ini mengimpor modul yang digunakan untuk membuat transformasi kolom, yang dapat digunakan dalam pipa pemodelan.
11. ``from sklearn.pipeline import Pipeline, make_pipeline``: Ini mengimpor modul yang digunakan untuk membuat pipa pemodelan, yang memungkinkan untuk menggabungkan langkah-langkah pemrosesan data dan pemodelan ke dalam satu alur.

Selanjutnya, kode tersebut juga mengimpor berbagai jenis model klasifikasi yang akan digunakan, termasuk Decision Tree, Random Forest, K-Nearest Neighbors, Naive Bayes, Support Vector Machine (SVM), Logistic Regression, dan Multi-Layer Perceptron (MLP). Selain itu, kode ini juga mengimpor berbagai metrik evaluasi seperti F1-score, ROC AUC, dan lainnya. Terakhir, kode ini memuat beberapa modul tambahan seperti SequentialFeatureSelector dari pustaka mlxtend, yang digunakan untuk seleksi fitur secara berurutan, dan ColumnTransformer untuk menggabungkan transformasi kolom dalam pipa pemodelan. Kode ini menciptakan dasar kerangka kerja untuk melakukan analisis data, preprocessing, pemodelan klasifikasi, dan evaluasi menggunakan berbagai teknik dan algoritma yang telah disebutkan di atas.

## 2. Data Preparation

Sebelum melakukan preparation data yang perlu dilakukan adalah memasukan dataset terlebih dahulu dengan menggunakan kode sebagai berikut :

```
[2] tgp = pd.read_csv('Data utk TGP #1.csv', delimiter = ';')
tgp
```

**Gambar 2.** Kode untuk memasukan dataset

Setelah memasukan dataset yang akan dilakukan klasifikasi maka akan dilakukan data preparation yang merupakan proses mempersiapkan dan mengorganisasi data sebelum diolah atau dianalisis oleh model pembelajaran mesin atau algoritma lainnya. Tahap ini sangat penting dalam proses analisis data dan pembangunan model, karena kualitas data yang baik dan sesuai sangat mempengaruhi hasil akhir dari model atau analisis yang dilakukan. Langkah-langkah dalam melakukan data preparation sebagai berikut :

### 2.1. Cek Missing Value

Pada tahap pembersihan data yang pertama kali dilakukan adalah melakukan cek *missing value* pada dataset yang akan dilakukan klasifikasi. Hal ini dikarenakan *missing values* dapat mempengaruhi kualitas model dan analisis

data. Dalam melakukan pengecekan *missing value* dilakukan dengan menggunakan kode seperti pada gambar dibawah.

```
# Cek apakah terdapat baris yang kosong
tgp.isnull().sum()

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

[5] #melakukan pengecekan apakah terdapat baris duplikat
tgp.duplicated().sum()

0
```

**Gambar 3.** Kode untuk melakukan cek missing value dan duplikat data

Pada hasil *run* dengan menggunakan kode diatas didapatkan bahwa pada dataset yang akan diolah tidak memiliki missing value maupun duplikat.

## 2.2. Pengecekan Tiap Kolom

Pengecekan data pada tiap kolom dalam sebuah dataset adalah langkah yang sangat penting dalam analisis data. Ini karena setiap kolom mewakili suatu variabel atau fitur dalam dataset, dan pemahaman yang mendalam tentang karakteristik setiap variabel ini sangat krusial untuk analisis data yang akurat dan interpretasi hasil yang tepat. Pada dataset yang dianalisis memiliki 16 variabel independent dan 1 variabel dependent (subscribe). Selain mengecek satu persatu juga dilakukan penampil untuk data keseluruhan pada dataset tersebut. Dalam melakukan pengecekan data pada tiap kolomnya menggunakan kode seperti pada gambar berikut :

```

[6] #melakukan pengecekan pada kolom contact
tgp['contact'].value_counts()

[7] #melakukan pengecekan pada kolom job
tgp['job'].value_counts()

[8] #melakukan pengecekan pada kolom age
tgp['age'].value_counts()

[9] #melakukan pengecekan pada kolom marital
tgp['marital'].value_counts()

[71] #melakukan pengecekan pada kolom education
tgp['education'].value_counts()

[11] #melakukan pengecekan pada kolom default
tgp['default'].value_counts()

[12] #melakukan pengecekan pada kolom housing
tgp['housing'].value_counts()

[13] #melakukan pengecekan pada kolom subscribe
tgp['subscribe'].value_counts()

[14] #melakukan pengecekan pada kolom loan
tgp['loan'].value_counts()

[15] #melakukan pengecekan pada kolom month
tgp['month'].value_counts()

[16] #melakukan pengecekan pada kolom day
tgp['day'].value_counts()

[17] #melakukan pengecekan pada kolom poutcome
tgp['poutcome'].value_counts()

[18] #melakukan pengecekan pada kolom pdays
tgp['pdays'].value_counts()

[19] #melakukan pengecekan pada kolom previous
tgp['previous'].value_counts()

[68] #melakukan pengecekan pada kolom campaign
tgp['campaign'].value_counts()

[69] #melakukan pengecekan pada kolom balance
tgp['balance'].value_counts()

```

Gambar 4. Kode untuk melakukan pengecekan tiap kolom

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	subscribe
count	45211.000000	45211	45211	45211	45211	45211.000000	45211	45211	45211	45211.000000	45211	45211.000000	45211.000000	45211.000000	45211	45211	45211
unique	NaN	12	3	4	2	NaN	2	2	3	NaN	12	NaN	NaN	NaN	NaN	4	2
top	NaN	blue-collar	married	secondary	no	NaN	yes	no	cellular	NaN	may	NaN	NaN	NaN	NaN	unknown	no
freq	NaN	9732	27214	23202	44396	NaN	25130	37967	29285	NaN	13766	NaN	NaN	NaN	NaN	36969	39922
mean	40.536210	NaN	NaN	NaN	NaN	1362.272058	NaN	NaN	NaN	15.806419	NaN	258.163080	2.783841	40.197828	0.580323	NaN	NaN
std	10.618762	NaN	NaN	NaN	NaN	3044.765829	NaN	NaN	NaN	8.322476	NaN	257.527812	3.098021	100.128746	2.303441	NaN	NaN
min	18.000000	NaN	NaN	NaN	NaN	-8019.000000	NaN	NaN	NaN	1.000000	NaN	0.000000	1.000000	-1.000000	0.000000	NaN	NaN
25%	33.000000	NaN	NaN	NaN	NaN	72.000000	NaN	NaN	NaN	8.000000	NaN	103.000000	1.000000	-1.000000	0.000000	NaN	NaN
50%	39.000000	NaN	NaN	NaN	NaN	448.000000	NaN	NaN	NaN	16.000000	NaN	180.000000	2.000000	-1.000000	0.000000	NaN	NaN
75%	48.000000	NaN	NaN	NaN	NaN	1428.000000	NaN	NaN	NaN	21.000000	NaN	319.000000	3.000000	-1.000000	0.000000	NaN	NaN
max	95.000000	NaN	NaN	NaN	NaN	102127.000000	NaN	NaN	NaN	31.000000	NaN	4918.000000	63.000000	871.000000	275.000000	NaN	NaN

Gambar 5. Tampilan tabel dari luaran *run* untuk pengecekan detail semua data perkolomnya

### 2.3. Pengecekan Outlier

Pengecekan outlier adalah proses identifikasi dan penanganan nilai-nilai yang jauh dari sebagian besar nilai dalam suatu dataset. Outlier adalah nilai ekstrim yang jauh berbeda dari sebagian besar data. Keberadaan outlier dapat mempengaruhi hasil analisis statistik dan merusak asumsi yang mendasari banyak teknik analisis. Pada pengecekan dataset ini dilakukan pada variabel yang bertipe numeric dengan menggunakan visualisasi dari boxplot dan *z\_score*.

```

ratio_data = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']

# Membuat satu gambar dengan subplot
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))

# Iterasi melalui kolom-kolom yang akan diplot
for i, data in enumerate(ratio_data):
    # Menentukan subplot yang sesuai
    row = i // 3
    col = i % 3

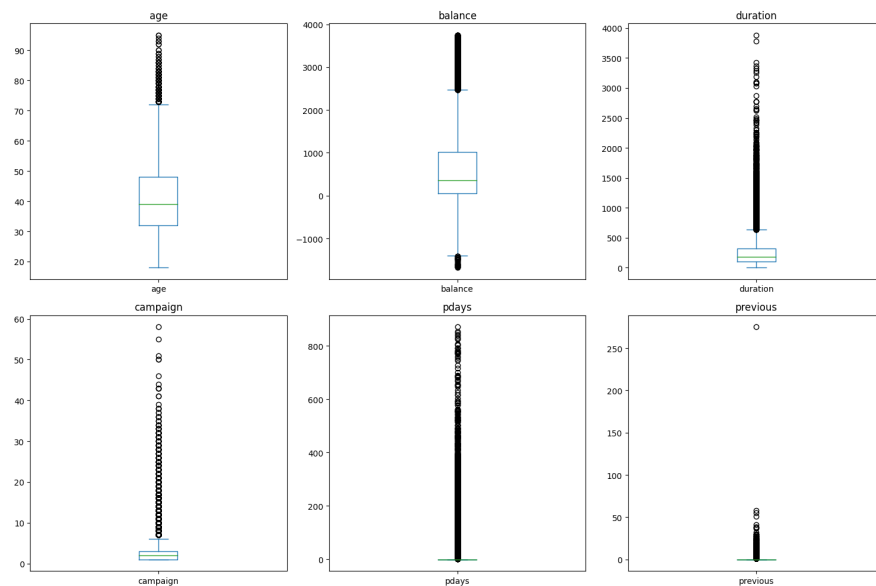
    # Membuat boxplot dalam subplot
    tgp1[data].plot(kind='box', ax=axes[row, col])
    axes[row, col].set_title(data)

# Mengatur tata letak subplot
plt.tight_layout()

# Menampilkan gambar
plt.show()

```

**Gambar 6.** Kode untuk menampilkan boxplot



**Gambar 7.** Hasil visualisasi boxplot dataset

Pada pengecekan outlier menggunakan boxplot ini memudahkan untuk memvisualisasikan distribusi data, termasuk nilai-nilai ekstrem, kuartil, median, dan sebaran data lainnya. Selain menggunakan boxplot kami juga melakukan pengecekan outlier pada tiap data numeric menggunakan z\_score dengan kode sebagai berikut :

```

[24] tgp1_mean = tgp1.balance.mean()
      tgp1_stdev = tgp1.balance.std()
      print("Balance rata-rata: {0}\nBalance Standard Deviasi: {1}".format(tgp1_mean, tgp1_stdev))

Balance rata-rata: 1362.2720576850766
Balance Standard Deviasi: 3044.765829168518

[25] # define our threshold value for z-score
      threshold = 1 # this means: we only want to tolerate values within ** 1 std. ** from the central value (mean)

[26] outlier = []
      for i in tgp1["balance"]:
          z = (i-tgp1_mean)/tgp1_stdev
          if abs(z) < threshold:
              outlier.append(i)
      print('Our outlier value in tgp is', outlier)

Our outlier value in tgp is [2143, 29, 2, 1506, 1, 231, 447, 2, 121, 593, 270, 390, 6, 71, 162, 229, 13, 52, 60, 0, 723, 7

```

**Gambar 8.** Kode dan hasil dari pengecekan outlier menggunakan z-score



## 2.4. Penghapusan Outlier

Setelah dilakukan pengecekan outlier ternyata pada data numeric memiliki outlier yang banyak maka perlu dilakukan penghapusan outlier. Penghapusan nilai outlier adalah tindakan menghapus atau menghilangkan data yang dianggap sebagai outlier dari dataset. Outlier adalah nilai yang jauh berbeda dari sebagian besar data dan bisa merusak analisis statistik atau model pembelajaran mesin.

```
[29] # Hitung mean dan standar deviasi dari 'z_score'
mean_z_score = tgp1['z_score'].mean()
std_z_score = tgp1['z_score'].std()

# Tentukan batas atas dan batas bawah untuk outlier
threshold = 3 # Threshold z-score untuk outlier
lower_bound = mean_z_score - threshold * std_z_score
upper_bound = mean_z_score + threshold * std_z_score

# List kolom-kolom yang ingin diberikan outlier treatment
ratio_data = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']

# Buang outlier
for data in ratio_data:
    tgp1 = tgp1[(tgp1['z_score'] >= lower_bound) & (tgp1['z_score'] <= upper_bound)]

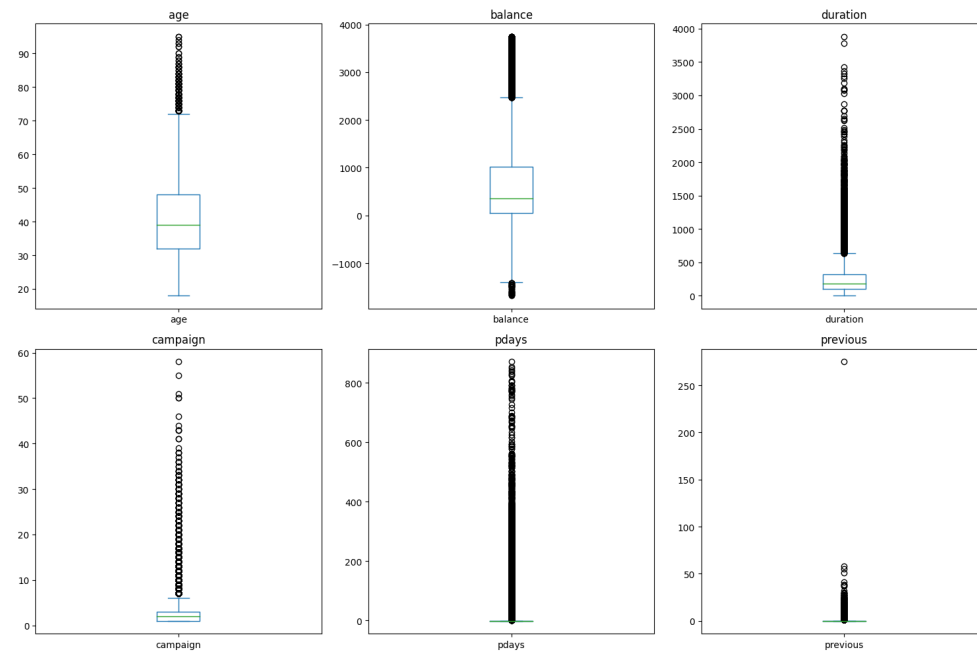
# Informasi DataFrame setelah menghapus outlier
tgp1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 40928 entries, 0 to 41735
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   age         40928 non-null  int64
 1   job         40928 non-null  object
 2   marital     40928 non-null  object
 3   education   40928 non-null  object
 4   default     40928 non-null  object
 5   balance     40928 non-null  int64
 6   housing     40928 non-null  object
 7   loan        40928 non-null  object
 8   contact     40928 non-null  object
 9   day         40928 non-null  int64
10  month       40928 non-null  object
11  duration    40928 non-null  int64
12  campaign    40928 non-null  int64
13  pdays       40928 non-null  int64
14  previous    40928 non-null  int64
15  poutcome    40928 non-null  object
16  subscribe   40928 non-null  object
17  z_score     40928 non-null  float64
dtypes: float64(1), int64(7), object(10)
memory usage: 5.9+ MB
```

**Gambar 9.** Kode dan hasil run untuk penghapusan outlier menggunakan z-score

Pada penghapusan outlier dengan z-score dilakukan dengan menghitung rata-rata (mean) dan deviasi standar (standard deviation) dari kolom `z_score` dalam dataset `tgp1`. `z_score`, lalu dilanjutkan dengan menentukan batas atas (`upper_bound`) dan batas bawah (`lower_bound`) berdasarkan nilai Z-Score. Nilai-nilai di luar batas ini dianggap sebagai outlier dan akan dihapus dari dataset dan kolom-kolom dalam dataset `tgp1` yang ingin diberikan outlier treatment. Setelah itu dilakukan pembuangan dengan melakukan iterasi melalui kolom-kolom yang dipilih dan buang baris-baris yang memiliki `z_score` di luar batas atas dan batas bawah yang telah ditentukan. Pada hasil akhirnya dapat

diketahui bahwa sisa data setelah pembuangan outlier adalah sejumlah 40928 baris data.



**Gambar 10.** Hasil visualisasi boxplot setelah penghapusan outlier

## 2.5. Categorical Encoding

Categorical encoding adalah proses mengubah variabel kategori (categorical variables) menjadi bentuk yang dapat digunakan oleh model *machine learning*. Sebagian besar algoritma *machine learning* tidak dapat bekerja langsung dengan variabel kategori, oleh karena itu, variabel kategori harus diubah menjadi bentuk numerik agar dapat digunakan dalam proses klasifikasi.

```

tgp1['job'] = tgp1['job'].replace('unknown', 0)
tgp1['education'] = tgp1['education'].replace('unknown', 0)
tgp1['contact'] = tgp1['contact'].replace('unknown', 0)
tgp1['poutcome'] = tgp1['poutcome'].replace('unknown', 0)
tgp1 = tgp1.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}}))
tgp1_encoded = tgp1.copy()

cat_columns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'subscribe', 'poutcome']
for col in cat_columns:
    tgp1_encoded[col] = tgp1_encoded[col].astype('category').cat.codes

tgp1_encoded.head()

```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	subscribe	z_score
0	58	5	1	3	0	2143	1	0	0	5	5	261	1	-1	0	0	0	0.258419
1	44	10	2	2	0	29	1	0	0	5	5	151	1	-1	0	0	0	-0.437895
2	33	3	1	2	0	2	1	1	0	5	5	76	1	-1	0	0	0	-0.446762
3	47	2	1	0	0	1506	1	0	0	5	5	92	1	-1	0	0	0	0.047205
4	33	0	2	0	0	1	0	0	0	5	5	198	1	-1	0	0	0	-0.447091

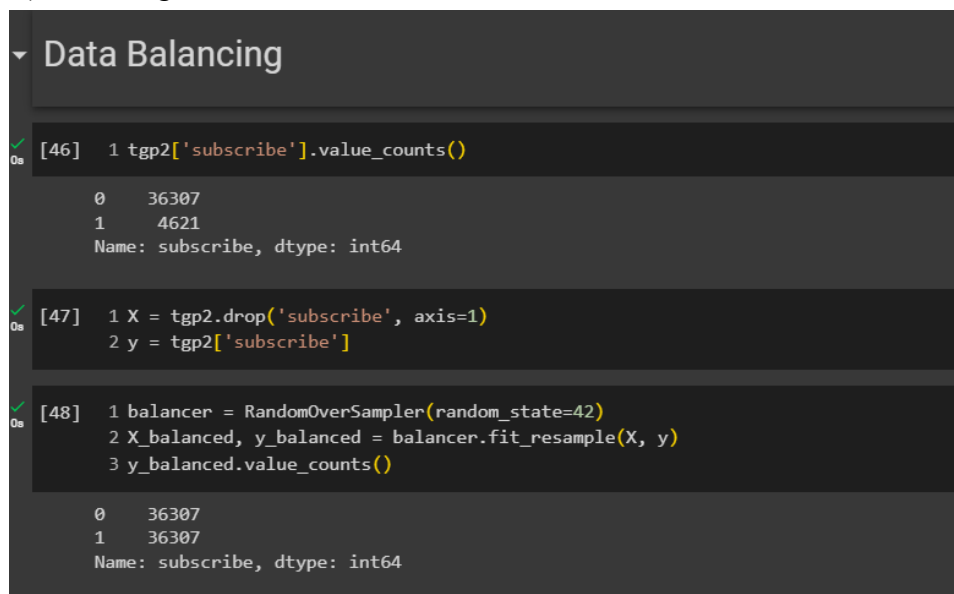
**Gambar 11.** Kode untuk melakukan label encoding

Pada *categorical encoding* disini menggunakan label encoding untuk mengubah nilai-nilai kategori menjadi nilai numerik berurutan (0, 1, 2, ..., n-1),

dimana  $n$  adalah jumlah kategori yang berbeda dalam variabel tersebut. Label *encoding* digunakan pada variabel kategori dengan tingkat ordinal (memiliki urutan tertentu).

### 3. Data Balancing

*Data balancing* merupakan teknik atau strategi untuk mengatasi masalah ketidakseimbangan dalam distribusi kelas atau label dalam dataset. Ketidakseimbangan kelas terjadi ketika salah satu kelas atau label dalam dataset memiliki jumlah sampel yang signifikan lebih sedikit atau lebih banyak daripada kelas lainnya. Hal ini dapat mempengaruhi kinerja model machine learning karena model cenderung memprediksi kelas mayoritas (kelas dengan jumlah sampel terbanyak) dan mengabaikan kelas minoritas.



```
Data Balancing

[46] 1 tgp2['subscribe'].value_counts()

0    36307
1     4621
Name: subscribe, dtype: int64

[47] 1 X = tgp2.drop('subscribe', axis=1)
     2 y = tgp2['subscribe']

[48] 1 balancer = RandomOverSampler(random_state=42)
     2 X_balanced, y_balanced = balancer.fit_resample(X, y)
     3 y_balanced.value_counts()

0    36307
1    36307
Name: subscribe, dtype: int64
```

**Gambar 12.** Melakukan *Data Balancing*

Pada pengerjaan TGP#2 dilakukan *data balancing* dengan menggunakan teknik random over sampling. Jadi, Sebelum dilakukan training, perlu dilakukan data balancing terlebih dahulu yakni menyetarakan variabel tujuan (Subscriber) pada kasus ini adalah 0 dan 1 yang awalnya bernilai 36307 untuk 0 serta 4621 untuk 1 menjadi 36307 untuk 0 dan 1 supaya ketika di training hasilnya tidak condong ke salah satu hasil.

### 4. Data Splitting

*Data splitting* adalah proses membagi dataset menjadi dua atau lebih subset yang berbeda dengan tujuan tertentu. Biasanya, dataset dibagi menjadi set pelatihan (*training set*) dan set pengujian (*test set*) untuk melatih dan menguji kinerja model *machine learning*. Proses ini penting untuk menghindari *overfitting*, yaitu ketika model terlalu "terlatih" pada data pelatihan sehingga tidak dapat menggeneralisasi dengan baik pada data baru. Pada analisis ini dilakukan pembagian keseluruhan

dataset menjadi data pelatihan (*training*) sebesar 70% dan data uji (*testing*) 30%. Kode yang dipakai untuk melakukan splitting tertera pada gambar berikut :

```
[53] X.reset_index(drop=True, inplace=True)
     y.reset_index(drop=True, inplace=True)
     X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.3, random_state=123, stratify=y_balanced)
```

**Gambar 13.** Kode untuk melakukan *Data Splitting*

Kode tersebut digunakan untuk mempersiapkan data dengan membagi dataset menjadi set pelatihan ( $x_{\text{train}}$ ,  $y_{\text{train}}$ ) dan set pengujian ( $x_{\text{test}}$ ,  $y_{\text{test}}$ ). Kode tersebut berfungsi untuk membagi dataset yang sudah seimbang ( $x_{\text{balanced}}$  dan  $y_{\text{balanced}}$ ) menjadi subset pelatihan ( $x_{\text{train}}$  dan  $y_{\text{train}}$ ) dan subset pengujian ( $x_{\text{test}}$  dan  $y_{\text{test}}$ ) menggunakan fungsi `train_test_split` dari pustaka *Scikit-Learn*. Pada kode juga digunakan “`stratify=y_balanced`” digunakan untuk memastikan bahwa distribusi kelas dalam dataset  $y_{\text{balanced}}$  dipertahankan dalam set pelatihan dan pengujian. Ini penting terutama jika dataset memiliki ketidakseimbangan kelas, sehingga memastikan bahwa setiap subset memiliki proporsi kelas yang sama dengan dataset asli.

## 5. Feature Scaling

Setelah melakukan *data splitting*, maka selanjutnya dilakukan *scaling*. Feature scaling adalah suatu teknik dalam pemrosesan data yang digunakan untuk mengubah nilai-nilai variabel ke dalam skala yang seragam tanpa mengubah bentuk distribusi variabel tersebut.

```
scaler = StandardScaler()

X_train['balance'] = scaler.fit_transform(X_train[['balance']]) # "fit" on the TRAIN set only, then transform
X_test['balance'] = scaler.transform(X_test[['balance']]) # while on the TEST set, just "transform" it

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

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

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

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

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

X_train.describe()
```

**Gambar 14.** Kode untuk melakukan *Scaling*

Kode tersebut melakukan standardization (pengubahan skala variabel agar memiliki rata-rata 0 dan deviasi standar 1) pada beberapa variabel dalam dataset menggunakan `StandardScaler` dari pustaka *Scikit-Learn*. Pada kode `scaler = StandardScaler()` merupakan kode yang menginisialisasi objek `scaler` yang merupakan instance dari `StandardScaler`. Selanjutnya Variabel 'balance', 'age', 'duration', 'campaign', 'pdays', dan 'previous' diubah skalanya pada set pelatihan dan pengujian menggunakan metode `.fit_transform()` pada set pelatihan (agar objek `scaler` "mempelajari" rata-rata dan deviasi standar dari set pelatihan) dan `.transform()` pada set pengujian (agar variabel dalam set pengujian diubah skala dengan menggunakan rata-rata dan deviasi standar yang telah dipelajari dari set pelatihan). Alasan menggunakan `.fit_transform()` pada set pelatihan dan `.transform()` pada set pengujian adalah untuk memastikan bahwa objek `scaler`

mempelajari parameter (rata-rata dan deviasi standar) dari set pelatihan saja, sehingga tidak ada kebocoran informasi dari set pengujian ke dalam proses pelatihan model.

## C. Model Klasifikasi

### 1. Logistic Regression

Logistic regression merupakan sebuah algoritma yang digunakan dalam analisis data statistik untuk mengeksplorasi hubungan antar variabel. Metode ini berfokus pada variabel dependen yang bersifat kategoris, baik nominal atau ordinal, dan variabel independen yang dapat memiliki sifat kategoris atau kontinu (Cecep Wahyu Cahyana & Akhsin Nurlayli, 2023).

```
1 numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
2 binary_features = ['default', 'housing', 'loan']
3 categorical_features = ['job', 'marital', 'education', 'contact', 'month', 'outcome']
4
5 numeric_transformer = Pipeline(steps=[('scaler', StandardScaler()), ('imputer', SimpleImputer(strategy='mean'))])
6 categorical_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')), ('onehot', OneHotEncoder())])
7 binary_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent'))])
8
9 preprocessor = ColumnTransformer(
10     transformers=[
11         ('num', numeric_transformer, numeric_features),
12         ('cat', categorical_transformer, categorical_features),
13         ('bin', binary_transformer, binary_features)
14     ],
15     remainder='drop' # Exclude other columns that are not specified
16 )
17
18 pipe = Pipeline(steps=[('preprocessor', preprocessor), ('clf', LogisticRegression(C=0.01, solver='newton-cg'))])
19 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=130)
20
21 # Melakukan validasi silang
22 scores = cross_val_score(pipe, X, y, cv=skf)
23
24 # Menampilkan hasil validasi silang
25 print("Hasil Akurasi validasi silang:")
26 print(scores)
```

Hasil Akurasi validasi silang:  
[0.9064256 0.90300513 0.90789152 0.89567554 0.9076472 0.89958466  
0.89738578 0.90202785 0.90273705 0.90542522]

Gambar 15. Kode untuk membangun pipeline serta model *Logistic Regression*

Kode tersebut adalah implementasi dari pemrosesan data dan pemodelan menggunakan algoritma Logistic Regression dalam konteks validasi silang (cross-validation). Pertama-tama, kode mendefinisikan tiga jenis fitur, yaitu fitur numerik, fitur biner, dan fitur kategorikal. Setiap jenis fitur kemudian diolah secara berbeda menggunakan pipa (Pipeline) yang sesuai. Fitur numerik dinormalisasi menggunakan *StandardScaler* dan diisi dengan nilai rata-rata, fitur kategorikal diisi dengan nilai yang paling sering muncul dan diubah menjadi representasi "one-hot encoding," sementara fitur biner diisi dengan nilai yang paling sering muncul.

Kemudian, digunakan *ColumnTransformer* untuk menggabungkan semua perubahan yang telah dilakukan pada fitur-fitur tersebut, dan 'remainder' diatur ke 'drop' untuk menghilangkan kolom lain yang tidak ada dalam daftar fitur numerik, kategorikal, atau biner. Selanjutnya, sebuah pipa lengkap dibangun dengan preprocessor sebagai tahap pertama, yang mengikuti langkah-langkah transformasi fitur, dan tahap kedua adalah algoritma Logistic Regression dengan parameter

tertentu. Proses validasi silang dilakukan menggunakan StratifiedKFold dengan 10 lipatan dan hasil akurasi validasi silang untuk model Logistic Regression ditampilkan sebagai keluaran. Kode ini berguna untuk mengukur seberapa baik model ini berkinerja dalam memprediksi target pada data yang diberikan.

```

1 kfold_set = pd.DataFrame(index=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], columns=['train', 'val'])
2 fold_num = 1
3 f1_scores = []
4 roc_auc_scores = []
5
6 for train_index, val_index in skf.split(X_train, y_train):
7     train_features, val_features = X_train.iloc[train_index], X_train.iloc[val_index]
8     train_labels, val_labels = y_train.iloc[train_index], y_train.iloc[val_index]
9     pipe.fit(train_features, train_labels)
10    pred_labels = pipe.predict(val_features)
11    f1 = f1_score(val_labels, pred_labels)
12    roc_auc = roc_auc_score(val_labels, pred_labels)
13    f1_scores.append(f1)
14    roc_auc_scores.append(roc_auc)
15    kfold_set.at[fold_num, 'train'] = train_index
16    kfold_set.at[fold_num, 'val'] = val_index
17    print(f"F1-score for fold {fold_num}: ", f1)
18    print(f"ROC-AUC score for fold {fold_num}: ", roc_auc, '\n')
19    fold_num += 1

```

```

F1-score for fold 1: 0.8347516326934494
ROC-AUC score for fold 1: 0.8357258041293518

F1-score for fold 2: 0.8293549023476031
ROC-AUC score for fold 2: 0.8298243968081604

F1-score for fold 3: 0.8356601515755884
ROC-AUC score for fold 3: 0.8378883710762689

F1-score for fold 4: 0.8380380380380381
ROC-AUC score for fold 4: 0.8408386489889959

F1-score for fold 5: 0.8383114297092792
ROC-AUC score for fold 5: 0.8402542132783175

F1-score for fold 6: 0.8319144723817067
ROC-AUC score for fold 6: 0.832973924723442

F1-score for fold 7: 0.8325377883850437
ROC-AUC score for fold 7: 0.8343519544613887

F1-score for fold 8: 0.8375552874949739
ROC-AUC score for fold 8: 0.8410430079659749

F1-score for fold 9: 0.836421597927874
ROC-AUC score for fold 9: 0.838483721414127

F1-score for fold 10: 0.8370517928286851
ROC-AUC score for fold 10: 0.8390397481306573

```

**Gambar 16.** Kode untuk menentukan F1-score dan ROC-AUC

Kode pertama membuat DataFrame "kfold\_set" yang akan digunakan untuk menyimpan indeks data latih dan data validasi untuk setiap lipatan dalam validasi silang. Selanjutnya, kode tersebut menginisialisasi variabel lipatan "fold\_num" dan dua daftar kosong "f1\_scores" dan "roc\_auc\_scores" yang akan digunakan untuk menyimpan skor F1 dan ROC-AUC untuk setiap lipatan. Selama iterasi melalui lipatan validasi silang menggunakan indeks yang dihasilkan oleh "StratifiedKFold", model yang telah diinisialisasi sebelumnya dipasang pada data latih, dan kemudian digunakan untuk membuat prediksi pada data validasi. Skor F1 dan ROC-AUC dihitung untuk setiap lipatan, dan indeks data latih dan validasi untuk setiap lipatan disimpan dalam "kfold\_set". Hasil dari setiap lipatan, termasuk skor F1 dan ROC-AUC, dicetak dalam bentuk keluaran yang menampilkan skor-skor performa model Logistic Regression pada masing-masing cross-validation ke-10. Hasil terbaik ada pada fold ke 5 dengan F1 Score sebesar 0.838 serta ROC-AUC Score 0.84.

Pemilihan nilai pada fold ke 5 ini karena nilai F1-score atau ROC-AUC score pada lipatan tersebut lebih tinggi dari fold lainnya. Nilai dari fold 5 ini akan digunakan untuk melakukan evaluasi model dengan kode berikut.

```
[ ] kfold_set.loc[1]

train    [1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 1...
val      [0, 6, 12, 21, 22, 26, 44, 51, 52, 56, 62, 63,...
Name: 1, dtype: object

[ ] X_train.iloc[kfold_set['train'].loc[1]].describe()
```

	age	job	marital	education	default	balance	housing	loan
count	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000
mean	0.001372	5.411533	1.206991	2.102916	0.016067	-0.001938	0.482097	0.138810
std	1.001370	3.240417	0.629736	0.782410	0.125735	0.998718	0.499685	0.345752
min	-1.943775	0.000000	0.000000	0.000000	0.000000	-2.601163	0.000000	0.000000
25%	-0.751179	2.000000	1.000000	2.000000	0.000000	-0.725588	0.000000	0.000000
50%	-0.240067	5.000000	1.000000	2.000000	0.000000	-0.358512	0.000000	0.000000
75%	0.611786	8.000000	2.000000	3.000000	0.000000	0.436994	1.000000	0.000000
max	4.615499	11.000000	2.000000	3.000000	1.000000	3.135581	1.000000	1.000000

```
[ ] X_fold_1 = X_train.iloc[kfold_set['train'].loc[1]]
y_fold_1 = y_train.iloc[kfold_set['train'].loc[1]]
decision_tree = DecisionTreeClassifier(min_samples_leaf= 2, max_depth= 5, min_samples_split =2)
decision_tree.fit(X_fold_1, y_fold_1)
y_pred = decision_tree.predict(X_test)
print(f"F1-score: {f1_score(y_test, y_pred)}")

F1-score: 0.8250033800531795
```

**Gambar 17.** Kode untuk evaluasi dengan penerapan hyperparameter tuning dan fold yang dipilih model *Logistic Regression*

Kode tersebut merupakan langkah untuk evaluasi model Logistic Regression menggunakan metrik F1-score. Hasil run kode tersebut menunjukkan F1-score sebesar 0.8072850487081745, yang merupakan nilai F1-score dari model Logistic Regression pada data uji. F1-score adalah suatu metrik yang menggabungkan presisi (precision) dan recall, memberikan gambaran keseluruhan tentang kinerja model untuk klasifikasi dua kelas. Semakin tinggi nilai F1-score, semakin baik kinerja model dalam memprediksi kelas-kelas yang diminati. Dalam konteks ini, nilai F1-score sebesar 0.8072850487081745 menunjukkan bahwa model Logistic Regression memiliki kinerja yang baik dalam memprediksi kelas-kelas target pada data uji yang digunakan.

## 2. Decision Tree

*Decision Tree* merupakan salah satu metode supervised machine learning non parametrik yang digunakan untuk klasifikasi atau regresi. Metode *decision tree* ini akan menghasilkan suatu model yang dapat memprediksi kategori data dengan cara mempelajari aturan penentuan kategori berdasarkan fitur-fitur yang dimiliki oleh data( Ochiai, Masuma, & Tomii, 2019). *Decision tree* membangun model dengan bentuk struktur pohon. Konsep ini dilakukan dengan memecah dataset menjadi subset yang lebih kecil lagi dan saling berkaitan.

Tujuan penggunaan algoritma *decision tree* adalah untuk memprediksi kelas atau nilai variabel target dengan mempelajari aturan keputusan sederhana yang disimpulkan dari data sebelumnya. Dalam konteks klasifikasi, *decision tree*



digunakan untuk memprediksi kelas atau label dari suatu sampel berdasarkan nilai-nilai atributnya. Pada setiap simpul dalam pohon keputusan, algoritma memilih atribut yang paling baik memisahkan data menjadi kelas-kelas yang berbeda. Proses ini berlanjut hingga mencapai daun pohon, di mana prediksi kelas dilakukan berdasarkan mayoritas kelas dari sampel-sampel yang mencapai daun tersebut.

```

numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
binary_features = ['default', 'housing', 'loan']
categorical_features = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler()), ('imputer', SimpleImputer(strategy='mean'))])
categorical_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')), ('onehot', OneHotEncoder())])
binary_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features),
        ('bin', binary_transformer, binary_features)
    ],
    remainder='drop' # Exclude other columns that are not specified
)

pipe = Pipeline(steps=[('preprocessor', preprocessor), ('clf', DecisionTreeClassifier(min_samples_leaf= 2, max_depth= 5, min_samples_split =2))])
# Initialize Stratified KFold splitter
strat_kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=130)

# Melakukan validasi silang
scores = cross_val_score(pipe, X, y, cv=strat_kfold)

# Menampilkan hasil validasi silang
print("Hasil Akurasi validasi silang:")
print(scores)

```

Hasil Akurasi validasi silang:  
[0.90105057 0.9076472 0.9064256 0.89934034 0.90495969 0.90398241  
0.90202785 0.90935744 0.90493646 0.90762463]

**Gambar 18.** Kode untuk membangun pipeline serta model *Decision tree*

Kode tersebut merupakan implementasi dari sebuah alur kerja yang kompleks dalam *machine learning* untuk melakukan validasi silang (*cross-validation*) menggunakan pipeline dan transformer. Pada tahap ini fitur-fitur dari dataset dipisahkan menjadi tiga kelompok: numerik, biner (yang hanya memiliki dua nilai), dan kategorikal. Selanjutnya, untuk mengatasi *missing values* dan melakukan normalisasi pada fitur numerik, digunakan dua transformasi: `'StandardScaler()'` untuk normalisasi dan `'SimpleImputer(strategy='mean')'` untuk mengisi nilai-nilai yang hilang. Sedangkan untuk fitur kategorikal, nilai-nilai yang hilang diisi dengan nilai yang paling sering muncul menggunakan `'SimpleImputer(strategy='most_frequent')'` dan dilakukan *one-hot encoding* menggunakan `'OneHotEncoder()'`. Kemudian, dilakukan pengelompokan transformasi menggunakan `'ColumnTransformer()'`. Fitur-fitur numerik diproses menggunakan `numeric_transformer`, fitur kategorikal menggunakan `categorical_transformer`, dan fitur biner menggunakan `binary_transformer`. Kolom-kolom lain yang tidak termasuk dalam ketiga jenis fitur ini akan diabaikan atau dihapus menggunakan `remainder='drop'`. Selanjutnya, semua langkah-langkah pemrosesan data tersebut diintegrasikan ke dalam sebuah pipeline menggunakan `'Pipeline()'`. Pipeline adalah cara untuk menyusun beberapa langkah pengolahan data dan model menjadi satu kesatuan. Dalam hal ini, terdapat dua langkah: preprocessor (yang melibatkan `ColumnTransformer`) dan model (`DecisionTreeClassifier` dengan parameter tertentu).

Setelah itu, dilakukan Stratified K-Fold Cross Validation dengan 10 lipatan (folds) menggunakan `'StratifiedKFold()'`. Stratified K-Fold memastikan bahwa setiap lipatan dari dataset memiliki proporsi kelas yang sama seperti dataset aslinya. Hasil dari validasi silang ini adalah Hasil Akurasi validasi silang: [0.90105057



0.9076472 0.9064256 0.89934034 0.90495969 0.90398241 0.90202785  
0.90935744 0.90493646 0.90762463].

```
kfold_set = pd.DataFrame(index=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], columns=['train', 'val'])
fold_num = 1
f1_scores = []
roc_auc_scores = []

for train_index, val_index in strat_kfold.split(X_train, y_train):
    train_features, val_features = X_train.iloc[train_index], X_train.iloc[val_index]
    train_labels, val_labels = y_train.iloc[train_index], y_train.iloc[val_index]
    pipe.fit(train_features, train_labels)
    pred_labels = pipe.predict(val_features)
    f1 = f1_score(val_labels, pred_labels)
    roc_auc = roc_auc_score(val_labels, pred_labels)
    f1_scores.append(f1)
    roc_auc_scores.append(roc_auc)
    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'val'] = val_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

F1-score for fold 1: 0.8230243902439024
ROC-AUC score for fold 1: 0.8215637270247897

F1-score for fold 2: 0.8169730152522486
ROC-AUC score for fold 2: 0.8158580089057165

F1-score for fold 3: 0.8180023687327279
ROC-AUC score for fold 3: 0.8186104381725502

F1-score for fold 4: 0.8164318317126413
ROC-AUC score for fold 4: 0.8180191824959724

F1-score for fold 5: 0.8221050576059363
ROC-AUC score for fold 5: 0.820773693797798

F1-score for fold 6: 0.8141139365267102
ROC-AUC score for fold 6: 0.8144800565764732

F1-score for fold 7: 0.817011314865392
ROC-AUC score for fold 7: 0.8154616763443028

F1-score for fold 8: 0.8185822987585103
ROC-AUC score for fold 8: 0.8217622883489337

F1-score for fold 9: 0.8212050078247262
ROC-AUC score for fold 9: 0.8201838394778813

F1-score for fold 10: 0.8176643768400393
ROC-AUC score for fold 10: 0.8171979535615899
```

**Gambar 19.** Kode untuk menentukan F1-score dan ROC-AUC

Kode tersebut adalah implementasi Stratified K-Fold Cross Validation secara manual menggunakan sebuah loop. Pertama-tama, sebuah DataFrame kosong `kfold_set` dibuat dengan dua kolom: 'train' untuk indeks data latih dan 'val' untuk indeks data validasi. Variabel `fold_num` digunakan untuk melacak nomor lipatan saat ini. Loop tersebut kemudian iterasi melalui setiap lipatan yang dihasilkan oleh `StratifiedKFold()`. Pada setiap iterasi, data latih dan data validasi dipisahkan menggunakan indeks yang diberikan oleh `train_index` dan `val_index`. Model (`pipe`, yang telah dijelaskan sebelumnya) dilatih pada data latih dan kemudian digunakan untuk membuat prediksi pada data validasi. Hasil prediksi ini kemudian digunakan untuk menghitung nilai F1-score dan ROC-AUC score, yang kemudian disimpan dalam list `f1_scores` dan `roc_auc_scores` masing-masing. Selain itu, indeks data latih dan validasi untuk setiap lipatan disimpan dalam DataFrame `kfold_set`.

Setiap hasil F1-score dan ROC-AUC score untuk setiap lipatan dicetak ke layar menggunakan pernyataan `print()`. Proses ini dilakukan sebanyak 10 kali, sesuai dengan jumlah lipatan yang diatur pada `StratifiedKFold()`. Hasil dari setiap lipatan, termasuk skor F1 dan ROC-AUC, dicetak dalam bentuk keluaran yang menampilkan skor-skor performa model *Decision tree* pada masing-masing cross-validation ke-10. Hasil terbaik ada pada fold ke 1 dengan F1-score sebesar 0.823 serta ROC-AUC Score 0.821. Fold ke 1 dipilih karena memiliki karena nilai F1-score atau ROC-AUC score pada lipatan tersebut lebih tinggi dari fold lainnya. Nilai dari fold 1 ini akan digunakan untuk melakukan evaluasi model dengan kode berikut.

```
[ ] kfold_set.loc[1]

train [1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 1...
val [0, 6, 12, 21, 22, 26, 44, 51, 52, 56, 62, 63,...
Name: 1, dtype: object

[ ] X_train.iloc[kfold_set['train'].loc[1]].describe()
```

	age	job	marital	education	default	balance	housing	loan	contact
count	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000	45746.000000
mean	0.001372	5.411533	1.206991	2.102916	0.016067	-0.001938	0.482097	0.138810	0.854326
std	1.001370	3.240417	0.629736	0.782410	0.125735	0.998718	0.499685	0.345752	0.501288
min	-1.943775	0.000000	0.000000	0.000000	0.000000	-2.601163	0.000000	0.000000	0.000000
25%	-0.751179	2.000000	1.000000	2.000000	0.000000	-0.725588	0.000000	0.000000	1.000000
50%	-0.240067	5.000000	1.000000	2.000000	0.000000	-0.358512	0.000000	0.000000	1.000000
75%	0.611786	8.000000	2.000000	3.000000	0.000000	0.436994	1.000000	0.000000	1.000000
max	4.615499	11.000000	2.000000	3.000000	1.000000	3.135581	1.000000	1.000000	2.000000

```
[ ] X_fold_1 = X_train.iloc[kfold_set['train'].loc[1]]
y_fold_1 = y_train.iloc[kfold_set['train'].loc[1]]
decision_tree = DecisionTreeClassifier(min_samples_leaf= 2, max_depth= 5, min_samples_split =2)
decision_tree.fit(X_fold_1, y_fold_1)
y_pred = decision_tree.predict(X_test)
print(f"F1-score: {f1_score(y_test, y_pred)}")

F1-score: 0.8250833800531795
```

**Gambar 20.** Kode untuk mengevaluasi evaluasi dengan penerapan hyperparameter tuning dan fold yang dipilih model *Decision Tree*

Kode tersebut merupakan langkah dalam evaluasi model *Decision Tree* menggunakan metrik F1-score. Dengan hasil F1-score sekitar 0.825, sehingga model *Decision Tree* ini menunjukkan kinerja yang baik dalam melakukan klasifikasi pada data uji. Nilai F1-score yang tinggi menunjukkan bahwa model memiliki tingkat presisi dan recall yang baik dalam memprediksi kelas positif dibandingkan dengan metrik F1-score.

### 3. Random Forest

Salah satu metode yang digunakan dalam data mining adalah *Random Forest*. Metode ini merupakan sebuah ansambel yang menggunakan *decision tree* sebagai *base classifier* yang dibangun dan digabungkan (Kulkarni & Sinha, 2014) Pada dasarnya. *Random Forest* berfokus pada *decision tree*. Dengan kata lain, *Random Forest* terdiri dari sejumlah *decision tree* yang digunakan untuk membentuk sebuah hutan acak yang berguna dalam proses klasifikasi dan prediksi data (Haristu, 2019). Dengan demikian, algoritma ini merupakan kombinasi masing-masing *tree* dari *Decision Tree* yang kemudian digabungkan menjadi satu model. Biasanya, *Random Forest* dipakai untuk masalah regresi dan klasifikasi dengan kumpulan data yang berukuran besar. *Random Forest* bekerja dengan membangun beberapa *Decision Tree* dan menggabungkannya demi mendapatkan prediksi yang lebih stabil dan akurat. ‘Hutan’ yang dibangun oleh *Random Forest* adalah kumpulan *decision tree* dimana biasanya dilatih dengan metode *bagging*. Ide umum dari metode *bagging* adalah kombinasi model pembelajaran untuk meningkatkan hasil keseluruhan.

```

numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
binary_features = ['default', 'housing', 'loan']
categorical_features = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler()), ('imputer', SimpleImputer(strategy='mean'))])
categorical_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')), ('onehot', OneHotEncoder())])
binary_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features),
        ('bin', binary_transformer, binary_features)
    ],
    remainder='drop' # Exclude other columns that are not specified
)

pipe = Pipeline(steps=[('preprocessor', preprocessor), ('clf', RandomForestClassifier(n_estimators=100, min_samples_split=20, min_samples_leaf=8, max_depth=10))])
# initialize Stratified Kfold splitter
strat_kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=130)

# Melakukan validasi silang
scores = cross_val_score(pipe, X, y, cv=strat_kfold)

# Menampilkan hasil validasi silang
print("Hasil Akurasi validasi silang:")
print(scores)

```

```

Hasil Akurasi validasi silang:
[0.89934034 0.8988517  0.90227217 0.90080625 0.90105057 0.90153921
 0.90170333 0.89950466 0.89657195 0.90151515]

```

**Gambar 21.** Kode untuk membangun pipeline serta model *Random Forest*

Kode tersebut digunakan untuk mengolah dan melatih model klasifikasi Random Forest Classifier dengan menggunakan preprocessor yang mencakup penskalaan fitur numerik, pengisian nilai yang hilang (missing values) dengan rata-rata untuk fitur numerik, dan pengisian nilai yang hilang dengan nilai yang paling sering muncul (mode) untuk fitur kategorikal dan biner. Proses ini dilakukan menggunakan teknik pipelining dalam scikit-learn. Data dibagi menjadi tiga jenis fitur: numerik, kategorikal, dan biner. Masing-masing jenis fitur diproses menggunakan transformers khusus: fitur numerik di-scala dan nilai yang hilang diisi dengan rata-rata, fitur kategorikal diisi dengan nilai yang paling sering muncul dan diubah menjadi vektor biner dengan one-hot encoding, dan fitur biner juga diisi dengan nilai yang paling sering muncul. Setelah itu, model Random Forest Classifier dengan parameter tertentu diterapkan pada data yang telah di-preprocess. Evaluasi dilakukan menggunakan validasi silang dengan Stratified K-Fold Cross-Validation dengan 10 lipatan, dan hasil akurasi untuk setiap lipatan dicetak ke layar. Hasil akurasi validasi silang menunjukkan kinerja model yang relatif stabil dan akurat di setiap lipatan.

```
[ ] kfold_set = pd.DataFrame(index=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], columns=['train', 'val'])
fold_num = 1
f1_scores = []
roc_auc_scores = []

for train_index, val_index in strat_kfold.split(X_train, y_train):
    train_features, val_features = X_train.iloc[train_index], X_train.iloc[val_index]
    train_labels, val_labels = y_train.iloc[train_index], y_train.iloc[val_index]
    pipe.fit(train_features, train_labels)
    pred_labels = pipe.predict(val_features)
    f1 = f1_score(val_labels, pred_labels)
    roc_auc = roc_auc_score(val_labels, pred_labels)
    f1_scores.append(f1)
    roc_auc_scores.append(roc_auc)
    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'val'] = val_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
```

```
F1-score for fold 1: 0.8787478785593061
ROC-AUC score for fold 1: 0.8735084503985155

F1-score for fold 2: 0.873446327683616
ROC-AUC score for fold 2: 0.8678034289578529

F1-score for fold 3: 0.8770600492517522
ROC-AUC score for fold 3: 0.8723271161759111

F1-score for fold 4: 0.8856327307326356
ROC-AUC score for fold 4: 0.8817694298167799

F1-score for fold 5: 0.880301602262017
ROC-AUC score for fold 5: 0.8750652168326154

F1-score for fold 6: 0.869891263592051
ROC-AUC score for fold 6: 0.863456775444473

F1-score for fold 7: 0.8780579601053818
ROC-AUC score for fold 7: 0.8725073236374288

F1-score for fold 8: 0.8860133206470028
ROC-AUC score for fold 8: 0.8821495839591826

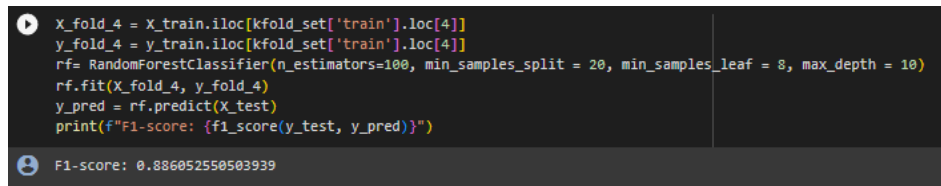
F1-score for fold 9: 0.8708097928436912
ROC-AUC score for fold 9: 0.8650315781064655

F1-score for fold 10: 0.8800904977375567
ROC-AUC score for fold 10: 0.8748524203069659
```

**Gambar 22.** Kode untuk mencari F1-score dari model *Random Forest*

Kode ini digunakan untuk melakukan validasi silang (cross-validation) dengan membagi data menjadi 10 lipatan (folds) menggunakan Stratified K-Fold. Pertama, DataFrame `kfold_set` dibuat untuk menyimpan indeks data yang digunakan sebagai data latih (train) dan data validasi (val) untuk masing-masing lipatan. Kemudian, kode melakukan loop melalui 10 lipatan yang telah dibuat. Dalam setiap iterasi lipatan, data latih dan data validasi dipisahkan berdasarkan indeks yang dihasilkan oleh Stratified K-Fold. Model yang telah diinisialisasi sebelumnya (dalam variabel `pipe`) diberi data latih dari iterasi saat ini. Model tersebut kemudian digunakan untuk membuat prediksi pada data validasi, dan metrik evaluasi, seperti F1-score dan ROC-AUC score, dihitung. Hasil metrik evaluasi ini dicatat dalam list F1-scores dan ROC-AUC scores untuk setiap lipatan. Selain itu, indeks data latih dan validasi untuk masing-masing lipatan juga disimpan dalam DataFrame `kfold_set`.

Selanjutnya, hasil F1-score dan ROC-AUC score untuk masing-masing lipatan dicetak ke layar sebagai bagian dari proses evaluasi. Dengan menggunakan validasi silang, kita dapat memperoleh gambaran yang lebih kuat tentang kinerja model pada berbagai data latih dan data validasi. Ini membantu dalam mengukur sejauh mana model dapat digeneralisasikan ke data yang belum pernah dilihat sebelumnya. Hasil evaluasi ini membantu kita memahami sejauh mana model dapat memprediksi dengan baik dan memberikan wawasan tentang stabilitas dan konsistensi kinerja model. Pada hasil run kode ini didapatkan bahwa fold ke 4 memiliki hasil yang paling bagus yaitu F1-score dan ROC-AUC sebesar 0.88.



```

X_fold_4 = X_train.iloc[kfold_set['train'].loc[4]]
y_fold_4 = y_train.iloc[kfold_set['train'].loc[4]]
rf= RandomForestClassifier(n_estimators=100, min_samples_split = 20, min_samples_leaf = 8, max_depth = 10)
rf.fit(X_fold_4, y_fold_4)
y_pred = rf.predict(X_test)
print(f"F1-score: {f1_score(y_test, y_pred)}")

F1-score: 0.886052550503939

```

**Gambar 23.** Kode untuk mengevaluasi evaluasi dengan penerapan hyperparameter tuning dan fold yang dipilih model *Random Forest*

Kode tersebut mengambil data latih dari fold keempat yang telah ditentukan sebelumnya melalui validasi silang. Data latih (X\_fold\_4 dan y\_fold\_4) kemudian digunakan untuk melatih model Random Forest Classifier. Model tersebut diberi parameter-parameter tertentu, yaitu 100 pohon (*estimators*), minimum sampel untuk *split* (*min\_samples\_split*) sebanyak 20, minimum sampel untuk daun (*min\_samples\_leaf*) sebanyak 8, dan kedalaman maksimum pohon (*max\_depth*) sebanyak 10. Setelah model dilatih, digunakan untuk membuat prediksi pada data uji (X\_test). Prediksi yang dihasilkan (y\_pred) kemudian dibandingkan dengan nilai sebenarnya dari data uji (y\_test). Nilai F1-score dihitung sebagai metrik evaluasi, yang menggabungkan presisi dan recall. Hasil F1-score sebesar 0.886052550503939 menunjukkan bahwa model *Random Forest Classifier* dengan konfigurasi tertentu memiliki kinerja yang baik dalam memprediksi kelas target pada data uji yang digunakan. F1-score yang tinggi menandakan kemampuan model dalam memprediksi dengan baik tanpa mengorbankan presisi atau recall secara signifikan.

#### 4. KNN

KNN (K-Nearest Neighbors) menjadi salah satu algoritma yang digunakan dalam proses klasifikasi dengan cara mempelajari kesamaan data berdasarkan fungsi jarak data satu dengan lainnya. Prinsip KNN adalah bahwa sampel yang paling mirip dengan kelas yang sama dari data training memiliki probabilitas paling tinggi. KNN sangat bergantung pada penentuan nilai k untuk memilih jumlah tetangga terdekatnya berdasar urutan dari nilai yang paling kecil dari hasil perhitungan jarak atau distance. Proses klasifikasi ditentukan dengan memilih suara terbanyak dari tetangga (Ulya, Soeleman & Budiman, 2021). Dengan prinsip yang dimiliki tersebut, KNN algoritma dinilai sederhana dan mudah diimplementasikan sehingga tidak perlu membangun model berulang kali, membuat parameter, atau asumsi tambahan lainnya. Namun, algoritma ini tentunya juga memiliki kelemahan yaitu menjadi lebih lambat jika jumlah data yang dikelompokkan semakin banyak. Biaya komputasinya cukup tinggi karena harus menghitung jarak antara titik data dengan semua sampel yang tersebar di sekitarnya.

```

numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
binary_features = ['default', 'housing', 'loan']
categorical_features = ['job', 'marital', 'education', 'contact', 'month', 'outcome']

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler()), ('imputer', SimpleImputer(strategy='mean'))])
categorical_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')), ('onehot', OneHotEncoder())])
binary_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features),
        ('bin', binary_transformer, binary_features)
    ],
    remainder='drop' # Exclude other columns that are not specified
)

pipe = Pipeline(steps=[('preprocessor', preprocessor), ('clf', KNeighborsClassifier(metric='manhattan', n_neighbors=9, weights='distance'))])
# initialize Stratified KFold splitter
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=130)

# Melakukan validasi silang
scores = cross_val_score(pipe, X, y, cv=skf)

# Menampilkan hasil validasi silang
print("Hasil Akurasi validasi silang:")
print(scores)

```

```

Hasil Akurasi validasi silang:
[0.90821762 0.8934034 0.90789152 0.8964085 0.90373809 0.90129489
 0.90031762 0.90202785 0.90224829 0.90591398]

```

**Gambar 24.** Kode untuk membangun pipeline serta model KNN

Kode ini digunakan untuk melakukan validasi silang (*cross-validation*) dengan menggunakan model *K-Nearest Neighbors* (KNN) Classifier. Pertama, fitur-fitur numerik (`numeric_features`), biner (`binary_features`), dan kategorikal (`categorical_features`) dari dataset telah diidentifikasi. Selanjutnya, transformer (pengubah) khusus telah didefinisikan untuk masing-masing jenis fitur. Untuk fitur numerik, dilakukan penskalaan dengan `StandardScaler` dan nilai yang hilang diisi dengan rata-rata. Fitur kategorikal diisi dengan nilai yang paling sering muncul (mode) dan diubah menjadi vektor biner menggunakan `OneHotEncoder`. Fitur biner juga diisi dengan nilai yang paling sering muncul. Kemudian, keseluruhan preprocessor diimplementasikan menggunakan `ColumnTransformer`. Setelah preprocessor selesai, model KNN Classifier dengan parameter tertentu (`n_neighbors=9`, `metric='manhattan'`, `weights='distance'`) ditambahkan ke dalam pipeline.

Selanjutnya, `Stratified K-Fold Cross-Validation` digunakan dengan 10 lipatan (`StratifiedKFold(n_splits=10, shuffle=True, random_state=130)`) untuk mengevaluasi model. Setiap lipatan memisahkan data menjadi bagian latih dan validasi. Metrik akurasi digunakan untuk mengevaluasi kinerja model pada setiap lipatan. Hasil akurasi validasi silang untuk masing-masing lipatan dicetak ke layar seperti pada gambar 24. Hasilnya menunjukkan akurasi yang cukup tinggi di setiap lipatan, menunjukkan bahwa model KNN dengan konfigurasi tertentu cukup baik dalam memprediksi kelas target pada dataset yang digunakan.

```

kfold_set = pd.DataFrame(index=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], columns=['train', 'val'])
fold_num = 1
f1_scores = []
roc_auc_scores = []

for train_index, val_index in skf.split(X_train, y_train):
    train_features, val_features = X_train.iloc[train_index], X_train.iloc[val_index]
    train_labels, val_labels = y_train.iloc[train_index], y_train.iloc[val_index]
    pipe.fit(train_features, train_labels)
    pred_labels = pipe.predict(val_features)
    f1 = f1_score(val_labels, pred_labels)
    roc_auc = roc_auc_score(val_labels, pred_labels)
    f1_scores.append(f1)
    roc_auc_scores.append(roc_auc)
    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'val'] = val_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

F1-score for fold 1: 0.9298920007321985
ROC-AUC score for fold 1: 0.9246655402152147

F1-score for fold 2: 0.9195360637912287
ROC-AUC score for fold 2: 0.9126668815532273

F1-score for fold 3: 0.9230489357831545
ROC-AUC score for fold 3: 0.9167974873754146

F1-score for fold 4: 0.9310850439882699
ROC-AUC score for fold 4: 0.9260424088226106

F1-score for fold 5: 0.9272030651340996
ROC-AUC score for fold 5: 0.9214876807144885

F1-score for fold 6: 0.9208998548621189
ROC-AUC score for fold 6: 0.9142073147509097

F1-score for fold 7: 0.92802338326635
ROC-AUC score for fold 7: 0.9224716227434202

F1-score for fold 8: 0.9241128298453138
ROC-AUC score for fold 8: 0.9179459228990737

F1-score for fold 9: 0.9239723535831211
ROC-AUC score for fold 9: 0.9177490725663245

F1-score for fold 10: 0.9272394881170017
ROC-AUC score for fold 10: 0.9216843762298308

```

**Gambar 25.** Kode untuk mencari F1-score dari model KNN

Kode tersebut mengimplementasikan validasi silang (*cross-validation*) dengan menggunakan Stratified K-Fold. Pertama, DataFrame kosong `kfold\_set` dibuat dengan kolom 'train' dan 'val' untuk menyimpan indeks data latih dan data validasi untuk setiap lipatan (fold). Variabel `fold\_num` diinisialisasi dengan nilai 1. Selanjutnya, dilakukan loop melalui setiap lipatan menggunakan `skf.split(X\_train, y\_train)`. Dalam setiap iterasi lipatan, data latih dan data validasi dipisahkan berdasarkan indeks yang dihasilkan oleh Stratified K-Fold. Model yang telah diinisialisasi sebelumnya (dalam variabel `pipe`) diberi data latih dari iterasi saat ini. Model tersebut kemudian digunakan untuk membuat prediksi pada data validasi, dan metrik evaluasi, seperti F1-score dan ROC-AUC score, dihitung menggunakan fungsi `f1\_score` dan `roc\_auc\_score` dari scikit-learn.

Hasil F1-score dan ROC-AUC score untuk masing-masing lipatan dicetak ke layar. Selain itu, indeks data latih dan validasi untuk masing-masing lipatan disimpan dalam DataFrame `kfold\_set` untuk keperluan referensi atau analisis lebih lanjut. Proses ini memberikan gambaran yang lebih baik tentang kinerja model pada data latih dan data validasi yang berbeda-beda, membantu memahami sejauh mana model dapat digeneralisasikan ke data yang belum pernah dilihat sebelumnya. Hasil ini dipilih fold ke 4 dengan nilai F1-score sebesar 0.93 dan ROC-AUC sebesar 0.926.

```
[ ] X_fold_4 = X_train.iloc[kfold_set['train'].loc[4]]
    y_fold_4 = y_train.iloc[kfold_set['train'].loc[4]]
    knn = KNeighborsClassifier(metric = 'manhattan', n_neighbors = 9, weights = 'distance')
    knn.fit(X_fold_4, y_fold_4)
    y_pred = knn.predict(X_test)
    print(f"F1-score: {f1_score(y_test, y_pred)}")

F1-score: 0.927627653226494
```

**Gambar 26.** Kode untuk mengevaluasi evaluasi dengan penerapan hyperparameter tuning dan fold yang dipilih model KNN

Kode ini mengimplementasikan algoritma K-Nearest Neighbors (KNN) pada data latih dari lipatan fold keempat yang telah ditentukan sebelumnya melalui validasi silang. Dalam proses ini, data latih (X\_fold\_4 dan y\_fold\_4) diambil dari indeks yang disimpan dalam kfold\_set untuk lipatan keempat. Model KNN kemudian dilatih dengan menggunakan data latih ini. Setelah model dilatih, digunakan untuk membuat prediksi pada data uji (X\_test). Hasil prediksi (y\_pred) dibandingkan dengan nilai sebenarnya dari data uji (y\_test). Nilai F1-score dihitung sebagai metrik evaluasi. F1-score adalah ukuran presisi dan recall yang seimbang, sehingga memberikan gambaran yang baik tentang kinerja model.

Hasil F1-score sebesar 0.927627653226494 menunjukkan bahwa model KNN dengan konfigurasi tertentu dan data latih dari lipatan keempat memiliki kinerja yang sangat baik dalam memprediksi kelas target pada data uji yang digunakan. F1-score yang tinggi menandakan kemampuan model dalam memprediksi dengan baik tanpa mengorbankan presisi atau recall secara signifikan. Semakin mendekati 1, semakin baik kinerja model tersebut.

## 5. Naive Bayes

*Naive Bayes* adalah algoritma pembelajaran mesin yang digunakan untuk klasifikasi dan pemodelan probabilistik. Algoritma ini berdasarkan pada teorema Bayes dan mengasumsikan bahwa semua fitur dalam data independen satu sama lain (sehingga disebut "naive"). Dengan kata lain, *Naive Bayes* menghitung probabilitas suatu kelas berdasarkan seberapa sering fitur-fitur tertentu muncul bersama dalam data pelatihan. Keuntungan penggunaan Naive Bayes adalah bahwa metode ini hanya membutuhkan jumlah data pelatihan (Training Data) yang kecil untuk menentukan estimasi parameter yang diperlukan dalam proses pengklasifikasian (Pattekari & Parveen, 2012).



```

numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
binary_features = ['default', 'housing', 'loan']
categorical_features = ['job', 'marital', 'education', 'contact', 'month', 'outcome']

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler()), ('imputer', SimpleImputer(strategy='mean'))])
categorical_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')), ('onehot', OneHotEncoder())])
binary_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features),
        ('bin', binary_transformer, binary_features)
    ],
    remainder='drop' # Exclude other columns that are not specified
)

naive_bayes = GaussianNB(var_smoothing=1e-9)

pipe = Pipeline(steps=[('preprocessor', preprocessor), ('clf', naive_bayes)])
# initialize Stratified KFold splitter
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=130)

# Melakukan validasi silang
scores = cross_val_score(pipe, X, y, cv=skf)

# Menampilkan hasil validasi silang
print("Hasil Akurasi validasi silang:")
print(scores)

```

Hasil Akurasi validasi silang:  
[0.86024921 0.85536281 0.87490838 0.85365258 0.86098216 0.85707305  
0.8614708 0.86684583 0.86070381 0.85532747]

**Gambar 27.** Kode untuk membangun pipeline serta model *Naive Bayes*

Kode di atas digunakan untuk mempersiapkan dan melatih model klasifikasi dengan menggunakan *Gaussian Naive Bayes* (`naive_bayes`) da Pertama, kode mendefinisikan tiga jenis fitur, yaitu fitur numerik, fitur biner, dan fitur kategorikal. Masing-masing jenis fitur kemudian diproses secara berbeda menggunakan pipa (`Pipeline`) yang sesuai, termasuk normalisasi untuk fitur numerik, pengisian nilai yang paling sering muncul untuk fitur kategorikal dan biner. Kemudian, `ColumnTransformer` digunakan untuk menggabungkan transformasi-fitur yang telah didefinisikan sebelumnya dan menghilangkan kolom lain yang tidak ada dalam daftar fitur numerik, kategorikal, atau biner. Model *Gaussian Naive Bayes* kemudian didefinisikan dan dipasang dengan preprocessor dalam pipa. Selanjutnya, validasi silang dilakukan menggunakan `Stratified K-Fold` dengan 10 lipatan, dan hasil akurasi validasi silang dicetak. Hasil yang dicetak adalah akurasi model *Naive Bayes* dalam memprediksi target pada data yang diberikan dalam 10 lipatan validasi silang. Hasil *cross validation* dengan akurasi yang muncul terbaik ada pada fold ke 3.

```

kfold_set = pd.DataFrame(index=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], columns=['train', 'val'])
fold_num = 1
f1_scores = []
roc_auc_scores = []

for train_index, val_index in skf.split(X_train, y_train):
    train_features, val_features = X_train.iloc[train_index], X_train.iloc[val_index]
    train_labels, val_labels = y_train.iloc[train_index], y_train.iloc[val_index]
    pipe.fit(train_features, train_labels)
    pred_labels = pipe.predict(val_features)
    f1 = f1_score(val_labels, pred_labels)
    roc_auc = roc_auc_score(val_labels, pred_labels)
    f1_scores.append(f1)
    roc_auc_scores.append(roc_auc)
    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'val'] = val_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

F1-score for fold 1: 0.6697761194029851
ROC-AUC score for fold 1: 0.7213936136581155

F1-score for fold 2: 0.6674440298507464
ROC-AUC score for fold 2: 0.719426271461176

F1-score for fold 3: 0.6682275813676785
ROC-AUC score for fold 3: 0.7211962214644426

F1-score for fold 4: 0.6595187038360734
ROC-AUC score for fold 4: 0.7188325467060894

F1-score for fold 5: 0.6483281953995732
ROC-AUC score for fold 5: 0.7082767088667954

F1-score for fold 6: 0.6575600565237871
ROC-AUC score for fold 6: 0.7139808788117288

F1-score for fold 7: 0.6558227251296558
ROC-AUC score for fold 7: 0.7128006283109638

F1-score for fold 8: 0.666044340723454
ROC-AUC score for fold 8: 0.7185042563949653

F1-score for fold 9: 0.6629396692289775
ROC-AUC score for fold 9: 0.7153561992450483

F1-score for fold 10: 0.672706155632985
ROC-AUC score for fold 10: 0.7227469500196773

```

**Gambar 28.** Kode untuk mencari F1-score dan ROC-AUC model *Naive Bayes*

Kode tersebut digunakan untuk melakukan validasi silang (cross-validation) dengan menggunakan Stratified K-Fold (skf) pada suatu model (dalam hal ini, model yang disebut "pipe"). Setelah itu, model diuji pada data validasi ("val\_features") dan prediksi dibuat. Skor F1 dan ROC-AUC dihitung untuk setiap lipatan, dan hasilnya disimpan dalam daftar "f1\_scores" dan "roc\_auc\_scores". Selain itu, indeks data latih dan data validasi untuk setiap lipatan juga dicatat dalam DataFrame "kfold\_set". Hasil dari setiap lipatan, termasuk skor F1 dan ROC-AUC, dicetak dalam bentuk keluaran untuk memantau performa model pada masing-masing lipatan validasi silang ke-10. Hasil terbaik ada pada lipatan ke 10 sehingga yang akan dipakai adalah kfold ke 10 karena F1 Score dan ROC-AUC score untuk fold 10 memiliki performa terbaik.

```

[247] X_fold_10 = X_train.iloc[kfold_set['train'].loc[5]]
      y_fold_10 = y_train.iloc[kfold_set['train'].loc[5]]
      naive_bayes = GaussianNB(var_smoothing=1e-9)
      naive_bayes.fit(X_fold_10, y_fold_10)
      y_pred = naive_bayes.predict(X_test)
      print(f"F1-score: {f1_score(y_test, y_pred)}")

F1-score: 0.7596572552324765

```

**Gambar 29.** Kode untuk mengevaluasi evaluasi dengan penerapan hyperparameter tuning dan fold yang dipilih model *Naive Bayes*

Pada gambar diatas merupakan kode untuk melakukan fit model lagi dengan tambahan *hyperparameter* var\_smoothing dengan nilai 1e-9 yang menunjukkan peningkatan pada f1 score dari 0.67 ke 0.75

## 6. SVM

*SVM (Support Vector Machine)* adalah algoritma pembelajaran mesin yang digunakan untuk klasifikasi dan regresi. Tujuannya adalah untuk menemukan hyperplane optimal yang memisahkan dua kelas data dengan margin maksimal, di mana margin adalah jarak antara hyperplane dan titik-titik terdekat dari kedua kelas (Suyanto, 2017). SVM dapat mengatasi masalah klasifikasi yang kompleks dengan menggunakan fungsi kernel yang memetakan data ke dalam ruang fitur yang lebih tinggi, memungkinkan pemisahan yang lebih baik. Algoritma ini dikenal karena kemampuannya menangani data yang tidak linier dan memiliki kestabilan dalam menghasilkan model yang baik.

```
numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
binary_features = ['default', 'housing', 'loan']
categorical_features = ['job', 'marital', 'education', 'contact', 'month', 'outcome']

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler()), ('imputer', SimpleImputer(strategy='mean'))])
categorical_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent')), ('onehot', OneHotEncoder())])
binary_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features),
        ('bin', binary_transformer, binary_features)
    ],
    remainder='drop' # Exclude other columns that are not specified
)

pipe = Pipeline(steps=[('preprocessor', preprocessor), ('clf', SVC(kernel='linear'))])

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=130)

# Melakukan validasi silang
scores = cross_val_score(pipe, X, y, cv=skf)

# Menampilkan hasil validasi silang
print("Hasil Akurasi validasi silang:")
print(scores)
```

Hasil Akurasi validasi silang:  
[0.89469827 0.89494258 0.8964085 0.89469827 0.89836306 0.8976301  
0.8988517 0.89738578 0.89418377 0.8973607 ]

**Gambar 30.** Kode untuk membangun pipeline serta model SVM

Kode tersebut digunakan untuk melakukan validasi silang (*cross-validation*) pada model *Support Vector Classifier (SVC)* dengan parameter kernel linear. Pertama, fitur-fitur dataset telah diidentifikasi dan dibagi ke dalam tiga kelompok: numerik (`'numeric_features'`), biner (`'binary_features'`), dan kategorikal (`'categorical_features'`). Selanjutnya, dilakukan *preprocessing* pada data menggunakan transformers yang sesuai untuk masing-masing kelompok fitur. Fitur numerik dinormalisasi menggunakan `'StandardScaler'`, sedangkan fitur kategorikal diisi dengan nilai yang paling sering muncul (mode) dan diubah menjadi vektor biner menggunakan `'OneHotEncoder'`. Fitur biner juga diisi dengan nilai yang paling sering muncul. Kemudian, transformer-transformer tersebut diaplikasikan pada data menggunakan `'ColumnTransformer'` untuk menghasilkan data yang telah di-preprocess. Data yang telah di-preprocess ini kemudian digunakan untuk melatih model SVC dengan kernel linear. Model ini kemudian dinilai kinerjanya menggunakan validasi silang dengan Stratified K-Fold Cross-Validation (dengan 10 lipatan). Akurasi dari masing-masing lipatan dicetak ke layar.

Hasil validasi silang menunjukkan akurasi yang cukup stabil dan konsisten di sekitar nilai 0.895-0.899, menandakan bahwa model SVC dengan kernel linear memiliki kinerja yang baik dan dapat memprediksi kelas target pada data uji dengan

tingkat akurasi yang tinggi. Validasi silang membantu memastikan bahwa hasil evaluasi model tidak dipengaruhi oleh pemisahan data latih dan uji tertentu, menghasilkan estimasi kinerja model yang lebih reliabel.

```
kfold_set = pd.DataFrame(index=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], columns=['train', 'val'])
fold_num = 1
f1_scores = []
roc_auc_scores = []

for train_index, val_index in skf.split(X_train, y_train):
    train_features, val_features = X_train.iloc[train_index], X_train.iloc[val_index]
    train_labels, val_labels = y_train.iloc[train_index], y_train.iloc[val_index]
    pipe.fit(train_features, train_labels)
    pred_labels = pipe.predict(val_features)
    f1 = f1_score(val_labels, pred_labels)
    roc_auc = roc_auc_score(val_labels, pred_labels)
    f1_scores.append(f1)
    roc_auc_scores.append(roc_auc)
    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'val'] = val_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
```

```
F1-score for fold 1: 0.8363851151801537
ROC-AUC score for fold 1: 0.8365137473212718

F1-score for fold 2: 0.8361714621256607
ROC-AUC score for fold 2: 0.8353345031336592

F1-score for fold 3: 0.8388625592417062
ROC-AUC score for fold 3: 0.8394641800514056

F1-score for fold 4: 0.8433399602385685
ROC-AUC score for fold 4: 0.8449714222548784

F1-score for fold 5: 0.8412541904949714
ROC-AUC score for fold 5: 0.8416294563029417

F1-score for fold 6: 0.8301812450748621
ROC-AUC score for fold 6: 0.830415412258628

F1-score for fold 7: 0.8327726103304967
ROC-AUC score for fold 7: 0.8337607841935143

F1-score for fold 8: 0.8405275779376499
ROC-AUC score for fold 8: 0.8430091890323633

F1-score for fold 9: 0.8373565492679066
ROC-AUC score for fold 9: 0.8382856325421235

F1-score for fold 10: 0.8387606078547464
ROC-AUC score for fold 10: 0.8392365210547028
```

**Gambar 31.** Kode untuk mencari F1-score dan ROC-AUC model SVM

Kode ini mengimplementasikan validasi silang dengan Stratified K-Fold Cross-Validation menggunakan variabel `skf` yang telah diinisialisasi sebelumnya. Pertama, sebuah `DataFrame` kosong dengan nama `kfold_set` dibuat untuk menyimpan indeks data latih dan data validasi dari setiap lipatan. Variabel `fold_num` diinisialisasi dengan nilai 1. Dalam setiap iterasi dari loop, `skf.split(X_train, y_train)` memberikan indeks untuk data latih dan data validasi. Data latih dan validasi dipisahkan dari `X_train` dan `y_train` menggunakan indeks tersebut. Model yang telah diinisialisasi (`pipe`) kemudian dilatih dengan data latih, dan digunakan untuk membuat prediksi pada data validasi.

Selanjutnya, F1-score dan ROC-AUC score dihitung menggunakan prediksi model dan nilai sebenarnya dari data validasi. Nilai-nilai F1-score dan ROC-AUC score untuk setiap lipatan dicetak ke layar. Dimana nilai fold 4 adalah yang terbaik dengan F1-score sebesar 0.9 dan ROC-AUC sebesar 0.9.

## 7. Neural Network

*Neural Network* (Jaringan Saraf Tiruan) adalah model pembelajaran mesin yang terdiri dari lapisan-lapisan (atau "layers") dari neuron buatan yang meniru konsep jaringan saraf manusia. Ada tiga jenis lapisan utama dalam Neural Network. Pertama, lapisan input menerima data mentah sebagai inputnya. Lalu, ada lapisan-lapisan tersembunyi (*hidden layers*), yang melakukan komputasi dan ekstraksi fitur secara bertahap. Setiap neuron dalam lapisan ini menerima sinyal dari neuron di lapisan sebelumnya dan menghasilkan output yang akan digunakan sebagai input untuk lapisan berikutnya. Terakhir, lapisan output mengeluarkan hasil prediksi dari jaringan. Melalui pelatihan, *Neural Network* mengubah bobot dan bias di setiap neuron untuk mempelajari pola dan hubungan dalam data, yang memungkinkannya untuk menyelesaikan berbagai tugas, seperti pengenalan gambar, klasifikasi teks, dan banyak lagi, dengan tingkat keberhasilan yang tinggi. Secara sederhana NN adalah sebuah alat pemodelan data statistik non-linear. NN dapat digunakan untuk memodelkan hubungan yang kompleks antara input dan output untuk menemukan pola-pola pada data (Muhandis, 2018).

Dalam Neural Network, pemberian *weight* (bobot) dan bias adalah proses yang memungkinkan jaringan untuk mempelajari dan menyesuaikan diri dengan data pelatihan. Bobot adalah parameter yang mengatur seberapa besar pengaruh setiap input pada neuron, dan mereka diperbarui selama pelatihan dengan menggunakan algoritma pembelajaran yang mengurangi kesalahan prediksi. Bias adalah parameter tambahan yang memungkinkan jaringan untuk menghasilkan keluaran bahkan jika inputnya adalah nol, juga diperbarui selama pelatihan. Kombinasi bobot dan bias memungkinkan jaringan untuk memodelkan pola yang kompleks dalam data, sehingga dapat menghasilkan prediksi yang akurat. Proses perbaruan bobot dan bias dilakukan untuk mengoptimalkan kinerja jaringan selama pelatihan dengan menggunakan teknik seperti backpropagation dan berbagai algoritma optimasi.

```
numeric_features = ['age', 'balance', 'day', 'duration', 'campaign', 'pdays', 'previous']
binary_features = ['default', 'housing', 'loan']
categorical_features = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']

numeric_transformer = Pipeline(steps=[('scaler', StandardScaler()), ('imputer', SimpleImputer(strategy='mean'))])
categorical_transformer = Pipeline(steps=[('loader', LoadingNumpy()), ('imputer', SimpleImputer(strategy='most_frequent')), ('onehot', OneHotEncoder())])
binary_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='most_frequent'))])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features),
        ('bin', binary_transformer, binary_features)
    ],
    remainder='drop' # Exclude other columns that are not specified
)

nn = MLPClassifier(activation='relu', solver='sgd', alpha=0.01, max_iter=1000)

pipe = Pipeline(steps=[('preprocessor', preprocessor), ('clf', nn)])

skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=130)

# Melakukan validasi silang
scores = cross_val_score(pipe, X, y, cv=skf)

# Menampilkan hasil validasi silang
print("Hasil Akurasi validasi silang:")
print(scores)
```

Hasil Akurasi validasi silang:  
[0.90935744 0.90813584 0.911312 0.89982898 0.91302223 0.9076472  
0.90935744 0.9088688 0.90347019 0.91324536]

Gambar 32. Kode untuk membangun pipeline serta model NN

Hasil akurasi validasi silang dicetak pada akhir kode. Ini memberikan gambaran seberapa baik model performa pada setiap fold yang berbeda, memberikan wawasan yang lebih baik tentang kinerja model secara keseluruhan. Dalam kasus ini, nilai akurasi berkisar antara sekitar 89.4% hingga 89.8% pada setiap lipatan, menunjukkan bahwa model tersebut memiliki kinerja yang konsisten.

```
kfold_set = pd.DataFrame(index=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], columns=['train', 'val'])
fold_num = 1
f1_scores = []
roc_auc_scores = []

for train_index, val_index in skf.split(X_train, y_train):
    train_features, val_features = X_train.iloc[train_index], X_train.iloc[val_index]
    train_labels, val_labels = y_train.iloc[train_index], y_train.iloc[val_index]
    pipe.fit(train_features, train_labels)
    pred_labels = pipe.predict(val_features)
    f1 = f1_score(val_labels, pred_labels)
    roc_auc = roc_auc_score(val_labels, pred_labels)
    f1_scores.append(f1)
    roc_auc_scores.append(roc_auc)
    kfold_set.at[fold_num, 'train'] = train_index
    kfold_set.at[fold_num, 'val'] = val_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

F1-score for fold 1: 0.896883984845979
ROC-AUC score for fold 1: 0.8943587942944213
F1-score for fold 2: 0.8960396039603961
ROC-AUC score for fold 2: 0.892589695786892
F1-score for fold 3: 0.8952164009111618
ROC-AUC score for fold 3: 0.8914099097383555

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686:
  warnings.warn(
F1-score for fold 4: 0.9031888485774299
ROC-AUC score for fold 4: 0.9082617497896887
F1-score for fold 5: 0.8893991580558744
ROC-AUC score for fold 5: 0.8862821249989549
F1-score for fold 6: 0.8891006097560976
ROC-AUC score for fold 6: 0.8854943366244419
F1-score for fold 7: 0.8929804372842347
ROC-AUC score for fold 7: 0.8982172738450543
F1-score for fold 8: 0.9005736137667303
ROC-AUC score for fold 8: 0.897692549237973
F1-score for fold 9: 0.9022499527320854
ROC-AUC score for fold 9: 0.8982884740261288
F1-score for fold 10: 0.9006471259992386
ROC-AUC score for fold 10: 0.89728453364817
```

**Gambar 33.** Kode untuk mencari F1-score dan ROC-AUC model NN

Dengan menggunakan konsep yang sama dengan langkah pencarian F1-score dan ROC-AUC pada metode-metode sebelumnya dapat dipilih fold terbaik pada model *Neural Network* dengan nilai F1-score dan ROC-AUC sebesar 0.9 yang dimana nilainya mendekati 1 dan lebih tinggi dari fold lainnya.

## D. Backward Selection

### 1. Logistic Regression

```
logreg_bs = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'Features'])
X_fold_5 = X_train.iloc[kfold_set['train'].loc[5]]
y_fold_5 = y_train.iloc[kfold_set['train'].loc[5]]

f1_scores = [] # Inisialisasi list untuk menyimpan F1-score

for i in range(16, 9, -1):
    logreg = LogisticRegression(max_iter=500)
    sbs = SF5(logreg, k_features=i, forward=False, floating=False, cv=0, scoring='f1')
    sbs.fit(X_fold_5, y_fold_5)

    X_train_sfs = sbs.transform(X_fold_5)
    X_test_sfs = sbs.transform(X_test)
    logreg.fit(X_train_sfs, y_fold_5)
    y_pred = logreg.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    f1_scores.append((i, f1, sbs.k_feature_names_)) # Menggunakan i untuk jumlah fitur
    features = str(sbs.k_feature_names_)
    logreg_bs.loc[i] = [f1, features]

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

best_combination = max(f1_scores, key=lambda x: x[1])
print(f"Kombinasi fitur terbaik: {best_combination[0]} fitur, F1-score: {best_combination[1]}")
print(f>Nama fitur yang terpilih: {best_combination[2]}")

F1-score dengan 16 fitur: 0.8865743132190655
F1-score dengan 15 fitur: 0.8892160719363174
F1-score dengan 14 fitur: 0.8891531698764805
F1-score dengan 13 fitur: 0.8898666416896922
F1-score dengan 12 fitur: 0.8895772540216984
F1-score dengan 11 fitur: 0.8894881631889211
F1-score dengan 10 fitur: 0.888428939358464
Kombinasi fitur terbaik: 12 fitur, F1-score: 0.8895772540216984
Nama fitur yang terpilih: ('age', 'job', 'education', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'outcome')
```

**Gambar 34.** Kode untuk melakukan backward selection

Kode di atas merupakan kode untuk melakukan *backward selection* menggunakan *Sequential Feature Selection* (SFS) dengan *model Logistic Regression*. Proses ini membantu kita memahami bagaimana memilih fitur-fitur yang paling relevan untuk meningkatkan kinerja model. Pertama yang dilakukan adalah membuat DataFrame kosong dengan nama `logreg_bs` yang akan digunakan untuk menyimpan hasil evaluasi model, seperti *F-1 Score* dan nama fitur yang terpilih. Pada proses ini, kami ingin mengetahui bagaimana performa dari model *logistic regression* ketika jumlah fitur yang digunakan berbeda. Dapat dilihat hasil akhir *f1 score* tertinggi adalah nilai 0,8095772540216984 dengan 12 fitur yakni *age, job, education, balance, housing, loan, contact, day, month, duration, campaign*, dan *poutcome*.

## 2. Decision Tree

```
# Inisialisasi DataFrame untuk menyimpan hasil
d_tree_bs = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'Features'])

X_fold_1 = X_train.iloc[kfold_set['train'].loc[1]]
y_fold_1 = y_train.iloc[kfold_set['train'].loc[1]]

f1_scores = [] # Inisialisasi list untuk menyimpan F1-score

for i in range(10, 9, -1):
    d_tree = DecisionTreeClassifier() # Menggunakan DecisionTreeClassifier
    sbs = SFS(d_tree, k_features=i, forward=False, floating=False, cv=0, scoring='f1')
    sbs.fit(X_fold_1, y_fold_1)

    X_train_sfs = sbs.transform(X_fold_1)
    X_test_sfs = sbs.transform(X_test)
    d_tree.fit(X_train_sfs, y_fold_1)
    y_pred = d_tree.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    f1_scores.append((i, f1, sbs.k_feature_names_)) # Menggunakan i untuk jumlah fitur
    features = str(sbs.k_feature_names_)
    d_tree_bs.loc[i] = [f1, features]

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

best_combination = max(f1_scores, key=lambda x: x[1])
print(f"Kombinasi fitur terbaik: {best_combination[0]} fitur, F1-score: {best_combination[1]}")
print(f>Nama fitur yang terpilih: {best_combination[2]}")

F1-score dengan 16 fitur: 0.959169244365886
F1-score dengan 15 fitur: 0.9582138287875851
F1-score dengan 14 fitur: 0.9598656294200848
F1-score dengan 13 fitur: 0.9581916913160567
F1-score dengan 12 fitur: 0.95811445000030895
F1-score dengan 11 fitur: 0.9366793365833967
F1-score dengan 10 fitur: 0.9380454506178173
Kombinasi fitur terbaik: 14 fitur, F1-score: 0.9598656294200848
Nama fitur yang terpilih: ('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays')
```

**Gambar 35.** kode untuk backward selection dengan model *decision tree*

Kode diatas juga digunakan untuk melakukan seleksi fitur dengan metode SFS pada model *Decision Tree*. langkah awal yang kami lakukan adalah membuat DataFrame bernama `d_tree_bs` yang akan digunakan untuk menyimpan nilai *F1 Score* dan fitur - fitur yang akan dipilih. Proses seleksi fitur ini dilakukan dengan mencoba berbagai kombinasi fitur dan memilih kombinasi yang menghasilkan *f1 score* tertinggi, yang digunakan sebagai matrik evaluasi. Hasil dari proses seleksi fitur disimpan dalam DataFrame yang disebut '`d_tree_bs`', yang berisi *f1 score* dan daftar fitur yang dipilih. Dari hasil proses diatas didapatkan *f1 score* tertinggi adalah 0,9598656294200848 dengan 14 fitur yakni *age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays*.

### 3. Random Forest

```
# Inisialisasi DataFrame untuk menyimpan hasil
rf_bs = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'Features'])

X_fold_4 = X_train.iloc[kfold_set['train'].loc[4]]
y_fold_4 = y_train.iloc[kfold_set['train'].loc[4]]

f1_scores = [] # Inisialisasi list untuk menyimpan F1-score

for i in range(16, 9, -1):
    rf = RandomForestClassifier() # Menggunakan RandomForestClassifier
    sbs = SFS(rf, k_features=i, forward=False, floating=False, cv=0, scoring='f1')
    sbs.fit(X_fold_4, y_fold_4)

    X_train_sfs = sbs.transform(X_fold_4)
    X_test_sfs = sbs.transform(X_test)
    rf.fit(X_train_sfs, y_fold_4)
    y_pred = rf.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    f1_scores.append((i, f1, sbs.k_feature_names_)) # Menggunakan i untuk jumlah fitur
    features = str(sbs.k_feature_names_)
    rf_bs.loc[i] = [f1, features]

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

best_combination = max(f1_scores, key=lambda x: x[1])
print(f"Kombinasi fitur terbaik: {best_combination[0]} fitur, F1-score: {best_combination[1]}")
print(f>Nama fitur yang terpilih: {best_combination[2]}")

F1-score dengan 16 fitur: 0.9668473913429918
F1-score dengan 15 fitur: 0.9670652028979065
F1-score dengan 14 fitur: 0.9666474219478616
F1-score dengan 13 fitur: 0.9665867519545132
F1-score dengan 12 fitur: 0.9654866471472467
F1-score dengan 11 fitur: 0.96616508787997507
F1-score dengan 10 fitur: 0.9664232276451727
Kombinasi fitur terbaik: 15 fitur, F1-score: 0.9670652028979065
Nama fitur yang terpilih: ('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous')
```

**Gambar 36.** Kode untuk backward selection dengan model *random forest*

Kami melakukan kode yang mirip dengan kode sebelumnya hanya berbeda dengan pemilihan fold. Kombinasi fitur terpilih untuk Random Forest dengan backward selection adalah 15 fitur dengan F1-Score yang mencapai 0.967 serta fitur yang dipakai hampir semua kecuali fitur *poutcome*. Model ini kami pilih sebagai model dengan performa terbaik yaitu F1-Score sebesar 0.967

### 4. KNN

```
# Inisialisasi DataFrame untuk menyimpan hasil
knn_bs = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'Features'])

X_fold_4 = X_train.iloc[kfold_set['train'].loc[4]]
y_fold_4 = y_train.iloc[kfold_set['train'].loc[4]]

f1_scores = [] # Inisialisasi list untuk menyimpan F1-score

for i in range(16, 9, -1):
    knn = KNeighborsClassifier(metric='manhattan', n_neighbors=9, weights='distance')
    sbs = SFS(knn, k_features=i, forward=False, floating=False, cv=0, scoring='f1')
    sbs.fit(X_fold_4, y_fold_4)

    X_train_sfs = sbs.transform(X_fold_4)
    X_test_sfs = sbs.transform(X_test)
    knn.fit(X_train_sfs, y_fold_4)
    y_pred = knn.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    f1_scores.append((i, f1, sbs.k_feature_names_)) # Menggunakan i untuk jumlah fitur
    features = str(sbs.k_feature_names_)
    knn_bs.loc[i] = [f1, features]

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

best_combination = max(f1_scores, key=lambda x: x[1])
print(f"Kombinasi fitur terbaik: {best_combination[0]} fitur, F1-score: {best_combination[1]}")
print(f>Nama fitur yang terpilih: {best_combination[2]}")

F1-score dengan 16 fitur: 0.927627653226494
F1-score dengan 15 fitur: 0.926908177599318
F1-score dengan 14 fitur: 0.9256219434403572
F1-score dengan 13 fitur: 0.9195945945945946
F1-score dengan 12 fitur: 0.921150592216582
F1-score dengan 11 fitur: 0.8836420766137223
F1-score dengan 10 fitur: 0.8859727769174341
Kombinasi fitur terbaik: 16 fitur, F1-score: 0.927627653226494
Nama fitur yang terpilih: ('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'day', 'month', 'duration', 'campaign', 'pdays', 'previous', 'poutcome')
```

**Gambar 37.** kode untuk backward selection dengan model *KNN*

Kami melakukan kode yang mirip dengan kode sebelumnya hanya berbeda dengan pemilihan fold. Kombinasi fitur terpilih untuk KNN dengan backward selection adalah 16 fitur dengan F1-Score yang mencapai 0.9276 yang berarti semua fitur dipakai.



## 5. Naive Bayes

```
# Inisialisasi DataFrame untuk menyimpan hasil
nb_bs = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'Features'])

X_fold_4 = X_train.iloc[kfold_set['train'].loc[4]]
y_fold_4 = y_train.iloc[kfold_set['train'].loc[4]]

f1_scores = [] # Inisialisasi list untuk menyimpan F1-score

for i in range(16, 9, -1):
    nb = GaussianNB()
    sbs = SFS(nb, k_features=i, forward=False, floating=False, cv=0, scoring='f1')
    sbs.fit(X_fold_4, y_fold_4)

    X_train_sfs = sbs.transform(X_fold_4)
    X_test_sfs = sbs.transform(X_test)
    nb.fit(X_train_sfs, y_fold_4)
    y_pred = nb.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    f1_scores.append((i, f1, sbs.k_feature_names_)) # Menggunakan i untuk jumlah fitur
    features = str(sbs.k_feature_names_)
    nb_bs.loc[i] = [f1, features]

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

best_combination = max(f1_scores, key=lambda x: x[1])
print(f"Kombinasi fitur terbaik: {best_combination[0]} fitur, F1-score: {best_combination[1]}")
print(f>Nama fitur yang terpilih: {best_combination[2]}")

F1-score dengan 16 fitur: 0.7595268153551222
F1-score dengan 15 fitur: 0.7814877430262045
F1-score dengan 14 fitur: 0.7843503352287192
F1-score dengan 13 fitur: 0.782542758446373
F1-score dengan 12 fitur: 0.7875116083668686
F1-score dengan 11 fitur: 0.7879217214203489
F1-score dengan 10 fitur: 0.7872274556631728
Kombinasi fitur terbaik: 11 fitur, F1-score: 0.7879217214203489
Nama fitur yang terpilih: ('age', 'marital', 'education', 'balance', 'housing', 'loan', 'contact', 'day', 'duration', 'campaign', 'previous')
```

**Gambar 38.** kode untuk backward selection dengan model *Naive Bayes*

Kami melakukan kode yang mirip dengan kode sebelumnya hanya berbeda dengan pemilihan fold. Kombinasi fitur terpilih untuk Naive Bayes dengan backward selection adalah 11 fitur dengan F1-Score yang mencapai 0.78. Fitur yang dipakai adalah ('age', 'marital', 'education', 'balance', 'housing', 'loan', 'contact', 'day', 'duration', 'campaign previous'). Hal ini terjadi karena data yang digunakan terbilang besar. Sedangkan Naive Bayes itu cocok digunakan untuk memprediksi data yang memiliki data points sedikit karena berdasarkan probabilitas.

## 6. SVM

```
# Inisialisasi DataFrame untuk menyimpan hasil
svm_bs = pd.DataFrame(index=range(10, 17), columns=['F1-Score', 'Features'])

X_fold_4 = X_train.iloc[kfold_set['train'].loc[4]]
y_fold_4 = y_train.iloc[kfold_set['train'].loc[4]]

f1_scores = [] # Inisialisasi list untuk menyimpan F1-score

for i in range(16, 9, -1):
    svm = SVC(kernel="linear")
    sbs = SFS(svm, k_features=i, forward=False, floating=False, cv=0, scoring='f1')
    sbs.fit(X_fold_4, y_fold_4)

    X_train_sfs = sbs.transform(X_fold_4)
    X_test_sfs = sbs.transform(X_test)
    svm.fit(X_train_sfs, y_fold_4)
    y_pred = svm.predict(X_test_sfs)

    f1 = f1_score(y_test, y_pred)
    f1_scores.append((i, f1, sbs.k_feature_names_)) # Menggunakan i untuk jumlah fitur
    features = str(sbs.k_feature_names_)
    svm_bs.loc[i] = [f1, features]

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

best_combination = max(f1_scores, key=lambda x: x[1])
print(f"Kombinasi fitur terbaik: {best_combination[0]} fitur, F1-score: {best_combination[1]}")
print(f>Nama fitur yang terpilih: {best_combination[2]}")
```

[89] 380m 34.6s

```
... F1-score dengan 16 fitur: 0.8082789693527949
F1-score dengan 15 fitur: 0.808662564006201
F1-score dengan 14 fitur: 0.808084606345476
F1-score dengan 13 fitur: 0.8074283027738599
```

**Gambar 39.** kode untuk backward selection dengan model *SVM*

Pada SVM, kode yang dibuat membutuhkan runtime yang sangat lama (380 menit), sehingga kami memutuskan untuk tidak menggunakan SVM model untuk pemilihan fitur menggunakan backward selection. SVM (Support Vector Machine) dapat membutuhkan waktu eksekusi yang lama karena algoritmanya menghadapi beberapa tantangan. Ini termasuk kompleksitas perhitungan untuk menemukan hyperplane optimal dalam data yang tidak selalu terdistribusi secara linear, pengolahan dan transformasi data sebelum pelatihan, serta pencarian parameter yang optimal. Masalah yang sangat besar dan kompleks juga dapat memperlambat SVM.

## 7. Neural Network

```
print("Kombinasi fitur terbaik: {best_combination[0]} fitur, F1-score: {best_combination[1]}.")  
print("Nama Fitur yang terpilih: {best_combination[2]}")
```

**Gambar 40.** Kode untuk backward selection dengan model *NN*

Runtime yang dibutuhkan untuk kode dengan model NN sangatlah lama sehingga kami memutuskan untuk tidak menggunakan backward feature selection

untuk NN. Kode yang diberikan adalah bagian dari proses seleksi fitur menggunakan metode Sequential Feature Selection (SFS) untuk meningkatkan performa model klasifikasi Multilayer Perceptron (MLP). Proses ini dilakukan dengan mencoba berbagai kombinasi fitur dengan jumlah yang berbeda, dimulai dari 16 fitur dan berkurang hingga 10 fitur. SFS secara berulang melibatkan pelatihan dan pengujian model MLP untuk setiap kombinasi fitur. Selain itu, untuk setiap kombinasi, F1-score dihitung dengan menggunakan data uji. Proses ini membutuhkan waktu yang lama, jumlah fitur awalnya besar serta dataset memiliki ukuran yang signifikan.

## E. Kesimpulan

Dalam melakukan klasifikasi terhadap dataset terkait kampanye direct marketing produk deposito bank berjangka dari suatu institusi perbankan di Portugal dari Mei 2008 hingga November 2010 dengan jumlah 45.211 baris dengan 16 atribut (variabel independent) dan 1 atribut target (Variabel dependent) kami menggunakan 7 metode klasifikasi yaitu dengan *Logistic Regression*, *Decision Tree*, *Random Forest*, *KNN*, *Naive Bayes*, *SVM*, dan *Neural Network*. Dalam melakukan klasifikasi, kelompok kami memiliki kendala pada waktu run hasil pada pencarian parameter menggunakan *Grid Search* dan *Random Search* serta beberapa model klasifikasi juga membutuhkan waktu yang lama saat melakukan sesi *running*. Namun, dari pengolahan dan analisis yang telah dilakukan didapatkan kesimpulan bahwa model klasifikasi terbaik adalah dengan metode *Random Forest* dan yang terburuk adalah *Naive Bayes*.

*Random Forest* menjadi metode klasifikasi terbaik, dikarenakan pada algoritma ini untuk menentukan hasil akurasi prediksi yang besar. Dimana prediksi tersebut dilakukan dengan mengambil rata-rata atau rata-rata keluaran dari berbagai hasil *Decision Tree* dan menambah jumlah pohon akan meningkatkan ketepatan hasilnya. Pada klasifikasi yang telah dilakukan pada dataset yang mana memiliki nilai korelasi data yang rendah (dibawah 0,5) didapatkan bahwa nilai akurasi dari *Random Forest* sekitar 0.96. Nilai tersebut mendekati nilai satu menunjukkan bahwa *Random Forest* memiliki kemampuan yang sangat baik dalam mengklasifikasikan data, terutama pada kasus positif dan memiliki keseimbangan yang baik antara mengidentifikasi kasus positif dan menghindari kesalahan identifikasi.

Sementara *Naive Bayes* menjadi metode klasifikasi terburuk dikarenakan pada algoritma ini menerapkan teknik supervised klasifikasi objek di masa depan dengan menetapkan label kelas ke instance/catatan menggunakan probabilitas. Metode *Naive Bayes* juga memiliki asumsi bahwa semua variabel independen dalam dataset adalah independen satu sama lain, yang dikenal sebagai asumsi "naive" dimana variabel dalam dataset seringkali saling terkait atau bergantung satu sama lain. Oleh karena itu, asumsi ini sering tidak terpenuhi pada data yang kompleks. Selain itu, *Naive Bayes* memiliki hasil akurasi yang paling rendah diantara metode lainnya yaitu sekitar 0.78.

## F. Daftar Pustaka

- Cahyana, C. W., & Nurlayli, A. (2023). Analisis Performa Logistic Regression, Naïve Bayes, dan Random Forest Sebagai Algoritma Pendeteksi Kanker Payudara. Departemen Pendidikan Teknik Elektronika dan Informatika, Fakultas Teknik, Universitas Negeri Yogyakarta.
- Haristu, R. A. (2019). Penerapan Metode Random Forest Untuk Prediksi Win Ratio Pemain Player Unknown Battleground. Yogyakarta: Skripsi.
- Kulkarni, V. Y., & Sinha, P. K. (2014). Effective Learning and Classification using Random Forest Algorithm, 3(11), 267–273.
- Muhandis Al Farhany. (2018). Perbaikan Faktor Daya Dalam Jaringan 3 Phase Menggunakan Neural Network. Jember : Jurusan Teknik Elektro Universitas Jember
- Ochiai, Y., Masuma, Y., & Tomii, N. (2019). Improvement of timetable robustness by analysis of drivers' operation based on decision trees. *Journal of Rail Transport Planning & Management*, 9(March), 57–65.
- Pattekari, S. A., Parveen, A. (2012). Prediction System for Heart Disease Using Naive Bayes, *International Journal of Advanced Computer and Mathematical Sciences*, ISSN 2230-9624, Vol. 3, No 3, Hal 290-294.
- Suyanto.(2017).Data Mining Untuk Klasifikasi dan Klasterisasi Data. Bandung: Informatika.
- Ulya, S., Soeleman, M.A., & Budiman, F. (2021). Optimasi Parameter K Pada Algoritma K-NN Untuk Klasifikasi Prioritas Bantuan Pembangunan Desa. [Optimization of K Parameters in the K-NN Algorithm for Priority Classification of Village Development Assistance]. Magister Teknik Informatika, Fakultas Ilmu Komputer, Universitas Dian Nuswantoro.