 Project Bimbingan Karir Data Science

Table of Contents

1) Pengumpulan Data

2) Menelaah Data

3) Validasi Data

4) Menetukan Object Data

5) Membersihkan Data

6) Konstruksi Data

7) Modelling

8) Evaluasi

9) Streamlit

10) Kesimpulan

 1) Pengumpulan Data

Dataset yang digunakan adalah dataset yang bersumber dari link berikut :C

Dataset yang dipakai adalah dataset dengan nama file "Hungarian.data" diharapkan sebelum memakai dataset tersebut anda dapat membaca deskripsi dataset yang ada di dalam file "heart-disease.names"

 2) Menelaah Data

pilih dan masukan library yang anda butuhkan untuk menelaah data

import pandas as pd

import re

import numpy as np

import itertools

 Load Data

masukkan dataset yang dibutuhkan dengan alamat penyimpanan yang tepat dan simpan kedalam sebuah variabel dir = 'hungarian.data'

buatlah iterasi untuk membaca dataset

with open(dir, encoding='Latin1') as file:

lines = [line.strip() for line in file]

lines[0:10]

['1254 0 40 1 1 0 0',

'-9 2 140 0 289 -9 -9 -9',

'0 -9 -9 0 12 16 84 0',

'0 0 0 0 150 18 -9 7',

'172 86 200 110 140 86 0 0',

'0 -9 26 20 -9 -9 -9 -9',

'-9 -9 -9 -9 -9 -9 -9 12',

'20 84 0 -9 -9 -9 -9 -9',

'-9 -9 -9 -9 -9 1 1 1',

'1 1 -9. -9. name']

setelah membaca file dataset lakukan iterasi sesuai jumlah kolom dan baris yang ada pada dataset. Untuk keterangan kolom dan baris dapat dilihat melalui deskripsi dataset yang sudah dijelaskan sebelumnya

data = itertools.takewhile(

lambda x: len(x) == 76,

(' '.join(lines[i:(i + 10)]).split() for i in range(0, len(lines), 10))

)

df = pd.DataFrame.from\_records(data)

df.head()

**0 1 2 3 4 5 6 7 8 9 ... 66 67 68 69 70 71 72 73 74 75**

0 1254 0 40 1 1 0 0 -9 2 140 ... -9 -9 1 1 1 1 1 -9. -9. name

1 1255 0 49 0 1 0 0 -9 3 160 ... -9 -9 1 1 1 1 1 -9. -9. name

2 1256 0 37 1 1 0 0 -9 2 130 ... -9 -9 1 1 1 1 1 -9. -9. name

3 1257 0 48 0 1 1 1 -9 4 138 ... 2 -9 1 1 1 1 1 -9. -9. name

4 1258 0 54 1 1 0 1 -9 3 150 ... 1 -9 1 1 1 1 1 -9. -9. name

5 rows × 76 columns

menampilan informasi dari file dataset yang sudah dimasukkan kedalam dataframe

df.info()

3 3 o u object

33 33 294 non-null object

34 34 294 non-null object

35 35 294 non-null object

36 36 294 non-null object

37 37 294 non-null object

38 38 294 non-null object

39 39 294 non-null object

40 40 294 non-null object

41 41 294 non-null object

42 42 294 non-null object

43 43 294 non-null object

44 44 294 non-null object

45 45 294 non-null object

46 46 294 non-null object

47 47 294 non-null object

48 48 294 non-null object

49 49 294 non-null object

50 50 294 non-null object

51 51 294 non-null object

52 52 294 non-null object

53 53 294 non-null object

54 54 294 non-null object

55 55 294 non-null object

56 56 294 non-null object

57 57 294 non-null object

58 58 294 non-null object

59 59 294 non-null object

60 60 294 non-null object

61 61 294 non-null object

62 62 294 non-null object

63 63 294 non-null object

64 64 294 non-null object

65 65 294 non-null object

66 66 294 non-null object

67 67 294 non-null object

68 68 294 non-null object

69 69 294 non-null object

70 70 294 non-null object

71 71 294 non-null object

72 72 294 non-null object

73 73 294 non-null object

74 74 294 non-null object

75 75 294 non-null object

dtypes: object(76)

Pada kondisi dataset yang kita miliki terdapat kondisi khusus yang dimana sebelum memasuki tahap validasi data untuk tipe data object atau string perlu dilakukan penghapusan fitur dikarenakan pada dataset ini nilai null disimbolkan dengan angka -9.0

df = df.iloc[:,:-1]

df = df.drop(df.columns[0], axis=1)

mengubah tipe data file dataset menjadi tipe data float sesuai dengan nilai null yaitu -9.0

df = df.astype(float)

df.info()

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

RangeIndex: 294 entries, 0 to 293

Data columns (total 74 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 1 294 non-null float64

1 2 294 non-null float64

2 3 294 non-null float64

3 4 294 non-null float64

4 5 294 non-null float64

5 6 294 non-null float64

6 7 294 non-null float64

7 8 294 non-null float64

8 9 294 non-null float64

9 10 294 non-null float64

10 11 294 non-null float64

11 12 294 non-null float64

12 13 294 non-null float64

13 14 294 non-null float64

14 15 294 non-null float64

15 16 294 non-null float64

16 17 294 non-null float64

17 18 294 non-null float64

18 19 294 non-null float64

19 20 294 non-null float64

20 21 294 non-null float64

21 22 294 non-null float64

22 23 294 non-null float64

23 24 294 non-null float64

24 25 294 non-null float64

25 26 294 non-null float64

26 27 294 non-null float64

27 28 294 non-null float64

28 29 294 non-null float64

29 30 294 non-null float64

30 31 294 non-null float64

31 32 294 non-null float64

32 33 294 non-null float64

33 34 294 non-null float64

34 35 294 non-null float64

35 36 294 non-null float64

36 37 294 non-null float64

37 38 294 non-null float64

38 39 294 non-null float64

39 40 294 non-null float64

40 41 294 non-null float64

41 42 294 non-null float64

42 43 294 non-null float64

43 44 294 non-null float64

44 45 294 non-null float64

45 46 294 non-null float64

46 47 294 non-null float64

47 48 294 non-null float64

48 49 294 non-null float64

49 50 294 non-null float64

50 51 294 non-null float64

51 52 294 non-null float64

52 53 294 non-null float64

 3) Validasi Data

Pada tahap ini bertujuan untuk mengetahui dan memahami isi dari dataset agar dapat dilakukan penanganan sesuai dengan kondisinya

mengubah nilai -9.0 menjadi nilai null value sesuai dengan deskripsi dataset

df.replace(-9.0, np.nan, inplace=True)

menghitung jumlah nilai null value

df.isnull().sum()

1 0

2 0

3 0

4 0

5 0

...

70 0

71 0

72 0

73 266

74 294

Length: 74, dtype: int64

df.head()

**1 2 3 4 5 6 7 8 9 10 ... 65 66 67 68 69 70 71 72 73 74** 0 0.0 40.0 1.0 1.0 0.0 0.0 NaN 2.0 140.0 0.0 ... NaN NaN NaN 1.0 1.0 1.0 1.0 1.0 NaN NaN 1 0.0 49.0 0.0 1.0 0.0 0.0 NaN 3.0 160.0 1.0 ... NaN NaN NaN 1.0 1.0 1.0 1.0 1.0 NaN NaN 2 0.0 37.0 1.0 1.0 0.0 0.0 NaN 2.0 130.0 0.0 ... NaN NaN NaN 1.0 1.0 1.0 1.0 1.0 NaN NaN 3 0.0 48.0 0.0 1.0 1.0 1.0 NaN 4.0 138.0 0.0 ... NaN 2.0 NaN 1.0 1.0 1.0 1.0 1.0 NaN NaN 4 0.0 54.0 1.0 1.0 0.0 1.0 NaN 3.0 150.0 0.0 ... NaN 1.0 NaN 1.0 1.0 1.0 1.0 1.0 NaN NaN 5 rows × 74 columns

df.info()

16 17 1 non-null float64

17 18 293 non-null float64

18 19 294 non-null float64

19 20 294 non-null float64

20 21 294 non-null float64

21 22 293 non-null float64

22 23 292 non-null float64

23 24 293 non-null float64

24 25 293 non-null float64

25 26 293 non-null float64

26 27 285 non-null float64

27 28 292 non-null float64

28 29 104 non-null float64

29 30 292 non-null float64

30 31 293 non-null float64

31 32 293 non-null float64

32 33 293 non-null float64

33 34 293 non-null float64

34 35 293 non-null float64

35 36 293 non-null float64

36 37 293 non-null float64

37 38 292 non-null float64

38 39 294 non-null float64

39 40 104 non-null float64

40 41 293 non-null float64

41 42 294 non-null float64

42 43 4 non-null float64

43 44 0 non-null float64

44 45 0 non-null float64

45 46 0 non-null float64

46 47 3 non-null float64

47 48 0 non-null float64

48 49 2 non-null float64

49 50 28 non-null float64

50 51 27 non-null float64

51 52 17 non-null float64

52 53 0 non-null float64

53 54 294 non-null float64

54 55 294 non-null float64

55 56 294 non-null float64

56 57 294 non-null float64

57 58 19 non-null float64

58 59 58 non-null float64

59 60 48 non-null float64

60 61 18 non-null float64

61 62 59 non-null float64

62 63 9 non-null float64

63 64 23 non-null float64

64 65 5 non-null float64

65 66 50 non-null float64

66 67 25 non-null float64

67 68 294 non-null float64

68 69 294 non-null float64

69 70 294 non-null float64

70 71 294 non-null float64

71 72 294 non-null float64

72 73 28 non-null float64

73 74 0 non-null float64

dtypes: float64(74)

 4) Menentukan Object Data

Memilih 14 fitur yang akan digunakan sesuai dengan deskripsi dataset

df\_selected = df.iloc[:, [1, 2, 7,8,10,14,17,30,36,38,39,42,49,56]]

df\_selected.head()

**2 3 8 9 11 15 18 31 37 39 40 43 50 57**

0 40.0 1.0 2.0 140.0 289.0 0.0 0.0 172.0 0.0 0.0 NaN NaN NaN 0.0

1 49.0 0.0 3.0 160.0 180.0 0.0 0.0 156.0 0.0 1.0 2.0 NaN NaN 1.0

2 37.0 1.0 2.0 130.0 283.0 0.0 1.0 98.0 0.0 0.0 NaN NaN NaN 0.0

3 48.0 0.0 4.0 138.0 214.0 0.0 0.0 108.0 1.0 1.5 2.0 NaN NaN 3.0

4 54.0 1.0 3.0 150.0 NaN 0.0 0.0 122.0 0.0 0.0 NaN NaN NaN 0.0

df\_selected.info()

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

RangeIndex: 294 entries, 0 to 293

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 2 294 non-null float64

1 3 294 non-null float64

2 8 294 non-null float64

3 9 293 non-null float64

4 11 271 non-null float64

5 15 286 non-null float64

6 18 293 non-null float64

7 31 293 non-null float64

8 37 293 non-null float64

9 39 294 non-null float64

10 40 104 non-null float64

11 43 4 non-null float64

12 50 28 non-null float64

13 57 294 non-null float64

dtypes: float64(14)

memory usage: 32.3 KB

mengganti nama kolom sesuai dengan 14 nama kolom yang ada pada deskripsi dataset

column\_mapping = {

2: 'age',

3: 'sex',

8: 'cp',

9: 'trestbps',

11: 'chol',

15: 'fbs',

18: 'restecg',

31: 'thalach',

37: 'exang',

39: 'oldpeak',

40: 'slope',

43: 'ca',

50: 'thal',

57: 'target'

}

df\_selected.rename(columns=column\_mapping, inplace=True)

<ipython-input-16-edcc9cd19c95>:18: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy df\_selected.rename(columns=column\_mapping, inplace=True)

df\_selected.info()

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

RangeIndex: 294 entries, 0 to 293

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 294 non-null float64

1 sex 294 non-null float64

2 cp 294 non-null float64

3 trestbps 293 non-null float64

4 chol 271 non-null float64

5 fbs 286 non-null float64

6 restecg 293 non-null float64

7 thalach 293 non-null float64

8 exang 293 non-null float64

9 oldpeak 294 non-null float64

10 slope 104 non-null float64

11 ca 4 non-null float64

12 thal 28 non-null float64

13 target 294 non-null float64

dtypes: float64(14)

memory usage: 32.3 KB

menghitung jumlah fitur pada dataset

df\_selected.value\_counts()

age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal target

47.0 1.0 4.0 150.0 226.0 0.0 0.0 98.0 1.0 1.5 2.0 0.0 7.0 1.0 1

dtype: int64

 5) Membersihkan Data

Sebelum melakukan pemodelan dilakukan pembersihan data agar model yang dihasilkan lebih akurat

menghitung jumlah null values yang ada diddalam dataset

df\_selected.isnull().sum()

age 0

sex 0

cp 0

trestbps 1

chol 23

fbs 8

restecg 1

thalach 1

exang 1

oldpeak 0

slope 190

ca 290

thal 266

target 0

dtype: int64

Berdasarkan output kode program diatas ada beberapa fitur yang hampir 90% datanya memiliki nilai null sehingga perlu dilakukan penghapusan fitur menggunakan fungsi drop

columns\_to\_drop = ['ca', 'slope','thal']

df\_selected = df\_selected.drop(columns\_to\_drop, axis=1)

df\_selected.isnull().sum()

age 0

sex 0

cp 0

trestbps 1

chol 23

fbs 8

restecg 1

thalach 1

exang 1

oldpeak 0

target 0

dtype: int64

Dikarenakan masih ada nilai null dibeberapa kolom fitur maka akan dilakukan pengisian nilai null menggunakan nilai mean di setiap kolomnya

meanTBPS = df\_selected['trestbps'].dropna()

meanChol = df\_selected['chol'].dropna()

meanfbs = df\_selected['fbs'].dropna()

meanRestCG = df\_selected['restecg'].dropna()

meanthalach = df\_selected['thalach'].dropna()

meanexang = df\_selected['exang'].dropna()

meanTBPS = meanTBPS.astype(float)

meanChol = meanChol.astype(float)

meanfbs = meanfbs.astype(float)

meanthalach = meanthalach.astype(float)

meanexang = meanexang.astype(float)

meanRestCG = meanRestCG.astype(float)

meanTBPS = round(meanTBPS.mean())

meanChol = round(meanChol.mean())

meanfbs = round(meanfbs.mean())

meanthalach = round(meanthalach.mean())

meanexang = round(meanexang.mean())

meanRestCG = round(meanRestCG.mean())

mengubah nilai null menjadi nilai mean yang sudah ditentukan sebelumnya

fill\_values = {'trestbps': meanTBPS, 'chol': meanChol, 'fbs': meanfbs,

'thalach':meanthalach,'exang':meanexang,'restecg':meanRestCG}

dfClean = df\_selected.fillna(value=fill\_values)

dfClean.info()

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

RangeIndex: 294 entries, 0 to 293

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 294 non-null float64

1 sex 294 non-null float64

2 cp 294 non-null float64

3 trestbps 294 non-null float64

4 chol 294 non-null float64

5 fbs 294 non-null float64

6 restecg 294 non-null float64

7 thalach 294 non-null float64

8 exang 294 non-null float64

9 oldpeak 294 non-null float64

10 target 294 non-null float64

dtypes: float64(11)

memory usage: 25.4 KB

dfClean.isnull().sum()

age 0

sex 0

cp 0

trestbps 0

chol 0

fbs 0

restecg 0

thalach 0

exang 0

oldpeak 0

target 0

dtype: int64

melalukan pengecekan terhadap duplikaksi data

duplicate\_rows = dfClean.duplicated()

dfClean[duplicate\_rows]

**age sex cp trestbps chol fbs restecg thalach exang oldpeak target**

163 49.0 0.0 2.0 110.0 251.0 0.0 0.0 160.0 0.0 0.0 0.0

print("All Duplicate Rows:")

dfClean[dfClean.duplicated(keep=False)]

All Duplicate Rows:

**age sex cp trestbps chol fbs restecg thalach exang oldpeak target**

90 49.0 0.0 2.0 110.0 251.0 0.0 0.0 160.0 0.0 0.0 0.0

163 49.0 0.0 2.0 110.0 251.0 0.0 0.0 160.0 0.0 0.0 0.0

Menghapus data yang memiliki duplikat

dfClean = dfClean.drop\_duplicates()

print("All Duplicate Rows:")

dfClean[dfClean.duplicated(keep=False)]

All Duplicate Rows:

**age sex cp trestbps chol fbs restecg thalach exang oldpeak target**

dfClean.head()

**age sex cp trestbps chol fbs restecg thalach exang oldpeak target**

0 40.0 1.0 2.0 140.0 289.0 0.0 0.0 172.0 0.0 0.0 0.0

1 49.0 0.0 3.0 160.0 180.0 0.0 0.0 156.0 0.0 1.0 1.0

2 37.0 1.0 2.0 130.0 283.0 0.0 1.0 98.0 0.0 0.0 0.0

3 48.0 0.0 4.0 138.0 214.0 0.0 0.0 108.0 1.0 1.5 3.0

4 54.0 1.0 3.0 150.0 251.0 0.0 0.0 122.0 0.0 0.0 0.0

dfClean['target'].value\_counts()

0.0 187

1.0 37

3.0 28

2.0 26

4.0 15

Name: target, dtype: int64

import seaborn as sns

import matplotlib.pyplot as plt

Mencari korelasi antar fitur

dfClean.corr()

**age sex cp trestbps chol fbs restecg thalach exang oldpeak target**

age 1.000000 0.014516 0.146616 0.246571 0.087101 0.181130 0.050672 -0.460514 0.239223 0.178172 0.210429 sex 0.014516 1.000000 0.245769 0.082064 0.027695 0.044372 -0.108656 -0.106959 0.154925 0.115959 0.220732 cp 0.146616 0.245769 1.000000 0.081293 0.134697 0.031930 -0.016372 -0.367819 0.494674 0.351735 0.427536

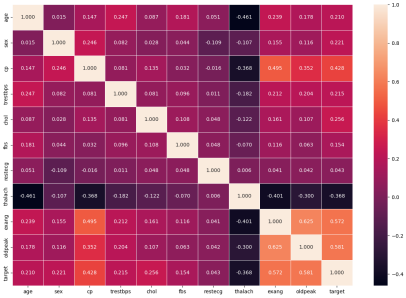
trestbps 0.246571 0.082064 0.081293 1.000000 0.080818 0.096222 0.011256 -0.181824 0.211507 0.204000 0.214898 chol 0.087101 0.027695 0.134697 0.080818 1.000000 0.107686 0.048081 -0.122038 0.161055 0.106743 0.256027 fbs 0.181130 0.044372 0.031930 0.096222 0.107686 1.000000 0.047988 -0.069722 0.115503 0.063179 0.154319 restecg 0.050672 -0.108656 -0.016372 0.011256 0.048081 0.047988 1.000000 0.006084 0.041290 0.042193 0.042643 thalach -0.460514 -0.106959 -0.367819 -0.181824 -0.122038 -0.069722 0.006084 1.000000 -0.400508 -0.300458 -0.367525 exang 0.239223 0.154925 0.494674 0.211507 0.161055 0.115503 0.041290 -0.400508 1.000000 0.624965 0.571710 oldpeak 0.178172 0.115959 0.351735 0.204000 0.106743 0.063179 0.042193 -0.300458 0.624965 1.000000 0.580732 target 0.210429 0.220732 0.427536 0.214898 0.256027 0.154319 0.042643 -0.367525 0.571710 0.580732 1.000000

cor\_mat=dfClean.corr()

fig,ax=plt.subplots(figsize=(15,10))

sns.heatmap(cor\_mat,annot=True,linewidths=0.5,fmt=".3f")

<Axes: >

 6) Konstruksi Data

Dalam tahap ini Konstruksi data salah satu tujuannya yaitu untuk menyesuaikan semua tipe data yang ada di dalam dataset. Namun pada tahap ini dataset sudah memiliki tipe data yang sesuai sehingga tidak perlu dilakukan penyesuaian kembali

dfClean.info()

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

Int64Index: 293 entries, 0 to 293

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 age 293 non-null float64

1 sex 293 non-null float64

2 cp 293 non-null float64

3 trestbps 293 non-null float64

4 chol 293 non-null float64

5 fbs 293 non-null float64

6 restecg 293 non-null float64

7 thalach 293 non-null float64

8 exang 293 non-null float64

9 oldpeak 293 non-null float64

10 target 293 non-null float64

dtypes: float64(11)

memory usage: 27.5 KB

dfClean.head(5)

**age sex cp trestbps chol fbs restecg thalach exang oldpeak target**

0 40.0 1.0 2.0 140.0 289.0 0.0 0.0 172.0 0.0 0.0 0.0

1 49.0 0.0 3.0 160.0 180.0 0.0 0.0 156.0 0.0 1.0 1.0

2 37.0 1.0 2.0 130.0 283.0 0.0 1.0 98.0 0.0 0.0 0.0

3 48.0 0.0 4.0 138.0 214.0 0.0 0.0 108.0 1.0 1.5 3.0

4 54.0 1.0 3.0 150.0 251.0 0.0 0.0 122.0 0.0 0.0 0.0

Setelah Menyesuaikan tipe dataset kita , kita harus memisahkan antara fitur dan target lalu simpan kedalam variabel.

X = dfClean.drop("target",axis=1).values

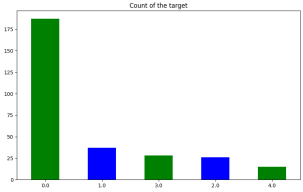
y = dfClean.iloc[:,-1]

Setelah kita memisahkan antara fitur dan target , sebaiknya kita melakukan pengecekan terlebih dahulu terhadap persebaran jumlah target terlebih dahulu.

dfClean['target'].value\_counts().plot(kind='bar',figsize=(10,6),color=['green','blue'])

plt.title("Count of the target")

plt.xticks(rotation=0);



Pada Grafik diatas menunjukan bahwa persebaran jumlah target tidak seimbang oleh karena itu perlu diseimbangkan terlebih dahulu. Menyeimbangkan target ada 2 cara yaitu oversampling dan undersampling. oversampling dilakukan jika jumlah dataset sedikit sedangkan undersampling dilakukan jika jumlah data terlalu banyak.

Disini kita akan melakukan oversampling dikarenakan jumlah data kita tidak banyak. Salah satu metode yang Oversampling yang akan kita gunakan adalah SMOTE

from imblearn.over\_sampling import SMOTE

# oversampling

smote = SMOTE(random\_state=42)

X\_smote\_resampled, y\_smote\_resampled = smote.fit\_resample(X, y)

plt.figure(figsize=(12, 4))

new\_df1 = pd.DataFrame(data=y)

plt.subplot(1, 2, 1)

new\_df1.value\_counts().plot(kind='bar',figsize=(10,6),color=['green','blue','red','yellow'])

plt.title("target before over sampling with SMOTE ")

plt.xticks(rotation=0);

plt.subplot(1, 2, 2)

new\_df2 = pd.DataFrame(data=y\_smote\_resampled)

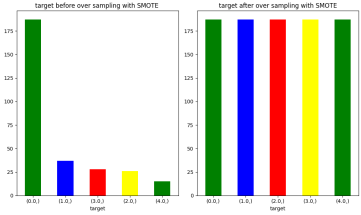
new\_df2.value\_counts().plot(kind='bar',figsize=(10,6),color=['green','blue','red','yellow'])

plt.title("target after over sampling with SMOTE")

plt.xticks(rotation=0);

plt.tight\_layout()

plt.show()

Pada Grafik diatas dapat dilihat ketika target belum di seimbangkan dan sudah diseimbangkan menggunakan oversampling.

new\_df1 = pd.DataFrame(data=y)

new\_df1.value\_counts()

target

0.0 187

1.0 37

3.0 28

2.0 26

4.0 15

dtype: int64

# over

new\_df2 = pd.DataFrame(data=y\_smote\_resampled)

new\_df2.value\_counts()

target

0.0 187

1.0 187

2.0 187

3.0 187

4.0 187

dtype: int64

Setelah menyeimbangkan persebaran jumlah target kita akan melakukan mengecekan apakah perlu dilakukan normalisasi/standarisasi pada datset kita.

dfClean.describe()

**age sex cp trestbps chol fbs restecg thalach exang oldpeak target** count 293.000000 293.000000 293.000000 293.000000 293.000000 293.000000 293.000000 293.000000 293.000000 293.000000 293.000000 mean 47.822526 0.726962 2.986348 132.662116 250.860068 0.068259 0.218430 139.058020 0.303754 0.588055 0.795222

std 7.824875 0.446282 0.965049 17.576793 65.059069 0.252622 0.460868 23.558003 0.460665 0.909554 1.238251 min 28.000000 0.000000 1.000000 92.000000 85.000000 0.000000 0.000000 82.000000 0.000000 0.000000 0.000000 25% 42.000000 0.000000 2.000000 120.000000 211.000000 0.000000 0.000000 122.000000 0.000000 0.000000 0.000000 50% 49.000000 1.000000 3.000000 130.000000 248.000000 0.000000 0.000000 140.000000 0.000000 0.000000 0.000000 75% 54.000000 1.000000 4.000000 140.000000 277.000000 0.000000 0.000000 155.000000 1.000000 1.000000 1.000000 max 66.000000 1.000000 4.000000 200.000000 603.000000 1.000000 2.000000 190.000000 1.000000 5.000000 4.000000

Pada deskripsi diatas dapat dilihat bahwa terdapat rentang nilai yang cukup jauh pada standar deviasi setiap fitur dataset yang kita miliki. Oleh karena itu perlu dilakukan normalisasi/standarisasi agar memperkecil rentang antara standar deviasi setiap kolom.

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_smote\_resampled\_normal = scaler.fit\_transform(X\_smote\_resampled)

len(X\_smote\_resampled\_normal)

935

dfcek1 = pd.DataFrame(X\_smote\_resampled\_normal)

dfcek1.describe()

**0 1 2 3 4 5 6 7 8 9** count 935.000000 935.000000 935.000000 935.000000 935.000000 935.000000 935.000000 935.000000 935.000000 935.000000 mean 0.563739 0.842507 0.818224 0.403413 0.341027 0.094277 0.117938 0.453354 0.598398 0.227015

std 0.174873 0.332492 0.274211 0.147493 0.110990 0.252030 0.199527 0.197232 0.450288 0.201293 min 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 25% 0.473283 1.000000 0.666667 0.305556 0.267954 0.000000 0.000000 0.312720 0.000000 0.000000 50% 0.578947 1.000000 1.000000 0.387952 0.330240 0.000000 0.000000 0.440606 0.962447 0.200000 75% 0.683363 1.000000 1.000000 0.487481 0.393811 0.000000 0.201473 0.593629 1.000000 0.386166 max 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000

Setelah dilakukan normalisasi pada fitur, selanjutnya kita perlu membagi fitur dan target menjadi data train dan test.

from sklearn.model\_selection import train\_test\_split

# membagi fitur dan target menjadi data train dan test (untuk yang oversample saja)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_smote\_resampled, y\_smote\_resampled, test\_size=0.2, random\_state=42,stratify=y\_smote\_resampled)

# membagi fitur dan target menjadi data train dan test (untuk yang oversample + normalization)

X\_train\_normal, X\_test\_normal, y\_train\_normal, y\_test\_normal = train\_test\_split(X\_smote\_resampled\_normal, y\_smote\_resampled, test\_size=0.2, random\_state=42,stratify = y\_smote\_resampled)  7) Model

Pada tahap ini kita akan memulai untuk membangun sebuah model.

Dibawah ini merupakan sebuah fungsi untuk menampilkan hasil akurasi dan rata - rata dari recall , f1 dan precision score setiap model. Fungsi

ini nantinya akan dipanggil di setiap model. Membuat Fungsi ini bersifat opsional.

from sklearn.metrics import accuracy\_score,recall\_score,f1\_score,precision\_score,roc\_auc\_score,confusion\_matrix,precision\_score

def evaluation(Y\_test,Y\_pred):

acc = accuracy\_score(Y\_test,Y\_pred)

rcl = recall\_score(Y\_test,Y\_pred,average = 'weighted')

f1 = f1\_score(Y\_test,Y\_pred,average = 'weighted')

ps = precision\_score(Y\_test,Y\_pred,average = 'weighted')

metric\_dict={'accuracy': round(acc,3),

'recall': round(rcl,3),

'F1 score': round(f1,3),

'Precision score': round(ps,3)

}

return print(metric\_dict)

 Oversample

 KNN

Pada tahap ini kita akan akan memulai membangun model dengan algoritma KNN dengan nilai neighbors yaitu 3.

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score, classification\_report

knn\_model = KNeighborsClassifier(n\_neighbors = 3)

knn\_model.fit(X\_train, y\_train)

▾ KNeighborsClassifier

KNeighborsClassifier(n\_neighbors=3)

Berikut adalah kode program untuk menampilkan hasil akurasi dengan algoritma KNN

y\_pred\_knn = knn\_model.predict(X\_test)

# Evaluate the KNN model

print("K-Nearest Neighbors (KNN) Model:")

accuracy\_knn\_smote = round(accuracy\_score(y\_test,y\_pred\_knn),3)

print("Accuracy:", accuracy\_knn\_smote)

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_knn))

K-Nearest Neighbors (KNN) Model:

Accuracy: 0.754

Classification Report:

precision recall f1-score support

0.0 0.65 0.39 0.49 38

1.0 0.73 0.81 0.77 37

2.0 0.80 0.86 0.83 37

3.0 0.77 0.87 0.81 38

4.0 0.78 0.84 0.81 37

accuracy 0.75 187

macro avg 0.75 0.76 0.74 187

weighted avg 0.74 0.75 0.74 187

evaluation(y\_test,y\_pred\_knn)

{'accuracy': 0.754, 'recall': 0.754, 'F1 score': 0.741, 'Precision score': 0.745}

Pada visualisasi ini ditampilkan visualisasi confusion matrix untuk membandingkan hasil prediksi model dengan nilai sebenarnya.

cm = confusion\_matrix(y\_test, y\_pred\_knn)

plt.figure(figsize=(8, 6))

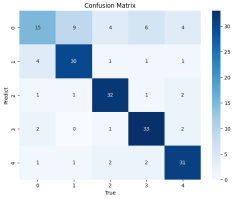
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()

 Random Forest 

Selanjutnya kita akan membangun model dengan algoritma random forest dengan n\_estimators yaitu 100, n\_estimators sendiri berguna mengatur jumlah pohon keputusan yang akan dibangun

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

▾ RandomForestClassifier

RandomForestClassifier(random\_state=42)

y\_pred\_rf = rf\_model.predict(X\_test)

# Evaluate the Random Forest model

print("\nRandom Forest Model:")

accuracy\_rf\_smote = round(accuracy\_score(y\_test, y\_pred\_rf),3)

print("Accuracy:",accuracy\_rf\_smote)

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_rf))

Random Forest Model:

Accuracy: 0.92

Classification Report:

precision recall f1-score support

0.0 0.94 0.89 0.92 38

1.0 0.85 0.92 0.88 37

2.0 0.89 0.89 0.89 37

3.0 0.95 0.97 0.96 38

4.0 0.97 0.92 0.94 37

accuracy 0.92 187

macro avg 0.92 0.92 0.92 187

weighted avg 0.92 0.92 0.92 187

evaluation(y\_test,y\_pred\_rf)

{'accuracy': 0.92, 'recall': 0.92, 'F1 score': 0.92, 'Precision score': 0.922}

cm = confusion\_matrix(y\_test, y\_pred\_rf)

plt.figure(figsize=(8, 6))

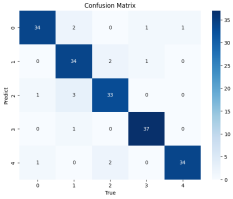
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()



 XGBoost

Pada tahap ini dalam membangun model, kita akan menggunakan algoritma XGBoost dengan learning rate yaitu 0.1. learning rate berguna untuk mengontrol seberapa besar kita menyesuaikan bobot model.

xgb\_model = XGBClassifier(learning\_rate=0.1, n\_estimators=100, random\_state=42)

xgb\_model.fit(X\_train, y\_train)

▾ XGBClassifier

XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=None, device=None, early\_stopping\_rounds=None,

enable\_categorical=False, eval\_metric=None, feature\_types=None,

gamma=None, grow\_policy=None, importance\_type=None,

interaction\_constraints=None, learning\_rate=0.1, max\_bin=None,

max\_cat\_threshold=None, max\_cat\_to\_onehot=None,

max\_delta\_step=None, max\_depth=None, max\_leaves=None,

min\_child\_weight=None, missing=nan, monotone\_constraints=None,

multi\_strategy=None, n\_estimators=100, n\_jobs=None,

num\_parallel\_tree=None, objective='multi:softprob', ...)

y\_pred\_xgb = xgb\_model.predict(X\_test)

# Evaluate the XGBoost model

print("\nXGBoost Model:")

accuracy\_xgb\_smote = round(accuracy\_score(y\_test, y\_pred\_xgb),3)

print("Accuracy:",accuracy\_xgb\_smote)

print("Classification Report:")

print(classification\_report(y\_test, y\_pred\_xgb))

XGBoost Model:

Accuracy: 0.904

Classification Report:

precision recall f1-score support

0.0 0.92 0.89 0.91 38

1.0 0.94 0.84 0.89 37

2.0 0.85 0.89 0.87 37

3.0 0.88 0.97 0.93 38

4.0 0.94 0.92 0.93 37

accuracy 0.90 187

macro avg 0.91 0.90 0.90 187

weighted avg 0.91 0.90 0.90 187

evaluation(y\_test,y\_pred\_xgb)

{'accuracy': 0.904, 'recall': 0.904, 'F1 score': 0.904, 'Precision score': 0.906}

cm = confusion\_matrix(y\_test, y\_pred\_xgb)

plt.figure(figsize=(8, 6))

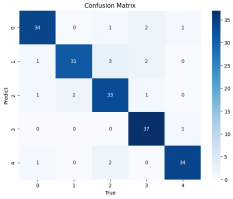
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()



 Oversample + Normalisasi

Pada bagian ini kita akan membuat sebuah model yang dimana data yang dipakai kali ini yang sudah dilakukan oversample dan normalisasi. Algoritma yang digunakan sama seperti sebelumnya yaitu KNN, Random Forest, dan XGBoost. Sekaligus dibuat visualisasi hasil evaluasi pada masing-masing model.

 KNN

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score, classification\_report

knn\_model = KNeighborsClassifier(n\_neighbors=3)

knn\_model.fit(X\_train\_normal, y\_train\_normal)

▾ KNeighborsClassifier

KNeighborsClassifier(n\_neighbors=3)

y\_pred\_knn = knn\_model.predict(X\_test\_normal)

# Evaluate the KNN model

print("K-Nearest Neighbors (KNN) Model:")

accuracy\_knn\_smote\_normal = round(accuracy\_score(y\_test\_normal,y\_pred\_knn),3)

print("Accuracy:", accuracy\_knn\_smote\_normal)

print("Classification Report:")

print(classification\_report(y\_test\_normal, y\_pred\_knn))

K-Nearest Neighbors (KNN) Model:

Accuracy: 0.861

Classification Report:

precision recall f1-score support

0.0 0.88 0.76 0.82 38

1.0 0.78 0.84 0.81 37

2.0 0.87 0.92 0.89 37

3.0 0.92 0.87 0.89 38

4.0 0.87 0.92 0.89 37

accuracy 0.86 187

macro avg 0.86 0.86 0.86 187

weighted avg 0.86 0.86 0.86 187

evaluation(y\_test\_normal,y\_pred\_knn)

{'accuracy': 0.861, 'recall': 0.861, 'F1 score': 0.861, 'Precision score': 0.863}

cm = confusion\_matrix(y\_test\_normal, y\_pred\_knn)

plt.figure(figsize=(8, 6))

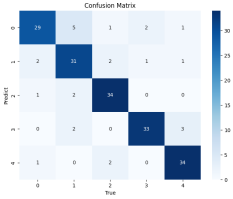
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()

 Random Forest

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42) rf\_model.fit(X\_train\_normal, y\_train\_normal)

▾ RandomForestClassifier

RandomForestClassifier(random\_state=42)

y\_pred\_rf = rf\_model.predict(X\_test\_normal)

# Evaluate the Random Forest model

print("\nRandom Forest Model:")

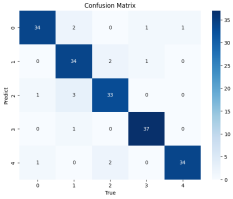
accuracy\_rf\_smote\_normal = round(accuracy\_score(y\_test\_normal, y\_pred\_rf),3) print("Accuracy:",accuracy\_rf\_smote\_normal )

print("Classification Report:")

print(classification\_report(y\_test\_normal, y\_pred\_rf))

Random Forest Model:

Accuracy: 0.92

Classification Report: 

precision recall f1-score support

0.0 0.94 0.89 0.92 38

1.0 0.85 0.92 0.88 37

2.0 0.89 0.89 0.89 37

3.0 0.95 0.97 0.96 38

4.0 0.97 0.92 0.94 37

accuracy 0.92 187

macro avg 0.92 0.92 0.92 187

weighted avg 0.92 0.92 0.92 187

evaluation(y\_test\_normal,y\_pred\_rf)

{'accuracy': 0.92, 'recall': 0.92, 'F1 score': 0.92, 'Precision score': 0.922}

cm = confusion\_matrix(y\_test\_normal, y\_pred\_rf)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()

 XGBOOST

xgb\_model = XGBClassifier(learning\_rate=0.1, n\_estimators=100, random\_state=42) xgb\_model.fit(X\_train\_normal, y\_train\_normal)

▾ XGBClassifier

XGBClassifier(base\_score=None, booster=None, callbacks=None,

colsample\_bylevel=None, colsample\_bynode=None,

colsample\_bytree=None, device=None, early\_stopping\_rounds=None, enable\_categorical=False, eval\_metric=None, feature\_types=None, gamma=None, grow\_policy=None, importance\_type=None, interaction\_constraints=None, learning\_rate=0.1, max\_bin=None, max\_cat\_threshold=None, max\_cat\_to\_onehot=None,

max\_delta\_step=None, max\_depth=None, max\_leaves=None, min\_child\_weight=None, missing=nan, monotone\_constraints=None, multi\_strategy=None, n\_estimators=100, n\_jobs=None, num\_parallel\_tree=None, objective='multi:softprob', ...)

y\_pred\_xgb = xgb\_model.predict(X\_test\_normal)

# Evaluate the XGBoost model

print("\nXGBoost Model:")

accuracy\_xgb\_smote\_normal = round(accuracy\_score(y\_test\_normal, y\_pred\_xgb),3) print("Accuracy:",accuracy\_xgb\_smote\_normal)

print("Classification Report:")

print(classification\_report(y\_test\_normal, y\_pred\_xgb))

XGBoost Model:

Accuracy: 0.904

Classification Report:

precision recall f1-score support

0.0 0.92 0.89 0.91 38

1.0 0.94 0.84 0.89 37

2.0 0.85 0.89 0.87 37

3.0 0.88 0.97 0.93 38

4.0 0.94 0.92 0.93 37

accuracy 0.90 187

macro avg 0.91 0.90 0.90 187

weighted avg 0.91 0.90 0.90 187

evaluation(y\_test\_normal,y\_pred\_xgb)

{'accuracy': 0.904, 'recall': 0.904, 'F1 score': 0.904, 'Precision score': 0.906}

cm = confusion\_matrix(y\_test\_normal, y\_pred\_xgb)

plt.figure(figsize=(8, 6))

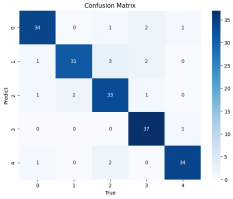
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()

 Tunning + Normalization + Oversample

Pada pembuatan model kali ini masih menggunakan algoritma yang sama (KNN, Random Forest, dan XGBoost), namun data yang digunakan adalah data yang sudah dilakukan TunNIng Parameter, Normalisasi, dan Oversample.

 KNN

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.model\_selection import RandomizedSearchCV

Setiap parameter tunnning tidak selalu sama karena bergantung pada algoritma yang digunakan.

knn\_model = KNeighborsClassifier()

param\_grid = {

"n\_neighbors": range(3, 21),

"metric": ["euclidean", "manhattan", "chebyshev"],

"weights": ["uniform", "distance"],

"algorithm": ["auto", "ball\_tree", "kd\_tree"],

"leaf\_size": range(10, 61),

}

knn\_model = RandomizedSearchCV(estimator=knn\_model, param\_distributions=param\_grid, n\_iter=100, scoring="accuracy", cv=5)

knn\_model.fit(X\_train\_normal, y\_train\_normal)

best\_params = knn\_model.best\_params\_

print(f"Best parameters: {best\_params}")

Best parameters: {'weights': 'distance', 'n\_neighbors': 4, 'metric': 'manhattan', 'leaf\_size': 30, 'algorithm': 'ball\_tree'}

y\_pred\_knn = knn\_model.predict(X\_test\_normal)

# Evaluate the KNN model

print("K-Nearest Neighbors (KNN) Model:")

accuracy\_knn\_smote\_normal\_Tun = round(accuracy\_score(y\_test\_normal,y\_pred\_knn),3)

print("Accuracy:", accuracy\_knn\_smote\_normal\_Tun)

print("Classification Report:")

print(classification\_report(y\_test\_normal, y\_pred\_knn))

K-Nearest Neighbors (KNN) Model:

Accuracy: 0.93

Classification Report:

precision recall f1-score support

0.0 0.94 0.89 0.92 38

1.0 0.86 0.86 0.86 37

2.0 0.92 0.92 0.92 37

3.0 0.97 0.97 0.97 38

4.0 0.95 1.00 0.97 37

accuracy 0.93 187

macro avg 0.93 0.93 0.93 187

weighted avg 0.93 0.93 0.93 187

evaluation(y\_test\_normal,y\_pred\_knn)

{'accuracy': 0.93, 'recall': 0.93, 'F1 score': 0.93, 'Precision score': 0.93}

cm = confusion\_matrix(y\_test\_normal, y\_pred\_knn)

plt.figure(figsize=(8, 6))

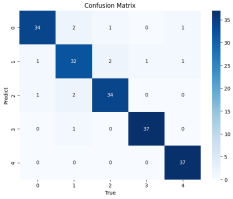
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()



 RandomForest

rf\_model = RandomForestClassifier()

param\_grid = {

"n\_estimators": [100, 200],

"max\_depth": [ 10, 15],

"min\_samples\_leaf": [1, 2],

"min\_samples\_split": [2, 5],

"max\_features": ["sqrt", "log2"],

# "random\_state": [42, 100, 200]

}

rf\_model = RandomizedSearchCV(rf\_model, param\_grid, n\_iter=100, cv=5, n\_jobs=-1)

rf\_model.fit(X\_train\_normal, y\_train\_normal)

best\_params = rf\_model.best\_params\_

print(f"Best parameters: {best\_params}")

/usr/local/lib/python3.10/dist-packages/sklearn/model\_selection/\_search.py:305: UserWarning: The total space of parameters 32 is smaller than n\_iter=100. Running 32 iterations. For exhaustive searches, u warnings.warn(

Best parameters: {'n\_estimators': 100, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_features': 'sqrt', 'max\_depth': 15}

y\_pred\_rf = rf\_model.predict(X\_test\_normal)

# Evaluate the Random Forest model

print("\nRandom Forest Model:")

accuracy\_rf\_smote\_normal\_Tun = round(accuracy\_score(y\_test\_normal, y\_pred\_rf),3)

print("Accuracy:",accuracy\_rf\_smote\_normal\_Tun)

print("Classification Report:")

print(classification\_report(y\_test\_normal, y\_pred\_rf))

Random Forest Model:

Accuracy: 0.904

Classification Report:

precision recall f1-score support

0.0 0.94 0.89 0.92 38

1.0 0.85 0.89 0.87 37

2.0 0.86 0.86 0.86 37

3.0 0.90 0.95 0.92 38

4.0 0.97 0.92 0.94 37

accuracy 0.90 187

macro avg 0.91 0.90 0.90 187

weighted avg 0.91 0.90 0.90 187

evaluation(y\_test\_normal,y\_pred\_rf)

{'accuracy': 0.904, 'recall': 0.904, 'F1 score': 0.904, 'Precision score': 0.906}

cm = confusion\_matrix(y\_test\_normal, y\_pred\_knn)

plt.figure(figsize=(8, 6))

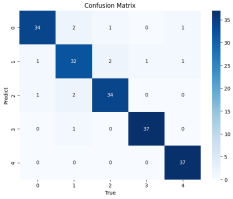
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()



 XGBOOST

xgb\_model = XGBClassifier()

param\_grid = {

"max\_depth": [3, 5, 7],

"learning\_rate": [0.01, 0.1],

"n\_estimators": [100, 200],

"gamma": [0, 0.1],

"colsample\_bytree": [0.7, 0.8],

}

xgb\_model = RandomizedSearchCV(xgb\_model, param\_grid, n\_iter=10, cv=5, n\_jobs=-1)

xgb\_model.fit(X\_train\_normal, y\_train\_normal)

best\_params = xgb\_model.best\_params\_

print(f"Best parameters: {best\_params}")

Best parameters: {'n\_estimators': 100, 'max\_depth': 7, 'learning\_rate': 0.1, 'gamma': 0, 'colsample\_bytree': 0.7}

y\_pred\_xgb = xgb\_model.predict(X\_test\_normal)

# Evaluate the XGBoost model

print("\nXGBoost Model:")

accuracy\_xgb\_smote\_normal\_Tun = round(accuracy\_score(y\_test\_normal, y\_pred\_xgb),3)

print("Accuracy:",accuracy\_xgb\_smote\_normal\_Tun)

print("Classification Report:")

print(classification\_report(y\_test\_normal, y\_pred\_xgb))

XGBoost Model:

Accuracy: 0.92

Classification Report:

precision recall f1-score support

0.0 0.90 0.95 0.92 38

1.0 0.91 0.86 0.89 37

2.0 0.89 0.86 0.88 37

3.0 0.93 1.00 0.96 38

4.0 0.97 0.92 0.94 37

accuracy 0.92 187

macro avg 0.92 0.92 0.92 187

weighted avg 0.92 0.92 0.92 187

evaluation(y\_test\_normal,y\_pred\_xgb)

{'accuracy': 0.92, 'recall': 0.92, 'F1 score': 0.919, 'Precision score': 0.92}

cm = confusion\_matrix(y\_test\_normal, y\_pred\_xgb)

plt.figure(figsize=(8, 6))

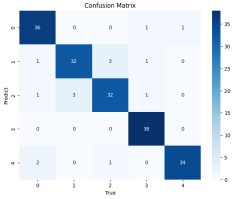
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

plt.title('Confusion Matrix')

plt.xlabel('True')

plt.ylabel('Predict')

plt.show()



 8) Evaluasi

Selanjutnya kita akan melakukan evaluasi data sekaligus membandingkan antar algoritma guna dengan tujuan mengetahui jenis model algoritma yang menghasilkan hasil akurasi terbaik.

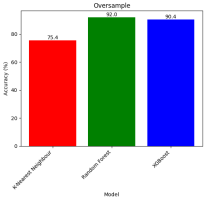
import matplotlib.pyplot as plt

model\_comp1 = pd.DataFrame({'Model': ['K-Nearest Neighbour','Random Forest',

'XGBoost'], 'Accuracy': [accuracy\_knn\_smote\*100,

accuracy\_rf\_smote\*100,accuracy\_xgb\_smote\*100]})

model\_comp1.head()

**Model Accuracy** 

0 K-Nearest Neighbour 75.4

1 Random Forest 92.0

2 XGBoost 90.4

# Membuat bar plot dengan keterangan jumlah

fig, ax = plt.subplots()

bars = plt.bar(model\_comp1['Model'], model\_comp1['Accuracy'], color=['red', 'green', 'blue']) plt.xlabel('Model')

plt.ylabel('Accuracy (%)')

plt.title('Oversample')

plt.xticks(rotation=45, ha='right') # Untuk memutar label sumbu x agar lebih mudah dibaca

# Menambahkan keterangan jumlah di atas setiap bar

for bar in bars:

yval = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2, yval, round(yval, 2), ha='center', va='bottom')

plt.show()

model\_comp2 = pd.DataFrame({'Model': ['K-Nearest Neighbour','Random Forest', 'XGBoost'], 'Accuracy': [accuracy\_knn\_smote\_normal\*100,

accuracy\_rf\_smote\_normal\*100,accuracy\_xgb\_smote\_normal\*100]})

model\_comp2.head()

**Model Accuracy**

0 K-Nearest Neighbour 86.1

1 Random Forest 92.0

2 XGBoost 90.4

# Membuat bar plot dengan keterangan jumlah

fig, ax = plt.subplots()

bars = plt.bar(model\_comp2['Model'], model\_comp2['Accuracy'], color=['red', 'green', 'blue']) plt.xlabel('Model')

plt.ylabel('Accuracy (%)')

plt.title('Normalization + Oversampling')

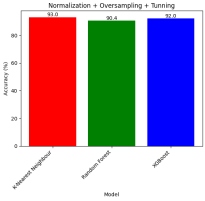
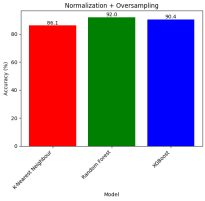
plt.xticks(rotation=45, ha='right') # Untuk memutar label sumbu x agar lebih mudah dibaca

# Menambahkan keterangan jumlah di atas setiap bar

for bar in bars:

yval = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2, yval, round(yval, 2), ha='center', va='bottom') plt.show()



model\_comp3 = pd.DataFrame({'Model': ['K-Nearest Neighbour','Random Forest', 'XGBoost'], 'Accuracy': [accuracy\_knn\_smote\_normal\_Tun\*100,

accuracy\_rf\_smote\_normal\_Tun\*100,accuracy\_xgb\_smote\_normal\_Tun\*100]})

model\_comp3.head()

**Model Accuracy**

0 K-Nearest Neighbour 93.0

1 Random Forest 90.4

2 XGBoost 92.0

# Membuat bar plot dengan keterangan jumlah

fig, ax = plt.subplots()

bars = plt.bar(model\_comp3['Model'], model\_comp3['Accuracy'], color=['red', 'green', 'blue']) plt.xlabel('Model')

plt.ylabel('Accuracy (%)')

plt.title('Normalization + Oversampling + Tunning')

plt.xticks(rotation=45, ha='right') # Untuk memutar label sumbu x agar lebih mudah dibaca

# Menambahkan keterangan jumlah di atas setiap bar

for bar in bars:

yval = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2, yval, round(yval, 2), ha='center', va='bottom') plt.show()

# Data frame

model\_compBest = pd.DataFrame({

'Model': ['K-Nearest Neighbour OverSample Tunning', 'Random Forest OverSample',

'XGB OverSample Standarization Tunning'],

'Accuracy': [accuracy\_knn\_smote\_normal\_Tun\*100, accuracy\_rf\_smote\_normal\*100,

accuracy\_xgb\_smote\_normal\_Tun\*100]

})

# Membuat bar plot dengan keterangan jumlah

fig, ax = plt.subplots()

bars = plt.bar(model\_compBest['Model'], model\_compBest['Accuracy'], color=['red', 'green', 'blue']) plt.xlabel('Model')

plt.ylabel('Accuracy (%)')

plt.title('Best Model Comparison')

plt.xticks(rotation=45, ha='right') # Untuk memutar label sumbu x agar lebih mudah dibaca

# Menambahkan keterangan jumlah di atas setiap bar

for bar in bars:

yval = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2, yval, round(yval, 2), ha='center', va='bottom')

plt.show()



 9) Streamlit

 10) Kesimpulan

Dari penelitian diatas setelah melakukan pemodelan dengan algoritma KNN, Random Forest, dan XGBoost dengan berbagai penanganan data antara lain menggunakan random over sampling SMOTE untuk penanganan imbalance data, RandomSearchCV untuk tunning, dan Normalisasi data. Dapat disimpulkan bahwa klasifikasi menggunakan Random Over Sampling SMOTE pada model KNN menghasilkan akurasi 75.4 %, model Random Forest dengan akurasi yang dihasilkan yaitu 92%, dan model XGBoots menghasilkan akurasi 90.4%. Disamping itu bila klasifikasi menggunakan data yang sudah dilakukan normalisasi dan Random Over Sampling SMOTE pada model KNN menghasilkan akurasi 86.1%, model Random Forest menghasilkan akurasi 92%, dan model XGBoots menghasilkan akurasi 90.4%. Dan pada klasifikasi menggunakan data yang telah dilakukan tunning RandomSearchCV, normalisasi, dan Random Over Sampling SMOTE dalam model KNN menghasilkan akurasi 93%, pada model Random Forest menghasilkan akurasi 87.7%. dan model XGBoots menghasilkan akurasi 92%. Oleh karena itu, dalam penanganan data yang optimal untuk mengatasi ketidakseimbangan data adalah dengan menggunakan metode random Oversampling SMOTE sekaligus yang dilengkapi dengan tuning menggunakan RandomSearchCV dan normalisasi data, memberikan hasil yang signifikan dalam meningkatkan akurasi model klasifikasi khususnya pada model KNN dan XGBoots, namun hal itu tidak terjadi pada model Random Forest yang

mengalami penurunan akurasi yang signifikan. Secara keseluruhan, penanganan dalam ketidakseimbangan data dengan menggunakan tunning parameter, normalisasi, dan oversampling dapat memberikan dampak signifikan terhadap performa model klasifikasi. Pemilihan model terbaik dan parameter optimal dapat meningkatkan akurasi dan kinerja model secara keseluruhan.

Unsupported Cell Type. Double-Click to inspect/edit the content.