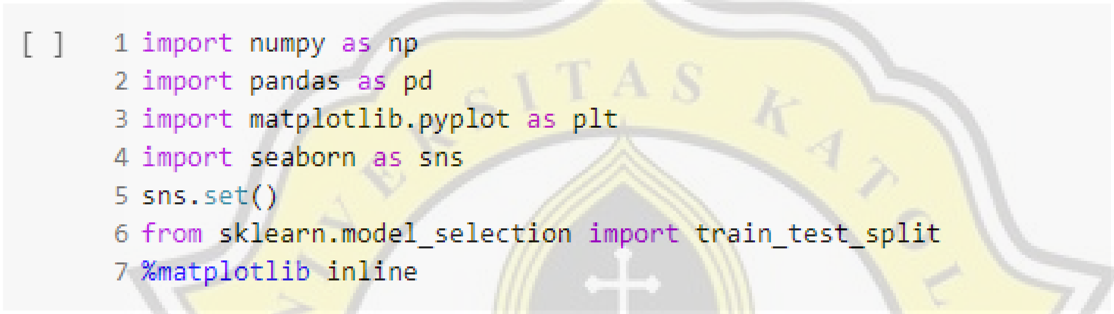
CHAPTER 5

IMPLEMENTATION AND RESULTS

5.1. Implementation

Chapter 5 explain the implementation and testing of projects about Diabetes Prediction using Decision Tree and XGBoost algorithm. Below is the code to implement Diabetes Prediction using Decision Tree and XGBoost algorithm

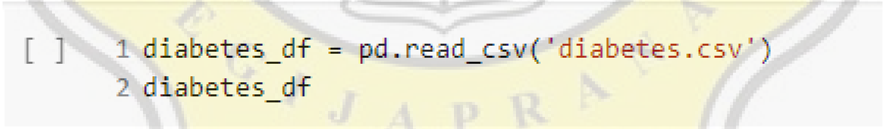


```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.model_selection import train_test_split
%matplotlib inline
```

Figure 5.1 Import some library

Importing some libraries :

1. Numpy : Numpy is the fundamental package for scientific computing
2. Pandas : Pandas is used for data analysis and manipulation tool
3. Matplotlib : Matplotlib is used for visualize the data
4. Seaborn : Seaborn is used for visualize the data
5. Sklearn train test split : train test split is used to split the train and test data



```
diabetes_df = pd.read_csv('diabetes.csv')
diabetes_df
```

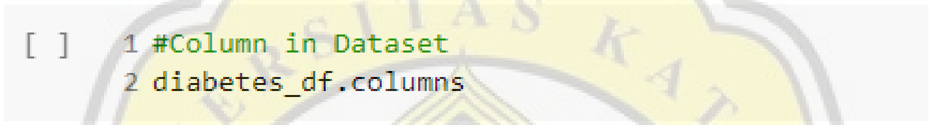
Figure 5.2 Import the dataset

Importing Pima Indian Diabetes Dataset (PIDD) using pandas. The raw dataset consist of 768 rows and 9 columns. This dataset was collected from Kaggle. The attributes that will be used in this project are : Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age, and Outcome. A detailed description of all attributes is given in here :

- Pregnancies : Number of times a woman got pregnant
- Glucose : Glucose concentration in oral glucose tolerance test for 120 min

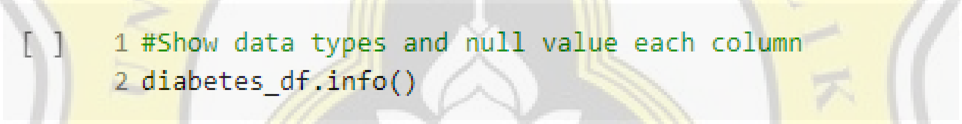
- Blood Pressure : Diastolic Blood Pressure
- Skin Thickness : Fold Thickness of Skin
- Insulin : Serum Insulin for 2 h
- BMI : Body Mass Index (weight/(height)^2)
- Diabetes Pedigree Function : Diabetes pedigree Function
- Age : Age (years)
- Outcome : Class variable (class value 1 for positive 0 for Negative for diabetes)

Exploratory Data Analysis



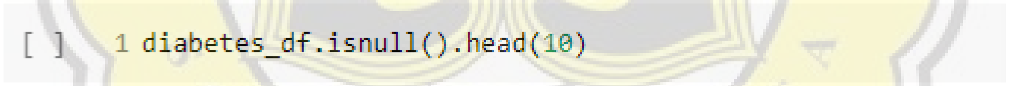
```
[ ]  #Column in Dataset
     diabetes_df.columns
```

Figure 5.3 Show all column in Dataset



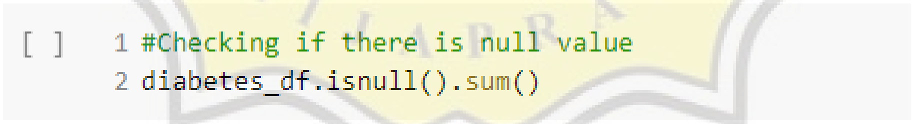
```
[ ]  #Show data types and null value each column
     diabetes_df.info()
```

Figure 5.4 Show dataset info



```
[ ]  diabetes_df.isnull().head(10)
```

Figure 5.5 Show dataset first 10 column



```
[ ]  #Checking if there is null value
     diabetes_df.isnull().sum()
```

Figure 5.6 Show the sum of null data before changing the zero value

Code Explanation :

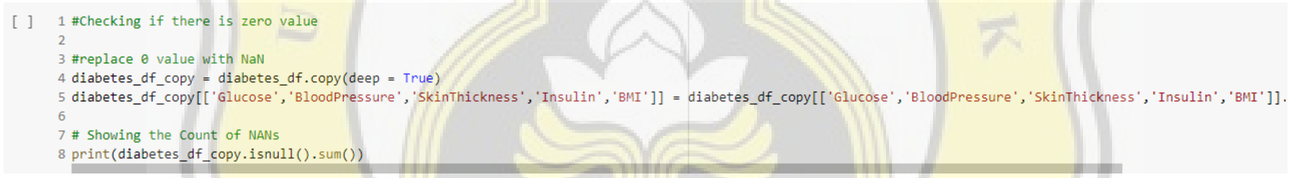
- diabetes_df.columns : Show all of the columns in the dataset
- diabetes_df.info() : Show the dataset info such as : the number of non-null data and data types

- diabetes_df.isnull().head(10) : Show the first 10 rows in the dataset to check if there is null data. False means non null data and True means null data
- diabetes_df.isnull().sum() : Show the sum of null data in each column before changing the zero value into null

Data Preprocessing

Data Preprocessing helps transform data so that a better machine learning model can be built, providing higher accuracy. The data preprocessing performs various functions: outlier detection and removal, filling missing values, and Oversampling minority data. In the dataset, 268 samples are classified as diabetic, and 500 were non-diabetics.

We have to check the null value in the dataset. There are no null value in the dataset but there are 5 attributes that impossible to have 0 value. They are : glucose, blood pressure, skin thickness, insulin, and BMI. All the zero values were replaced with the median value of that attribute to prevent inconsistent value. [1]



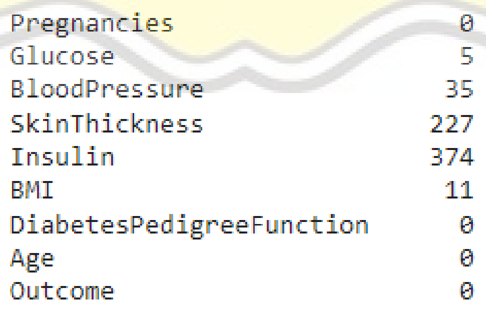
```
[ ]  #Checking if there is zero value

     #replace 0 value with NaN
     diabetes_df_copy = diabetes_df.copy(deep = True)
     diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetes_df_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].

     # Showing the Count of NANs
     print(diabetes_df_copy.isnull().sum())
```

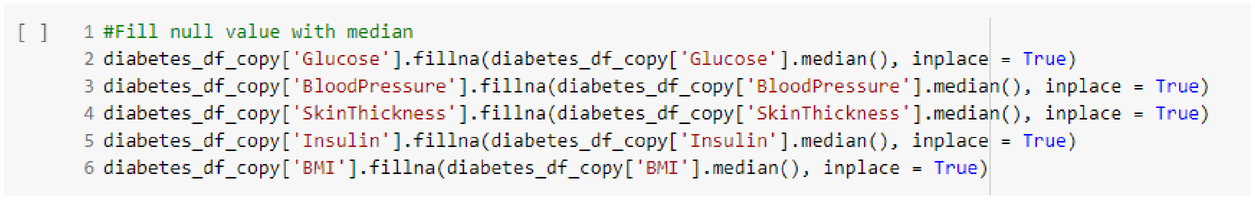
Figure 5.7 Changing zero value into null value

We have to change the value on Glucose, Blood Pressure, Skin Thickness, Insulin, and BMI because its impossible to have 0 value on these column. The function of deep = True parameter is to make a copy of the index and data in the data frame



```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
```

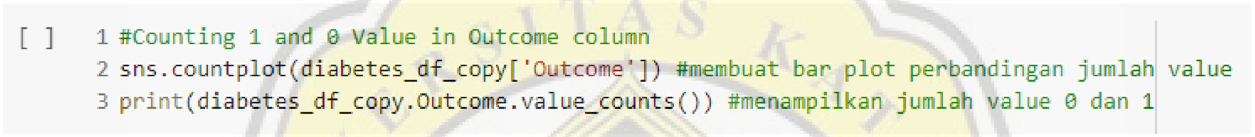
Figure 5.8 null value from 5 attributes mentioned above



```
[ ]  #Fill null value with median
     diabetes_df_copy['Glucose'].fillna(diabetes_df_copy['Glucose'].median(), inplace = True)
     diabetes_df_copy['BloodPressure'].fillna(diabetes_df_copy['BloodPressure'].median(), inplace = True)
     diabetes_df_copy['SkinThickness'].fillna(diabetes_df_copy['SkinThickness'].median(), inplace = True)
     diabetes_df_copy['Insulin'].fillna(diabetes_df_copy['Insulin'].median(), inplace = True)
     diabetes_df_copy['BMI'].fillna(diabetes_df_copy['BMI'].median(), inplace = True)
```

Figure 5.9 Fill null value with median

In this part, we have to change all of the null values with median. The column to be filled with median score are : Glucose, Blood Pressure, Skin Thickness, Insulin, and BMI. inplace = True parameter is used to save the modification from data frame diabetes_df_copy. the modification is filling the null value with the median value

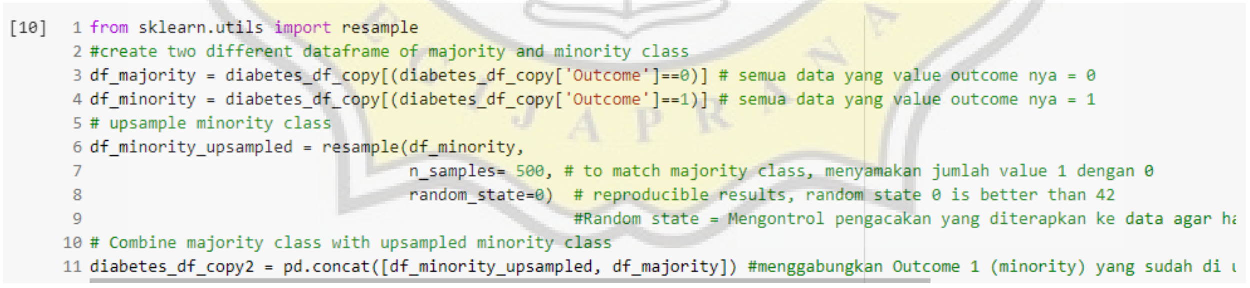


```
[ ]  #Counting 1 and 0 Value in Outcome column
     sns.countplot(diabetes_df_copy['Outcome']) #membuat bar plot perbandingan jumlah value
     print(diabetes_df_copy.Outcome.value_counts()) #menampilkan jumlah value 0 dan 1
```

Figure 5.10 Checking on unbalanced data

Visualize the 1 and 0 values on Outcome column to check if the data is balanced or not and print the 1 and 0 value counts

The class (Outcome) imbalance problem is solved by oversampling the minority class using synthetic minority oversampling technique (SMOTE). There are 268 samples classified as diabetic and 500 samples classified as non-diabetic. Now, we have to balance the class into 500 samples classified as diabetic and 500 classified as non-diabetic to prevent inconsistent result.



```
[10]  1 from sklearn.utils import resample
      2 #create two different dataframe of majority and minority class
      3 df_majority = diabetes_df_copy[(diabetes_df_copy['Outcome']==0)] # semua data yang value outcome nya = 0
      4 df_minority = diabetes_df_copy[(diabetes_df_copy['Outcome']==1)] # semua data yang value outcome nya = 1
      5 # upsample minority class
      6 df_minority_upsampled = resample(df_minority,
      7                                  n_samples= 500, # to match majority class, menyamakan jumlah value 1 dengan 0
      8                                  random_state=0)  # reproducible results, random state 0 is better than 42
      9                                                   #Random state = Mengontrol pengacakan yang diterapkan ke data agar ha
     10 # Combine majority class with upsampled minority class
     11 diabetes_df_copy2 = pd.concat([df_minority_upsampled, df_majority]) #menggabungkan Outcome 1 (minority) yang sudah di ι
```

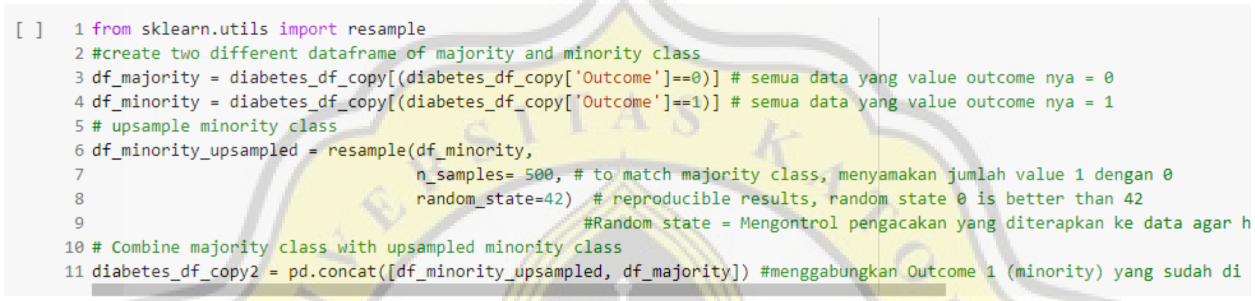
Figure 5.11 Oversampling method

Oversampling the minority class (Value 1) by equating the number of values with the majority class (Value 0) using Synthetic Minority Oversampling Technique (SMOTE).

- Line 1 : importing resample from sklearn
- Line 3 : create majority data frame (all of the rows where the Outcome value is 0)

- Line 4 : create minority data frame (all of the rows where the Outcome value is 1)
- Line 6-8 : Oversampling technique by calling resample library. n_samples = 500 is used to generate the minority value until 500 data, random_state = 0 is used to determines random number generation for shuffling the data
- Line 11 : Combining the majority dataframe which is all of outcome variable with value is 0 and upsampled minority dataframe after processing in line 6-9

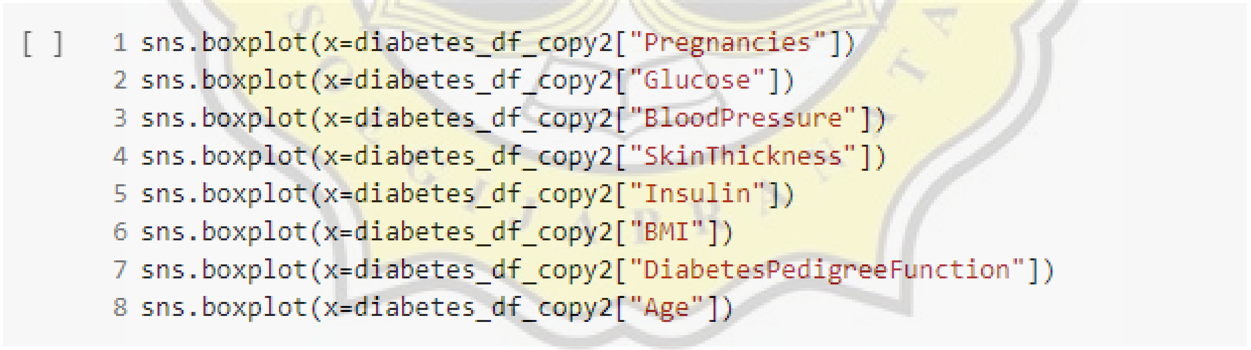
Random state = 42 in Oversampling part



```
[ ]   1 from sklearn.utils import resample
      2 #create two different dataframe of majority and minority class
      3 df_majority = diabetes_df_copy[(diabetes_df_copy['Outcome']==0)] # semua data yang value outcome nya = 0
      4 df_minority = diabetes_df_copy[(diabetes_df_copy['Outcome']==1)] # semua data yang value outcome nya = 1
      5 # upsample minority class
      6 df_minority_upsampled = resample(df_minority,
      7                                  n_samples= 500, # to match majority class, menyamakan jumlah value 1 dengan 0
      8                                  random_state=42)  # reproducible results, random state 0 is better than 42
      9                                                   #Random state = Mengontrol pengacakan yang diterapkan ke data agar h
     10 # Combine majority class with upsampled minority class
     11 diabetes_df_copy2 = pd.concat([df_minority_upsampled, df_majority]) #menggabungkan Outcome 1 (minority) yang sudah di
```

Figure 5.12 Using Random state = 42

There are outliers in Pregnancies, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function and Age. We check the outliers using a box plot for each attributes.



```
[ ]   1 sns.boxplot(x=diabetes_df_copy2["Pregnancies"])
      2 sns.boxplot(x=diabetes_df_copy2["Glucose"])
      3 sns.boxplot(x=diabetes_df_copy2["BloodPressure"])
      4 sns.boxplot(x=diabetes_df_copy2["SkinThickness"])
      5 sns.boxplot(x=diabetes_df_copy2["Insulin"])
      6 sns.boxplot(x=diabetes_df_copy2["BMI"])
      7 sns.boxplot(x=diabetes_df_copy2["DiabetesPedigreeFunction"])
      8 sns.boxplot(x=diabetes_df_copy2["Age"])
```

Figure 5.13 Showing Outlier in each attributes using boxplot

From line 1-8 is used to showing boxlot in each attributes from Pregnancies to Age.

- x=diabetes_df_copy2 : used to define the dataset
- [“Pregnancies - Age”] : used to define the column to show the boxplot

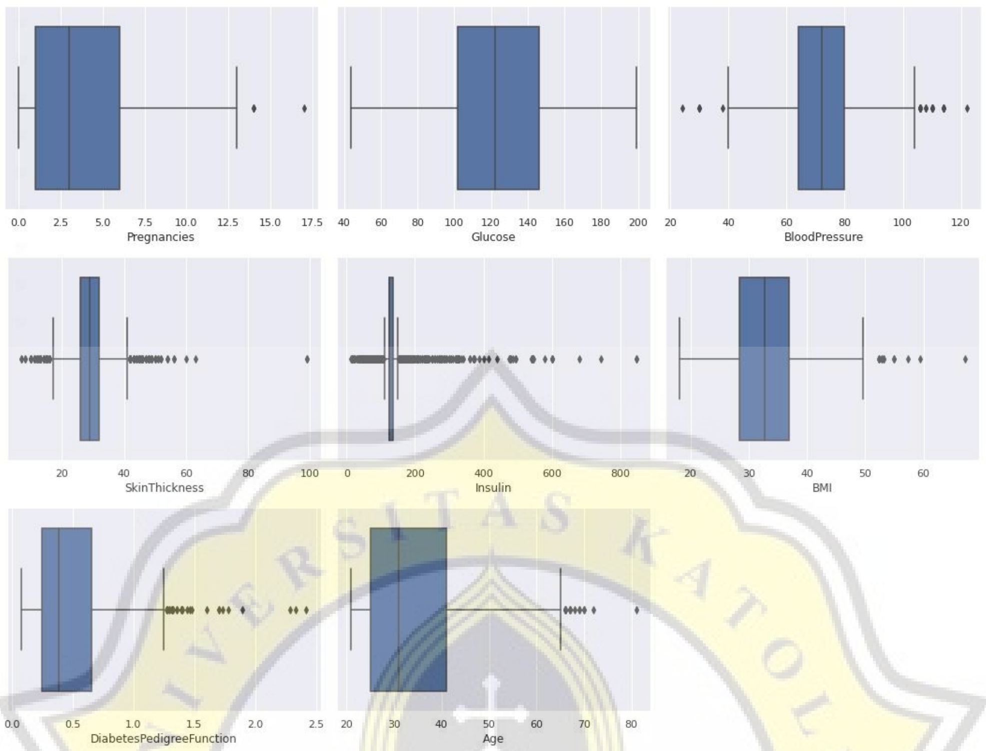
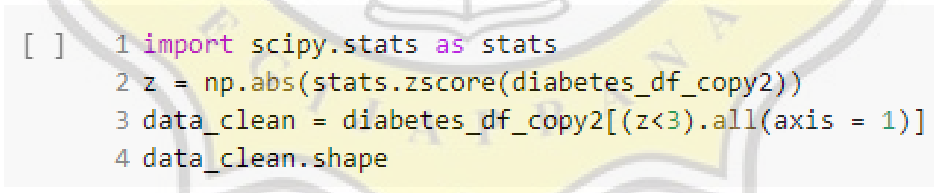


Figure 5.14 Outlier Detection using boxplot

We have to remove those outliers using Z-Score because Z-Score can give better outlier detection than other methods. After some rows containing outliers are removed, the number of rows which was originally 1000 becomes only 941 rows. the applicable Z-Score threshold is between -3 and 3 for 2 diagnostic categories (True or False). [15]



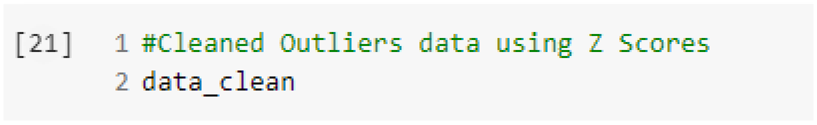
```
[ ]   1 import scipy.stats as stats
      2 z = np.abs(stats.zscore(diabetes_df_copy2))
      3 data_clean = diabetes_df_copy2[(z<3).all(axis = 1)]
      4 data_clean.shape
```

Figure 5.15 Applying Z-Score to remove the rows with outlier

The function of this part is to remove outlier using Z-Score. the range of Z-Score values between 3 to - 3. Here are the explanation of the code :

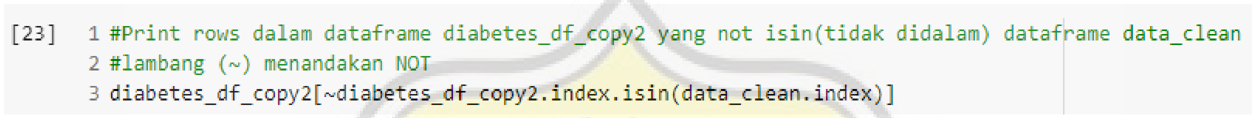
- Line 1 : importing scipy to use stats
- Line 2 : applying z-score into dataframe and changing negative Z-Score value into positive using np.absolute

- Line 3 : creating a new clean dataframe by printing all of the old dataframe where the z value less than 3, applied in all of column (axis = 1)
- Line 4 : Print the shape of the new dataframe



```
[21]  #Cleaned Outliers data using Z Scores
      data_clean
```

Figure 5.16 Showing the new dataframe after cleaned with Z-Score



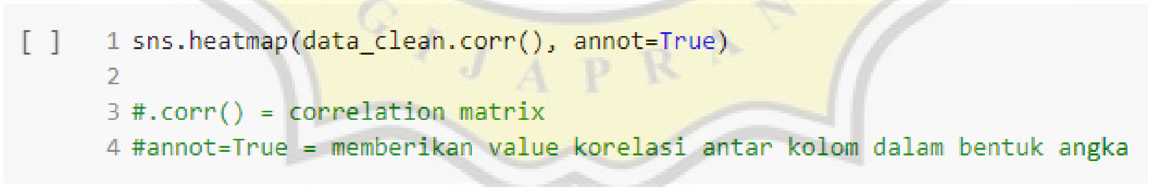
```
[23]  #Print rows dalam dataframe diabetes_df_copy2 yang not isin(tidak didalam) dataframe data_clean
      #lambang (~) menandakan NOT
      diabetes_df_copy2[~diabetes_df_copy2.index.isin(data_clean.index)]
```

Figure 5.17 Showing all of the data with outlier

- data_clean : show the new dataframe after cleaned with Z-Score
- diabetes_df_copy2[~diabetes_df_copy2.index.isin(data_clean.index)] : Print rows in the diabetes_df_copy2 dataframe that are not in the data_clean data frame

Data Correlation

After data preprocessing, we have to check the class correlation with other variable using Correlation Matrix. Correlation Matrix are used to see the relevance between columns. Here my column target is the "Output" column, and the relevance between the columns has a positive value. Then I will use all columns as indicators.



```
[ ]  sns.heatmap(data_clean.corr(), annot=True)

     #.corr() = correlation matrix
     #annot=True = memberikan value korelasi antar kolom dalam bentuk angka
```

Figure 5.18 Showing correlation matrix

- sns.heatmap(data_clean.corr(), annot=True) : Showing correlation matrix between column

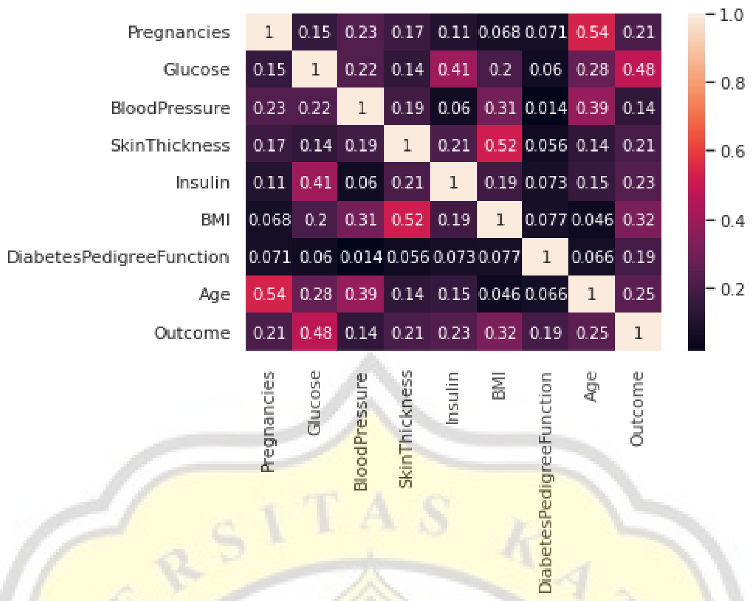
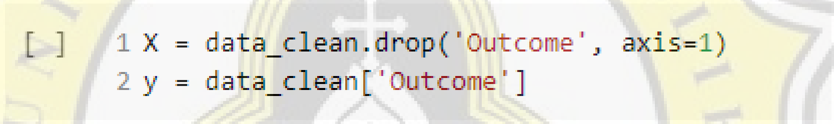


Figure 5.19 Show column relevances using Correlation Matrix Heatmap



```
[ ]  X = data_clean.drop('Outcome', axis=1)
     y = data_clean['Outcome']
```

Figure 5.20 Splitting X and y. X is the attributes, y is the result output

5.2. Results

We divided the dataset into 4 categories : 90% train and 10% test data, 80% train and 20% test data, 70% train and 30% test data, 60% train and 40% test data. Various operations and analyses using random state (0 and 42) in oversampling method and Decision Tree at the same time and data balance (imbalanced and balanced dataset) are represented here :

Algorithm	Evaluation metric	Train : Test			
		90:10	**80:20**	**70:30**	**60:40**
Decision Tree	Accuracy	88.42	88.88	82.33	79.57
	Precision	83.92	86.81	81.50	78.50
	Recall	95.91	89.77	83.80	82.19
	F1-Score	89.52	88.26	82.63	80.30
XGBoost	Accuracy	78.94	83.06	82.33	78.51
	Precision	79.59	78.57	83.33	80.55

	Recall	79.59	87.50	80.98	75.91
	F1-Score	79.59	82.79	82.14	78.16

Table 5.1 Model Evaluation with Random State = 0, Balanced Dataset

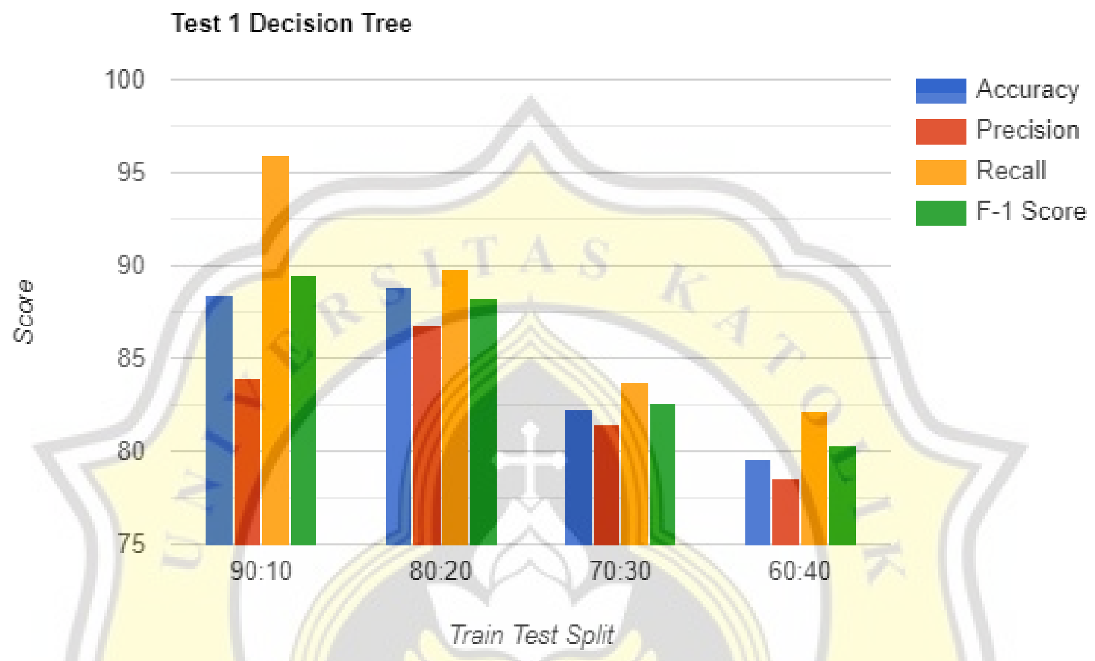


Figure 5.21 Decision Tree Model Evaluation with Random State = 0, Balanced Dataset

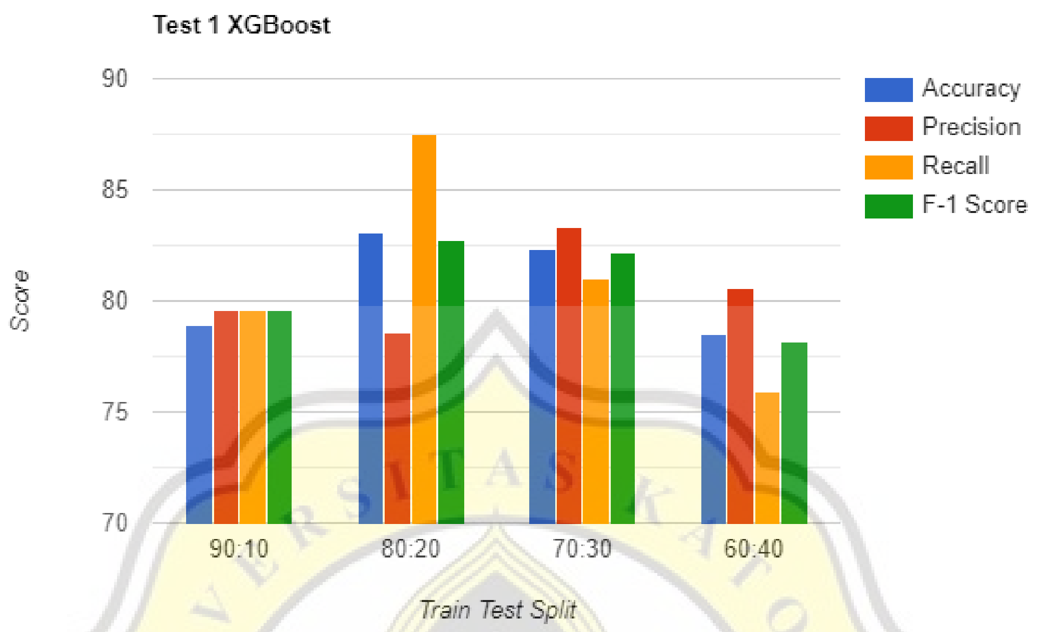


Figure 5.22 XGBoost Model Evaluation with Random State = 0, Balanced Dataset

With ratio of 90:10, For the Decision Tree, the model gets 88.42 % accuracy, precision gets 83.92%, recall gets 95.91%, and F1-Score gets 89.52%. For the XGBoost, the model gets 78.94 % accuracy, precision gets 79.59%, recall gets 79.59%, and F1-Score gets 79.59%.

With ratio of 80:20, For the Decision Tree, the model gets 88.88 % accuracy, precision gets 86.81%, recall gets 83.80%, and F1-Score gets 82.63 %. For the XGBoost, the model gets 83.06 %, accuracy, precision gets 78.57%, recall 87.50%, and F1-Score gets 82.79%.

With ratio of 70:30, For the Decision Tree, the model gets 82.33 % accuracy, precision gets 81.50%, recall gets 91.82%, and F1-Score gets 91.77 %. For the XGBoost, the model gets 82.33 % accuracy, precision gets 83.33%, recall gets 80.98%, and F1-Score gets 82.14%.

With ratio of 60:40, For the Decision Tree, the model gets 79.57 % accuracy, precision gets 78.50%, recall gets 82.19%, and F1-Score gets 80.30 %. For the XGBoost, the model gets 78.51 % accuracy, precision gets 80.55%, recall gets 75.91%, and F1-Score gets 78.16%.

Algorithm	Evaluation metric	Train : Test			
		90:10	**80:20**	**70:30**	**60:40**

Decision Tree	Accuracy	80.85	83.51	85.46	83.51
	Precision	70.90	77.98	81.64	80.90
	Recall	95.12	92.39	91.48	89.89
	F1-Score	81.25	84.57	86.28	85.16
XGBoost	Accuracy	78.72	82.44	83.68	82.71
	Precision	68.42	76.57	80.25	81.51
	Recall	95.12	92.39	89.36	86.86
	F1-Score	79.59	83.74	84.56	84.10

Table 5.2 Model Evaluation with Random State = 42, Balanced Dataset

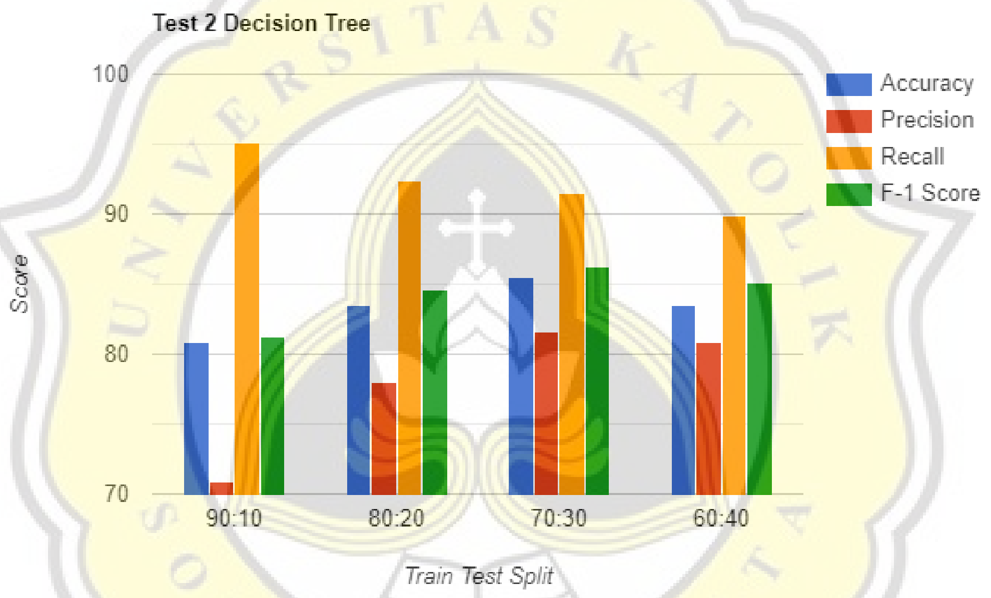


Figure 5.23 Decision Tree Model Evaluation with Random State = 42, Balanced Dataset

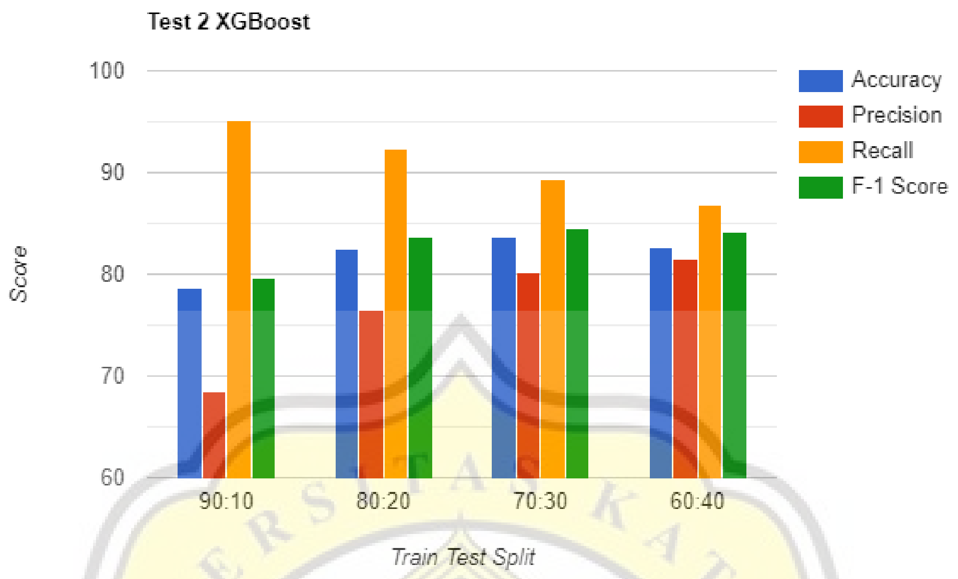


Figure 5.24 XGBoost Model Evaluation with Random State = 42, Balanced Dataset

With ratio of 90:10, For the Decision Tree, the model gets 80.85 % accuracy, precision gets 70.90%, recall gets 95.12%, and F1-Score gets 81.25%. For the XGBoost, the model gets 78.72 % accuracy, precision gets 68.42%, recall gets 95.12%, and F1-Score gets 79.59%.

With ratio of 80:20, For the Decision Tree, the model gets 83.51 % accuracy, precision gets 77.98%, recall gets 92.39%, and F1-Score gets 84.57 %. For the XGBoost, the model gets 82.44 %, accuracy, precision gets 76.57%, recall gets 92.39%, and F1-Score gets 83.74%.

With ratio of 70:30, For the Decision Tree, the model gets 85.46 % accuracy, precision gets 81.64%, recall gets 91.48%, and F1-Score gets 86.28 %. For the XGBoost, the model gets 83.68 % accuracy, precision gets 80.25%, recall gets 89.36%, and F1-Score gets 84.56%.

With ratio of 60:40, For the Decision Tree, the model gets 83.51 % accuracy, precision gets 80.90%, recall gets 89.89%, and F1-Score gets 85.16 %. For the XGBoost, the model gets 82.71 % accuracy, precision gets 81.51%, recall gets 86.86%, and F1-Score gets 84.10%.

Algorithm	Evaluation metric	Train : Test			
		90:10	**80:20**	**70:30**	**60:40**

Decision Tree	Accuracy	75.00	71.52	71.75	69.09
	Precision	55.55	55.35	55.00	50.45
	Recall	71.42	65.95	63.76	61.11
	F1-Score	62.50	60.19	59.06	55.27
XGBoost	Accuracy	83.33	77.77	79.16	77.43
	Precision	71.42	63.15	67.64	62.88
	Recall	71.42	76.59	66.66	67.77
	F1-Score	71.42	69.23	67.15	65.24

Table 5.3 Model Evaluation with Random State = 0, Imbalanced Dataset

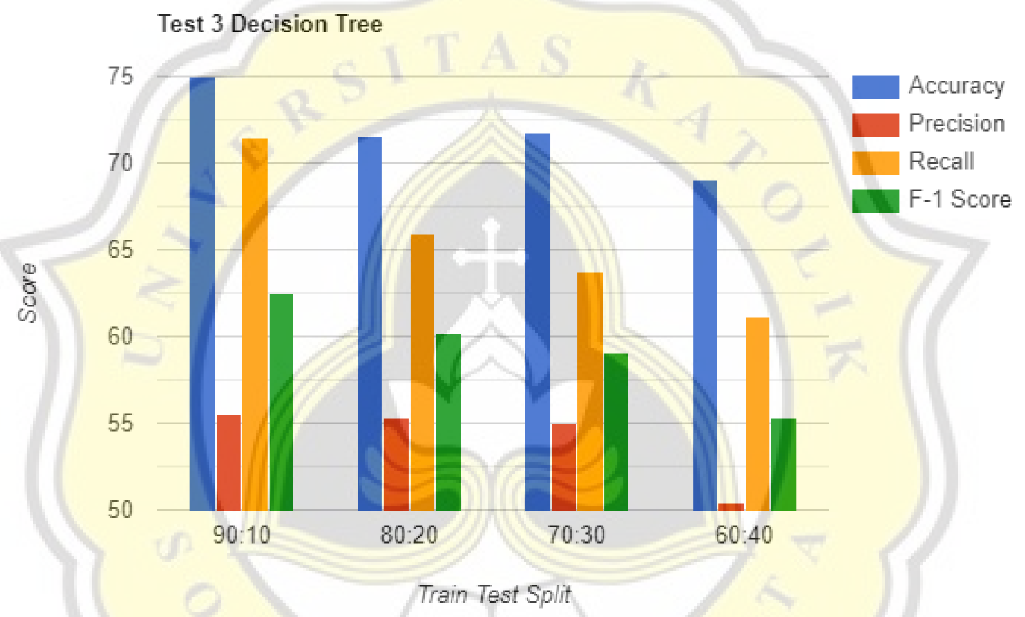


Figure 5.25 Decision Tree Model Evaluation with Random State = 0, Imbalanced Dataset

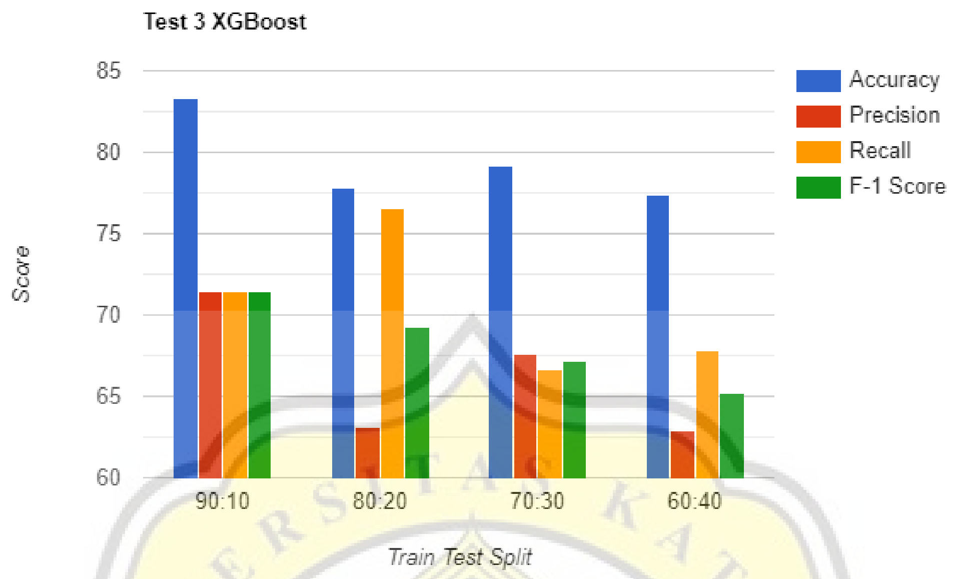


Figure 5.26 XGBoost Model Evaluation with Random State = 0, Imbalanced Dataset

With ratio of 90:10, For the Decision Tree, the model gets 75.00 % accuracy, precision gets 55.55%, recall gets 71.42%, and F1-Score gets 62.50%. For the XGBoost, the model gets 83.33 % accuracy, precision gets 71.42%, recall gets 71.42%, and F1-Score gets 71.42%.

With ratio of 80:20, For the Decision Tree, the model gets 71.52 % accuracy, precision gets 55.35%, recall gets 65.95%, and F1-Score gets 60.19 %. For the XGBoost, the model gets 77.77 %, accuracy, precision gets 63.15%, recall gets 76.59%, and F1-Score gets 69.23%.

With ratio of 70:30, For the Decision Tree, the model gets 71.75 % accuracy, precision gets 55.00%, recall gets 63.76%, and F1-Score gets 59.06 %. For the XGBoost, the model gets 79.16 % accuracy, precision gets 67.64%, recall gets 66.66%, and F1-Score gets 67.15%.

With ratio of 60:40, For the Decision Tree, the model gets 69.09 % accuracy, precision gets 50.45%, recall gets 61.11%, and F1-Score gets 55.27 %. For the XGBoost, the model gets 77.43 % accuracy, precision gets 62.88%, recall gets 67.77%, and F1-Score gets 65.24%.

Algorithm	Evaluation metric	Train : Test			
		90:10	**80:20**	**70:30**	**60:40**

Decision Tree	Accuracy	70.83	72.22	72.22	73.26
	Precision	47.82	56.86	52.63	53.68
	Recall	55.00	61.70	62.50	60.71
	F1-Score	51.16	59.18	57.14	56.98
XGBoost	Accuracy	79.16	67.44	79.62	81.59
	Precision	63.15	74.35	65.62	66.66
	Recall	60.00	61.70	65.62	73.80
	F1-Score	61.53	67.44	65.62	70.05

Table 4.4 Model Evaluation with Random State = 42, Imbalanced Dataset

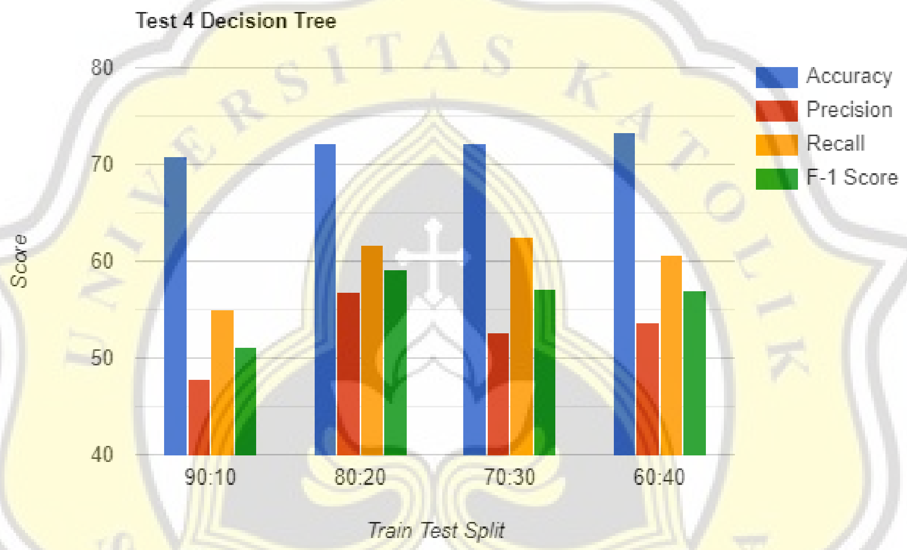


Figure 5.27 Decision Tree Model Evaluation with Random State = 42, Imbalanced Dataset

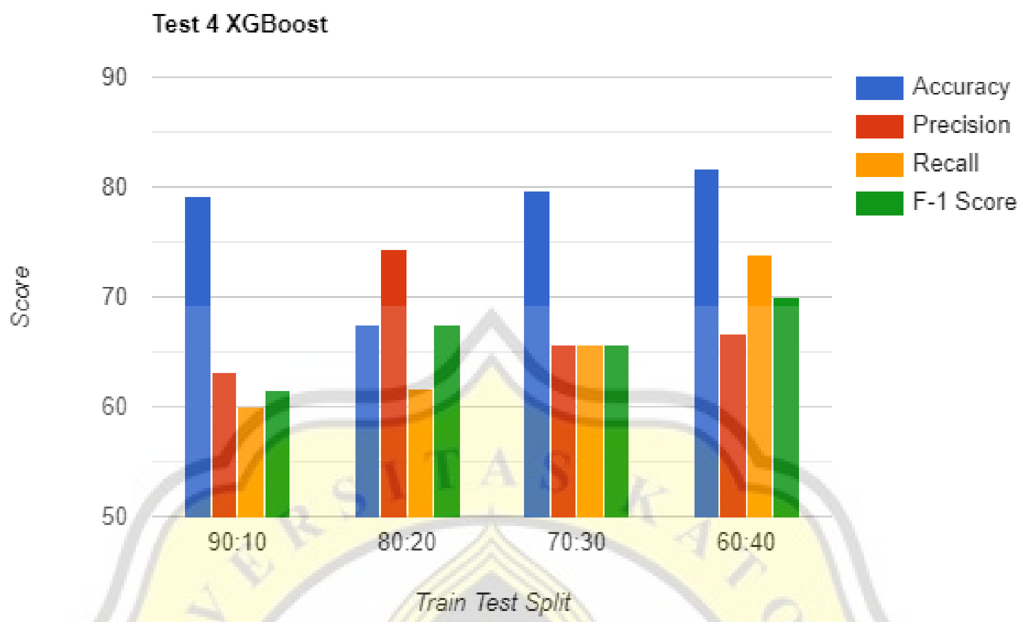


Figure 5.28 XGBoost Model Evaluation with Random State = 42, Imbalanced Dataset

With ratio of 90:10, For the Decision Tree, the model gets 70.83 % accuracy, precision gets 47.82%, recall gets 55.00%, and F1-Score gets 51.16%. For the XGBoost, the model gets 79.16 % accuracy, precision gets 63.15%, recall gets 60.00%, and F1-Score gets 61.53%.

With ratio of 80:20, For the Decision Tree, the model gets 72.22 % accuracy, precision gets 56.86%, recall gets 61.70%, and F1-Score gets 59.18 %. For the XGBoost, the model gets 67.44 %, accuracy, precision gets 74.35%, recall gets 61.70%, and F1-Score gets 67.44%.

With ratio of 70:30, For the Decision Tree, the model gets 72.22 % accuracy, precision gets 52.63%, recall gets 62.50%, and F1-Score gets 57.14 %. For the XGBoost, the model gets 79.62 % accuracy, precision gets 65.62%, recall gets 65.62%, and F1-Score gets 65.62%.

With ratio of 60:40, For the Decision Tree, the model gets 73.26 % accuracy, precision gets 53.68%, recall gets 60.71%, and F1-Score gets 56.98 %. For the XGBoost, the model gets 81.59 % accuracy, precision gets 66.66%, recall gets73.80%, and F1-Score gets 70.05%.

5.3. Evidence based on research which says that XGBoost has higher accuracy than the Decision Tree

Algorithm	Evaluation metric	Train : Test			
		90:10	**80:20**	**70:30**	**60:40**
Decision Tree	Accuracy	70.83	69.44	66.66	71.52
	Precision	50.00	52.45	48.42	53.38
	Recall	71.42	68.08	66.66	70.00
	F1-Score	58.82	59.25	56.09	60.57
XGBoost	Accuracy	79.16	79.16	78.24	76.04
	Precision	66.66	66.03	66.17	60.00
	Recall	57.14	74.46	65.21	70.00
	F1-Score	61.53	70.00	65.69	64.61

Table 5.5 Model Evaluation with Random State = 0, Imbalanced Dataset without Age Attributes

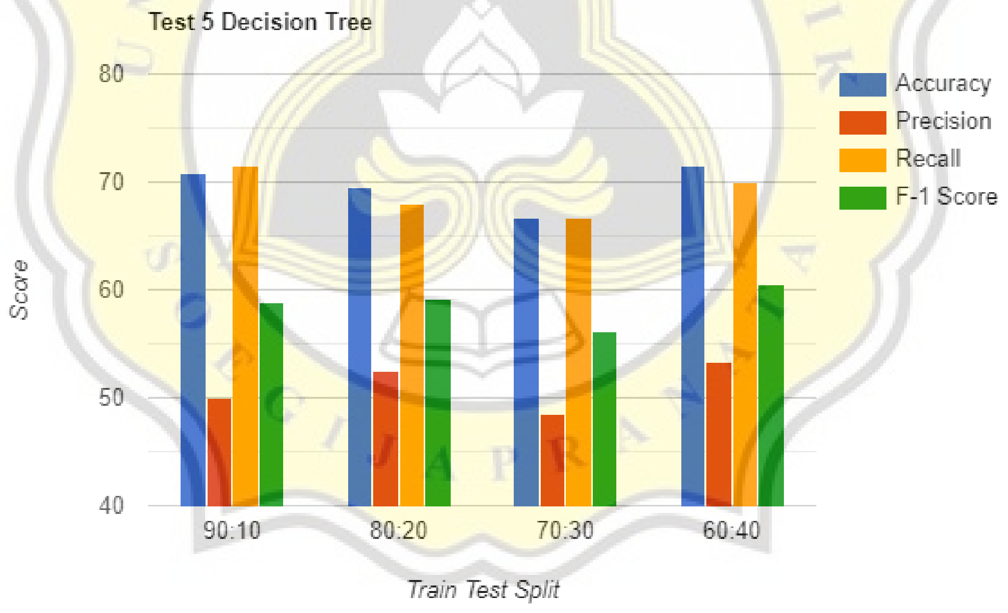


Figure 5.29 Decision Tree Model Evaluation with Random State = 0, Imbalanced Dataset without Age Attributes

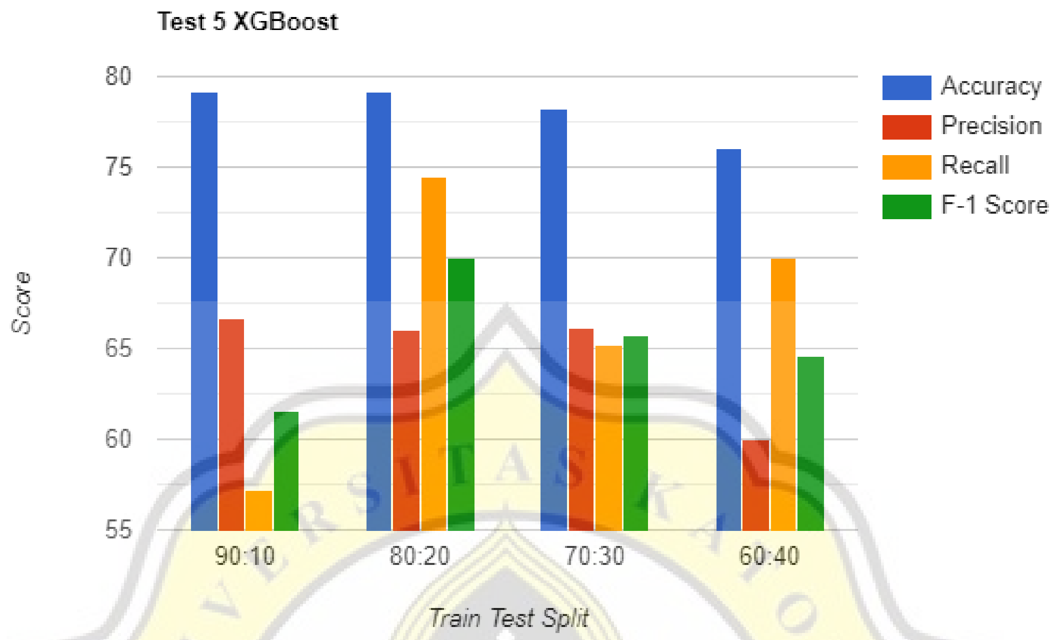


Figure 5.30 XGBoost Model Evaluation with Random State = 0, Imbalanced Dataset without Age Attributes

With ratio of 90:10, For the Decision Tree, the model gets 70.83 % accuracy, precision gets 50.00%, recall gets 71.42%, and F1-Score gets 58.82%. For the XGBoost, the model gets 79.16 % accuracy, precision gets 66.66%, recall gets 57.14%, and F1-Score gets 61.53%.

With ratio of 80:20, For the Decision Tree, the model gets 69.44 % accuracy, precision gets 52.45%, recall gets 68.08%, and F1-Score gets 59.25 %. For the XGBoost, the model gets 79.16 %, accuracy, precision gets 66.03%, recall gets 74.46%, and F1-Score gets 70.00%.

With ratio of 70:30, For the Decision Tree, the model gets 66.66 % accuracy, precision gets 48.42%, recall gets 66.66%, and F1-Score gets 56.09 %. For the XGBoost, the model gets 78.24 % accuracy, precision gets 66.17%, recall gets 65.21%, and F1-Score gets 65.69%.

With ratio of 60:40, For the Decision Tree, the model gets 71.52 % accuracy, precision gets 53.38%, recall gets 70.00%, and F1-Score gets 60.57 %. For the XGBoost, the model gets 76.04 % accuracy, precision gets 60.00%, recall gets 70.00%, and F1-Score gets 64.61%

Algorithm	Evaluation metric	Train : Test			
		90:10	**80:20**	**70:30**	**60:40**
Decision Tree	Accuracy	69.44	71.52	71.29	73.26
	Precision	45.45	56.00	51.35	53.33
	Recall	50.00	59.57	59.37	66.66
	F1-Score	47.61	57.73	55.07	59.25
XGBoost	Accuracy	77.77	78.47	79.16	80.20
	Precision	62.50	70.00	66.10	66.66
	Recall	50.00	59.57	60.93	64.28
	F1-Score	55.55	64.36	63.41	65.45

Table 5.6 Model Evaluation with Random State = 42, Imbalanced Dataset without Age Attributes

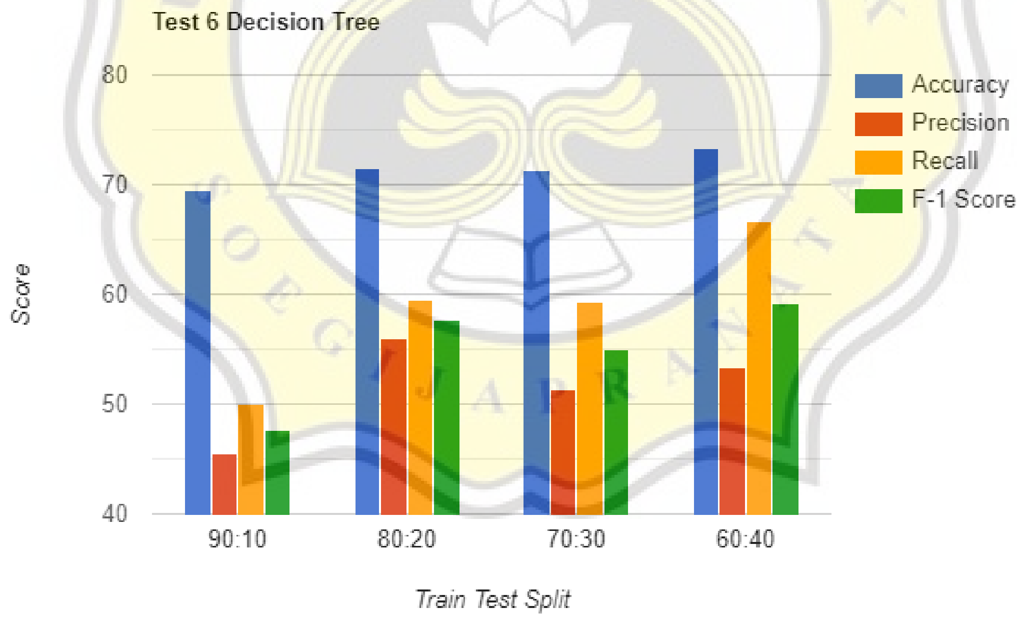


Figure 5.31 Decision Tree Model Evaluation with Random State = 42, Imbalanced Dataset without Age Attributes

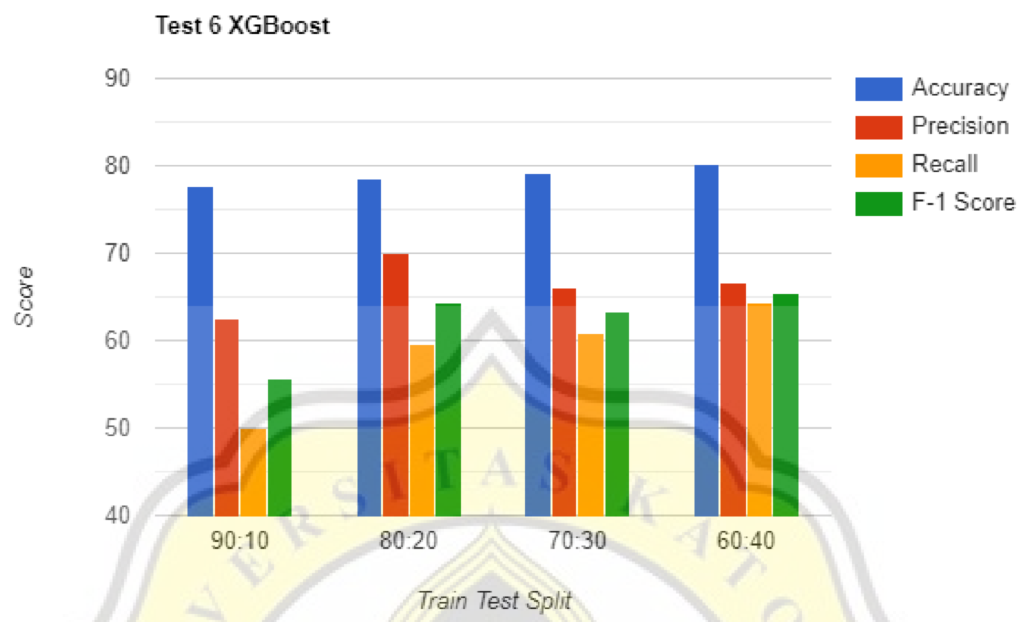


Figure 5.32 XGBoost Model Evaluation with Random State = 42, Imbalanced Dataset without Age Attributes

With ratio of 90:10, For the Decision Tree, the model gets 69.44 % accuracy, precision gets 45.45%, recall gets 50.00%, and F1-Score gets 47.61%. For the XGBoost, the model gets 77.77 % accuracy, precision gets 62.50%, recall gets 50.00%, and F1-Score gets 55.55%.

With ratio of 80:20, For the Decision Tree, the model gets 71.52 % accuracy, precision gets 56.00%, recall gets 59.57%, and F1-Score gets 57.73 %. For the XGBoost, the model gets 78.47 %, accuracy, precision gets 70.00%, recall gets 59.57%, and F1-Score gets 54.36%.

With ratio of 70:30, For the Decision Tree, the model gets 71.29 % accuracy, precision gets 51.35%, recall gets 59.37%, and F1-Score gets 55.07 %. For the XGBoost, the model gets 79.16 % accuracy, precision gets 66.10%, recall gets 60.93%, and F1-Score gets 63.41%.

With ratio of 60:40, For the Decision Tree, the model gets 73.26 % accuracy, precision gets 53.33%, recall gets 66.66%, and F1-Score gets 59.25 %. For the XGBoost, the model gets 80.20 % accuracy, precision gets 66.66%, recall gets 64.28%, and F1-Score gets 65.45%