# CHAPTER 5
# IMPLEMENTATION AND RESULT

## 5.1. Implementation

The research is implemented by using the following code:

```
1       data=pd.read_csv("./dataset_SKRIPSI/DATASET_SKRIPSI_TRANSPOSED2.csv",
        delimiter=';')

2       data["Tahun"] = pd.to_datetime(data["Tahun"])
```

The first line of the code are basic commands to read a csv file containing the original dataset of medical workforce number. This csv file is already preprocessed using Microsoft Excel. The second line converts “Tahun” column to datetime format in order to make it compatible with SDV library.

```
3       n=split*len(data)

4       df_1 = data.iloc[:int(n),:]

5       df_2 = data.iloc[int(n):,:]

6       column_names = data.columns.values.tolist()
```

Line 3 declares split number based on percentage value inputted to the code in the split variable. The original data is then split according to the split value. Detail 1 being active means the code uses only the original data to train and test the algorithms. Detail 2 being active means the code test/evaluate the algorithms using the original data only. Column names variable is used to display or point currently experimented column.

```
7       model = PAR(sequence_index="Tahun")

8       model.fit(df_1)
```

Line 7 assigns synthetic data model with “Tahun” as the sequence or time series index. The data is then fitted to the said model.

```
9       ins=[5, 10, 15, 20, 25]

10      time_data=[]

11      start_time = time.time()

12      new_data = model.sample(num_sequences=ins[0])

13      time_data.append(time.time() - start_time)

14      print("running time = ", (time.time() - start_time), " seconds")
```

Line 9 declares the number of sequences you desire. Sequence affects the total number of generated data. Line 10 declares time data variable used to count how much seconds passed during the data generations. Line 12 generates the new dataset based on the original dataset and number of previously declared sequence. Line 13 calculates time difference between the starting time declaration that is Line 14 and the present.

```
15      stat1 = data.describe()

16      stat2 = new_data.describe()

17      plt.figure(figsize=(14,10))

18      x_desc = ["original data", "5 seq"]

19      y_desc=[stat1.Anestesi[0],stat2.Anestesi[0]]

20      plt.subplot(2,2,1)

21      plt.plot(x_desc,y_desc,"-b",marker = '|',label='number of data')

22      plt.legend(loc="upper left")

23      x_desc = ["original data", "5 seq"]

24      y_desc = [stat1.Anestesi[1],stat2.Anestesi[1]]

25      plt.subplot(2,2,2)

26      plt.plot(x_desc,y_desc,"-r",marker = '|',label='mean')

27      plt.legend(loc="upper left")
```

This part of the code visualizes some notable stats the generated datasets possess. Stat 1 variable visualizes the original data, and Stat 2 visualizes the generated data. Describe function gives deviation standard, mean, max, min, and data amount possessed by each column in the dataset. Line 17 to 27 visualizes the stats using matplot library.

```
28      test_value=[[0]*200 for i in range(21)]

29      i=0

30      if detail[1]==0: new_data=df_1

31      for _ in range(20):

32              if detail[2]==0: test_value[i]=df_2[column_names[i+1]].values

33              elif detail[2]==1: test_value[i]=data[column_names[i+1]].values

34              i=i+1
```

Line 28 declares test data to be used as evaluator. Test data is chosen between two datasets, the first one being the byproduct of data split if the technique is used, the second one being the original dataset. Line 31 to 34 injects data from appropriate column from the original data to the test value variable.

```
35    x=[]
36    year_init=2015-len(new_data)
37    year=year_init
38    for _ in range(int(len(new_data)):
39          year=year+1
40          x.append(year)
41    if detail[2]==0:
42          if detail[1]==1:
43                x=[]
44                x=new_data['Tahun'].values
45    x = np.array(x)
46    x = x.reshape(-1, 1)
47    y = new_data.iloc[:,(col+1):(col+2)].values.astype(int)
48    regressor = SVR()
49    regressor.fit(x,y.ravel())
50    regressor2 = RandomForestRegressor()
51    regressor2.fit(x, y.ravel())
```

Line 36 declares initial year variable containing year until 2015. The purpose is to fill out the time data, as SDV does not generate year. Training data is fitted into regressor models for the SVR and RF prediction to work. Both algorithms are implemented using sklearn library. X and Y are used to define the time and value axis for the prediction respectively.

```
52    year=2015
53    for j in test_value[col]:
54          year=year+1
55          test_year.append(year)
56    if detail[1]==1:
57          test_year=df_2['Tahun'].values
58    if detail[2]==1:
59          test_year=data['Tahun'].values
60    y_pred = []
```

```
for j in test_year:
        y_pred.append(int(regressor.predict([[j]])))
Y_pred = []
for j in test_year:
        Y_pred.append(int(regressor2.predict(np.array(j).reshape(1, 1))))
```

This part predicts the future medical workforce and appends the predicted value inside variables for testing. In this case, the variables are y_pred and Y_pred, containing SVR and RF predictions respectively.

```
def MAPE(test_data, predicted_data):
        result = np.empty(test_data.shape)
        for j in range(test_data.shape[0]):
                result[j] = (test_data[j] - predicted_data[j]) / test_data[j]
                result = np.mean(np.abs(result)) * 100
        return result
mse_svr = np.mean(((test_value[col] - np.array(y_pred)) ** 2))
mse_rf = np.mean(((test_value[col] - np.array(Y_pred)) ** 2))
mape_svr = MAPE(np.asarray(test_value[col]),np.asarray(y_pred))
mape_rf = MAPE(np.asarray(test_value[col]),np.asarray(Y_pred))
```

The prediction results are tested with the rest of split data, or using the original dataset. MAPE and MSE are calculated by utilizing numpy library to achieve the formula. The accuracy test results are stored in variables for analysis.

## 5.2. Result

### 5.2.1. Comparison between Original Data and SDV Generated Data

**Table 3: SDV Running Time**

| N of Sequence | N of Data Generated | Running Time (s) |
|---|---|---|
| 1 | 160 | 0.198 |
| 5 | 800 | 0.758 |
| 10 | 1600 | 1.499 |
| 15 | 2400 | 2.271 |
| 20 | 3200 | 3.037 |
| 25 | 4000 | 3.761 |

During the experiments, SDV requires significant amount of time to finish. For 25 sequence which generates 4000 data, processing time reached above 3 seconds during the data generation phase. Processing time increases consistently throughout each addition of sequence, around 0.7 seconds for every 5 additional sequences which in this context 800 data. The following plot is the stats of generated data.

Processing time might differ between system, and could be dependent to individual processor and RAM. This time data is tested using Intel i7 10870H processor with 8 gigabytes of RAM. During the experiment, RAM load saw about 20% increase, while CPU saw only 10% increase. This means there is no bottleneck happening within the author's system.

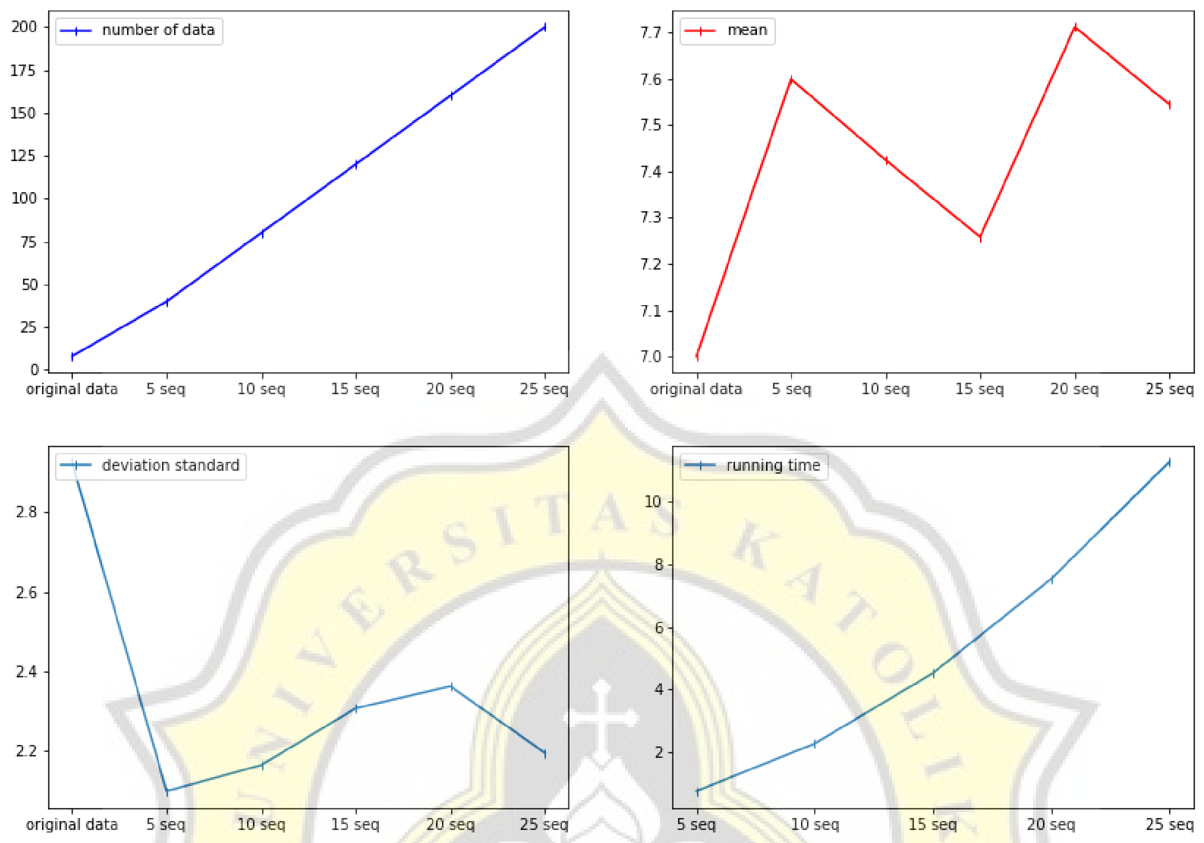


**Figure 1: Stats of SDV Generated Data using Forensic Index**

Figure 1 shows statistical description of the original and generated data. Sequences in this context closely tied to the number of generated data as mentioned in previous sections. As visualized by the Figure 1, data generated from this method has much lower deviation standard than the original data. Lower deviation means the generated datasets are not as chaotic as the original data, as their value are closer to the mean. Mean values of the generated datasets fluctuate a lot, but the value wise, they are close to the original data. The generated dataset values are also notably higher than the original mean value.

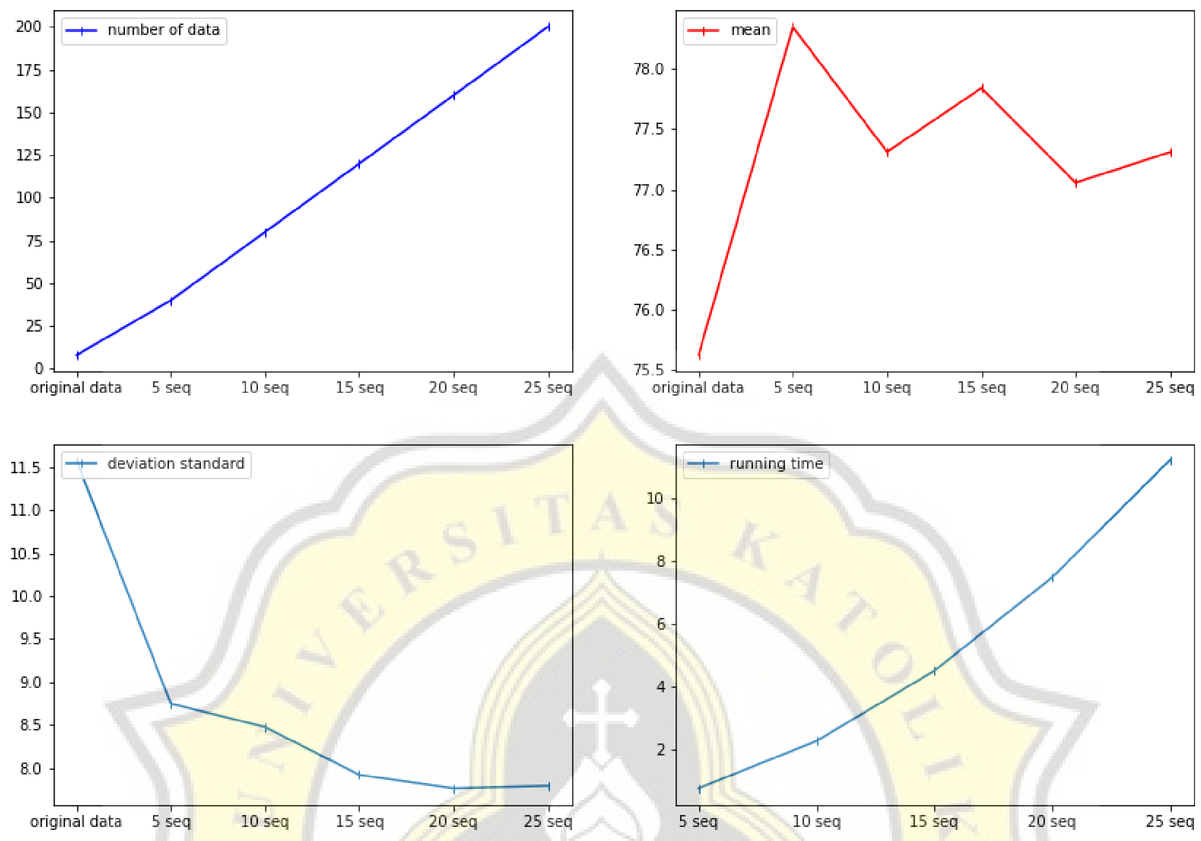


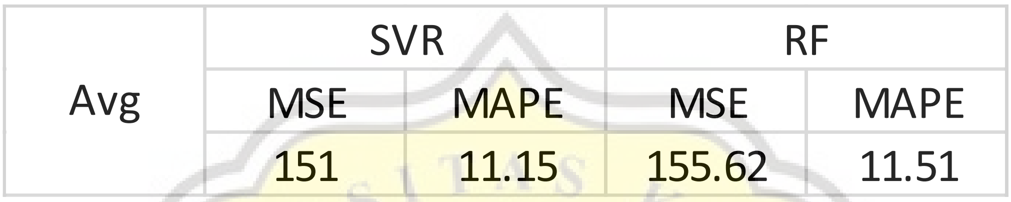
**Figure 2: Stats of SDV Generated Data using Bedah Index**

Judged from Figure 2, the deviation standard decrease seems consistent throughout different data stats. The Figure 2 uses Bedah column index which has different data scale than Figure 1. Running time did not change much between different data scales. Mean stat tendency, however, seems to vary greatly between different indexes. Mean values of generated data in Figure 1 seems fluctuates more than in Figure 2, in which the values are more stable. There is no apparent correlation between the changes in mean value, and the usage of SDV as far as the author can tell.

### 5.2.2. Comparison of Accuracy between SDV data and Original

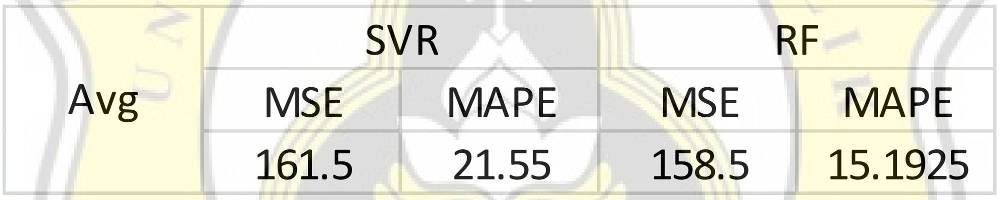
Accuracy tests are visualized in the following spreadsheet format. Average score is calculated using mean of MSE and MAPE from all columns.

**Table 4: SDV Generated Data Error Value 6 data**



|  | SVR |  | RF |  |
|---|---|---|---|---|
| Avg | MSE | MAPE | MSE | MAPE |
|  | 151 | 11.15 | 155.62 | 11.51 |

**Table 5: Original Data Error Value**



|  | SVR |  | RF |  |
|---|---|---|---|---|
| Avg | MSE | MAPE | MSE | MAPE |
|  | 161.5 | 21.55 | 158.5 | 15.1925 |

Table 4 contains error values of prediction algorithms fitted with SDV generated data, and evaluated with the original dataset. In Table 5, the original dataset is split to provide both training and testing data. Table 4 uses 1 sequence or 6 data generated by SDV. In this comparison both SDV generated data and original data are equal in data amount. Both error value data are taken from the average of three experiments, to ensure their consistency.

The result is mildly satisfying in SDV's favor. The average SDV MSE and MAPE score for both SVR and RF are lower than using just the original dataset, albeit just a little bit. SVR in particular saw a significant decrease in MAPE value, which is almost half of the original data. SVR MSE value also saw a considerable decrease from 161 to 151.

**Table 6: SDV Generated Data Error Value 30 data**

| Avg | SVR | | RF | |
|---|---|---|---|---|
| | MSE | MAPE | MSE | MAPE |
| | 161.37 | 11.24 | 158.5 | 11.45 |

On 30 data as shown in Table 6, the error value of both SVR and RF rose slightly. The error values did not exceed the original data, but are very close. Only SVR MAPE value managed to stay far below the original. RF MSE error value in this experiment is equal to the original dataset. SDV generated dataset is still better than the original data in this experiment overall.

**Table 7: SDV Generated Data Error Value 60 data**

| Avg | SVR | | RF | |
|---|---|---|---|---|
| | MSE | MAPE | MSE | MAPE |
| | 161.37 | 11.24 | 160.75 | 12.48 |

**Table 8: SDV Generated Data Error Value 90 data**

| Avg | SVR | | RF | |
|---|---|---|---|---|
| | MSE | MAPE | MSE | MAPE |
| | 149.12 | 11.41 | 147 | 11.05 |

On 60 data as shown in Table 7, the error value of RF rose slightly. SVR error value is completely unchanged from the previous experiment. RF MAPE score rose slightly, but the change is very small. On 90 data shown in Table 8, both algorithm error value fell significantly, surpassing the accuracy of using 6 SDV data. In this experiment, MAPE error value is at its lowest achieved by RF.

**Table 9: SDV Generated Data Error Value 120 data**

|  | SVR |  | RF |  |
|---|---|---|---|---|
| Avg | MSE | MAPE | MSE | MAPE |
|  | 149.12 | 11.96 | 139 | 11.41 |

On 120 data, the error value fell significantly for RF, while SVR remained unchanged. This marked the lowest MSE error value achieved in this experiment achieved by RF.

**Table 10: SDV Generated Data Error Value 150 data**

|  | SVR |  | RF |  |
|---|---|---|---|---|
| Avg | MSE | MAPE | MSE | MAPE |
|  | 161.37 | 11.24 | 146.5 | 11.13 |

On 150 data, Random Forest Algorithm saw an increase in both MSE and MAPE score compared to 90 data. RF on the other hand, saw very little decrease to its error values from both MSE and MAPE. SDV generated dataset, once again, performed better than the original. 90 to 120 data seem to be the most optimal number of data to train RF algorithm in the context of this research. SVR also achieved its lowest MSE error value at 90 to 120 data, while its MAPE error value is the lowest on 6 data.

From these five data, SDV generated dataset performs better than the original dataset. The error value differences between the original data and SDV generated data are not massive, but prediction with synthetic data almost consistently more accurate than using the original dataset.

### 5.2.3. Comparison of Accuracy SVR and RF Algorithm for SDV

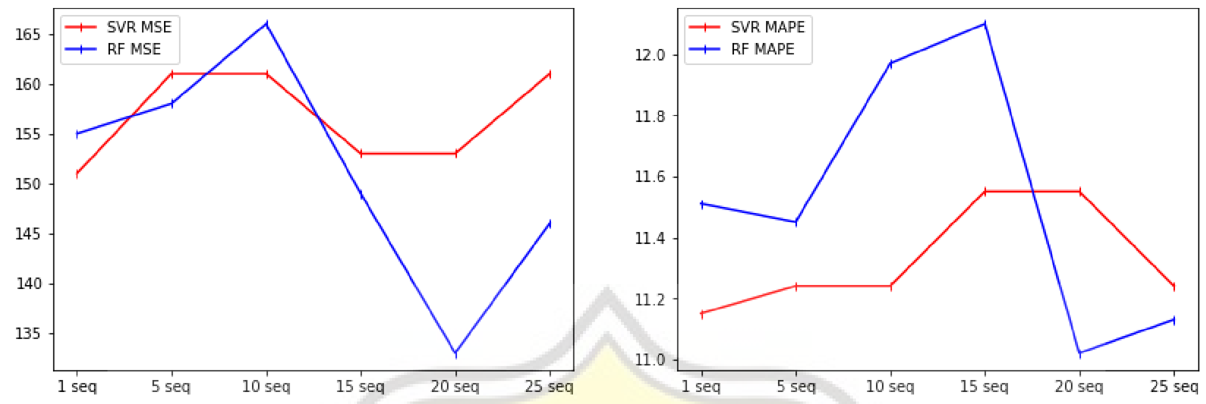


**Figure 3: SVR and RF using SDV**

Error value shown by Figure 3 shown that both algorithms scored very similar. Overall, there is no decisive answer as to which algorithm is better. According to MSE score, RF produced less error. According to MAPE score, however, SVR produced less error. It is worth noting that Random Forest performed better for the original dataset according to Table 5.

Random Forest seems to predict better at higher data availability as shown in Figure 3. SVR only managed to maintain better performance over Random Forest at lower and mid data availability, which are around 1 sequence to 15 sequences. SVR is also the most positively affected algorithm, as the algorithm saw significant increase of accuracy by using SDV dataset at lower sequence number based on Table 4.