ASSESSMENT REPORT

CII3C3 Machine Learning

Completed to fulfil clustering assignment in CII3C3-IF-43-INT Machine Learning

Muhammad Furqon Fahlevi / 1301194214

Clustering

COMPUTER SCIENCE MAJOR

SCHOOL OF COMPUTING

2021

A. Problem Formulation

There is a vehicle dataset that has multiple variables or columns. Datasets will be identified and grouped similar data points in a larger data set or called *Clustering*. Prior to the clustering stage, the datasets will go through a pre-processing or dropping of data stage before it is used to ensure or enhance performance. Datasets that have gone through the pre-processing stage will go through the next stage, namely clustering. In the clustering stage, only 2 variables or columns will be used, namely 'Umur' and 'Premi'. From the two selected data, it is found that the correlation from the heatmap is 0.06. Based on the problem from the dataset, I chose the K-Mean algorithm because it is more often used by many people and the time in running the algorithm is relatively very fast

B. Data Exploration and Preparation

1. Import Library

There are several libraries used for clustering.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy import stats
```

2. Import Datasets

Using pandas as pd to import csv file named “data_train”.

```
# Import dataset into new dataframe named data_train
data_train = pd.read_csv("kendaraan_train.csv")
data_train.head()
```

	id	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertar
0	1	Wanita	30.0	1.0	33.0	1.0	< 1 Tahun	Tidak	28029.0	152.0	97.0	
1	2	Pria	48.0	1.0	39.0	0.0	> 2 Tahun	Pernah	25800.0	29.0	158.0	
2	3	NaN	21.0	1.0	46.0	1.0	< 1 Tahun	Tidak	32733.0	160.0	119.0	
3	4	Wanita	58.0	1.0	48.0	0.0	1-2 Tahun	Tidak	2630.0	124.0	63.0	
4	5	Pria	50.0	1.0	35.0	0.0	> 2 Tahun	NaN	34857.0	88.0	194.0	

3. Pre-processing for dataset

3.1. Check NaN data

```
# Check NaN data
data_train.isna()
```

	id	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan
0	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False
2	False	True	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	True	False	False	False

285831 rows × 12 columns

Cleansing data or drop the data column or variable that has a NaN value. This serves to avoid empty data while doing the processing.

```
# Drop NaN data
data_train = data_train.dropna()
data_train
```

	id	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangga
0	1	Wanita	30.0	1.0	33.0	1.0	< 1 Tahun	Tidak	28029.0	152.0	9
1	2	Pria	48.0	1.0	39.0	0.0	> 2 Tahun	Pernah	25800.0	29.0	15
3	4	Wanita	58.0	1.0	46.0	0.0	1-2 Tahun	Tidak	2630.0	124.0	6
5	6	Pria	21.0	1.0	35.0	1.0	< 1 Tahun	Tidak	22735.0	152.0	17
8	9	Wanita	20.0	1.0	8.0	1.0	< 1 Tahun	Tidak	30786.0	160.0	3

171068 rows × 12 columns

3.2. Check duplicated data

To check is there any duplicated or no.

```
# Check duplicate data
print("Duplicated data:",data_train.duplicated().sum())
```

```
Duplicated data: 0
```

3.3.Change categorical into numeric data

Algorithm cannot operate on label data directly. They require all input and output variables to be numeric.

```
# Categorical -> Numeric data
data_train['Jenis_Kelamin'] = data_train['Jenis_Kelamin'].map({'Pria':0, 'Wanita':1})
data_train['Kendaraan_Rusak'] = data_train['Kendaraan_Rusak'].map({'Tidak':0, 'Pernah':1})
data_train['Umur_Kendaraan'] = data_train['Umur_Kendaraan'].map({'< 1 Tahun':0, '1-2 Tahun':1, '> 2 Tahun':2})
data_train.head()
```

	id	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan	Tertar
0	1	1	30.0	1.0	33.0	1.0	0	0	28029.0	152.0	97.0	
1	2	0	48.0	1.0	39.0	0.0	2	1	25800.0	29.0	158.0	
3	4	1	58.0	1.0	46.0	0.0	1	0	2630.0	124.0	63.0	
5	6	0	21.0	1.0	35.0	1.0	0	0	22735.0	152.0	171.0	
8	9	1	20.0	1.0	8.0	1.0	0	0	30786.0	160.0	31.0	

3.4. Drop unnecessary data

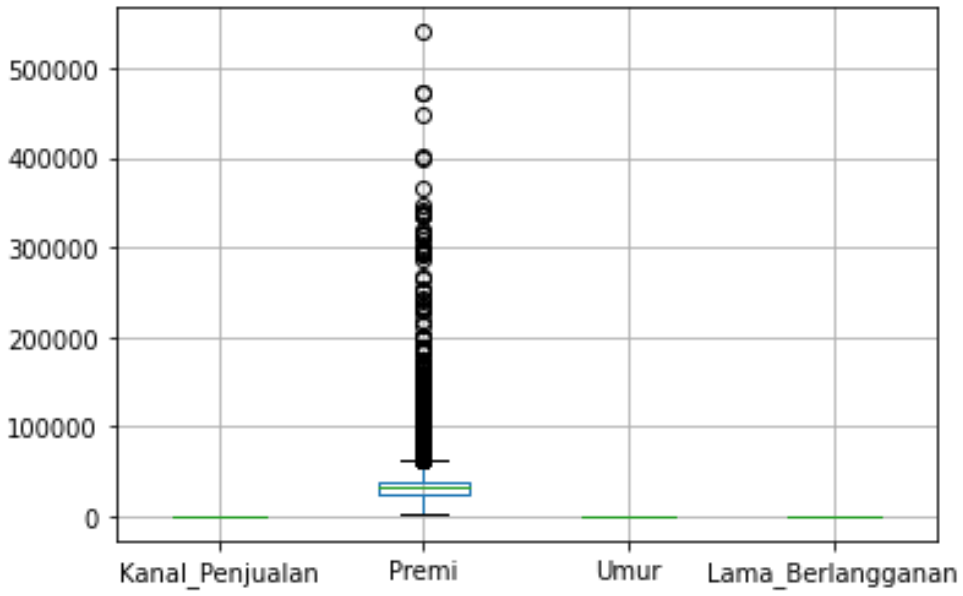
```
# Drop unnecessary data
del data_train['id']
del data_train['Tertarik']
data_train.head()
```

	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan
0	1	30.0	1.0	33.0	1.0	0	0	28029.0	152.0	97.0
1	0	48.0	1.0	39.0	0.0	2	1	25800.0	29.0	158.0
3	1	58.0	1.0	48.0	0.0	1	0	2630.0	124.0	63.0
5	0	21.0	1.0	35.0	1.0	0	0	22735.0	152.0	171.0
8	1	20.0	1.0	8.0	1.0	0	0	30786.0	160.0	31.0

3.5. Check outliers

Check the outliers for these columns because this is the candidate who may be continued to the clustering stage.

```
data_train[['Kanal_Penjualan', 'Premi', 'Umur', 'Lama_Berlangganan']].boxplot()
```



Drop outliers using z-score

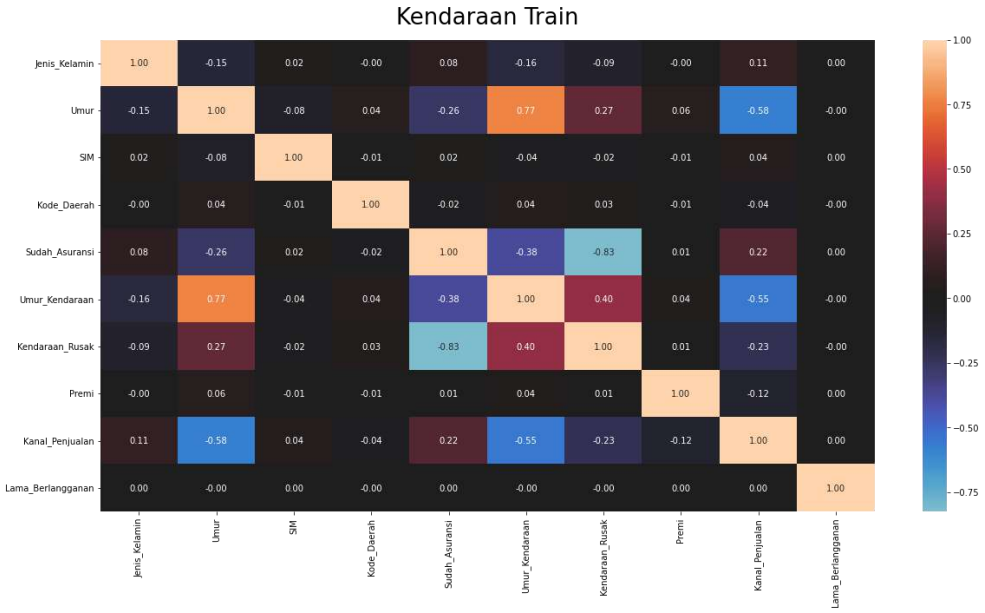
```
z = np.abs(stats.zscore(data_train[['Kanal_Penjualan', 'Premi', 'Umur', 'Lama_Berlangganan']]))
threshold = 3

data_new = data_train[(z < threshold).all(axis=1)]
data_new
```

	Jenis_Kelamin	Umur	SIM	Kode_Daerah	Sudah_Asuransi	Umur_Kendaraan	Kendaraan_Rusak	Premi	Kanal_Penjualan	Lama_Berlangganan
0	1	30.0	1.0	33.0	1.0	0	0	28029.0	152.0	97.0
1	0	48.0	1.0	39.0	0.0	2	1	25800.0	29.0	158.0
3	1	58.0	1.0	48.0	0.0	1	0	2630.0	124.0	63.0
5	0	21.0	1.0	35.0	1.0	0	0	22735.0	152.0	171.0
8	1	20.0	1.0	8.0	1.0	0	0	30786.0	160.0	31.0

3.6. Heatmap

This is the correlation between columns or variables

```
fig, ax = plt.subplots(figsize=(20,10))
heatmap = sns.heatmap(data_new.corr(), center=0, annot=True, fmt=".2f")
heatmap.set_title('Kendaraan Train', fontdict={'fontsize':26}, pad=18)
```



C. Modeling

1. Select 2 variables or columns for clustering

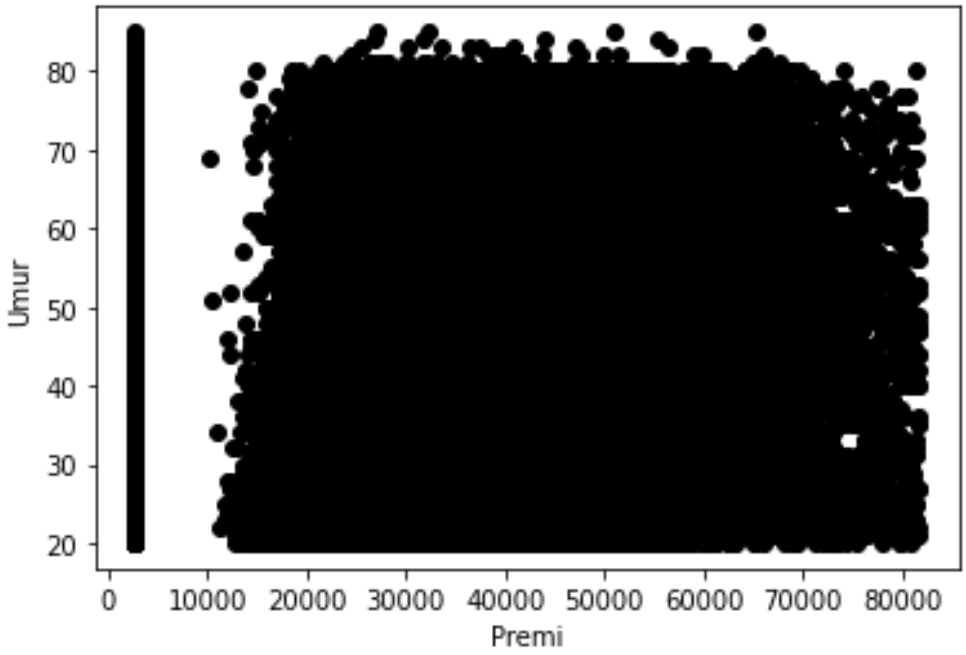
We choose ‘Premi’ and ‘Umur’ because we can see from the heatmap correlation they are about 0.06.

```
# Select 2 datas / columns for clustering
data_selected = data_new.loc[:, ['Premi', 'Umur']]
data_selected.head()
```

	Premi	Umur
0	28029.0	30.0
1	25800.0	48.0
3	2630.0	58.0
5	22735.0	21.0
8	30786.0	20.0

2. Data visualization before clustering

```
# Data visualization before clustering
ds_val = data_selected.values

plt.scatter(ds_val[:,0],ds_val[:,1], color='black')
plt.xlabel('Premi')
plt.ylabel('Umur')
plt.show()
```



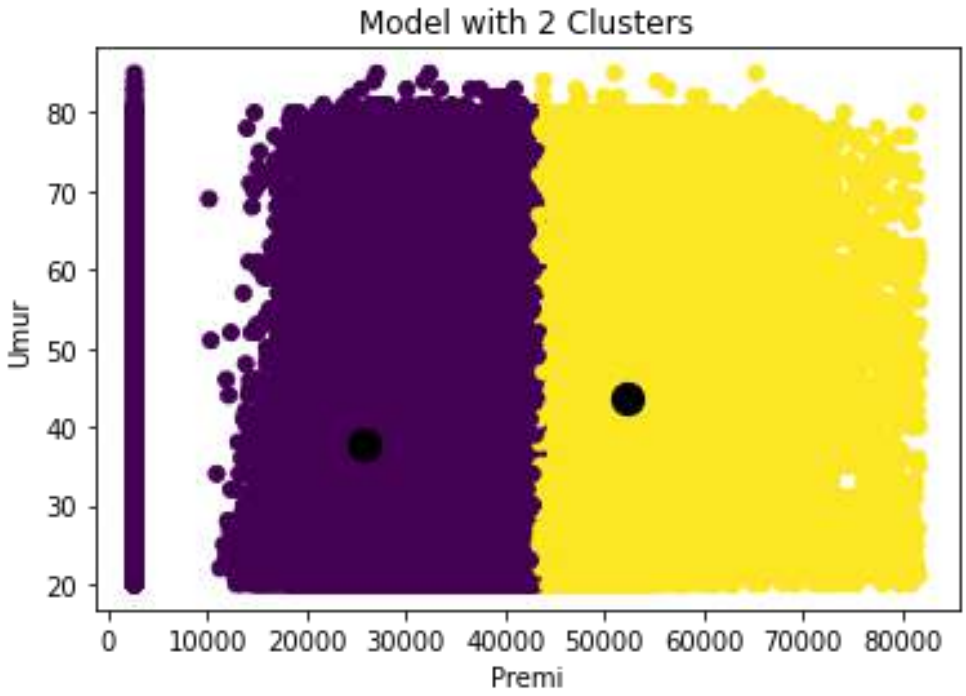
3. **Define K-Means**

   For the clustering stage, will be use the K-Means algorithm. First iteration after while is for the clustering, second iteration is to calculate the centroids using Euclidian Distance. The output for this K-Means will be returning the cluster and centroids value.

```python
# Define K-Means
def kMeans(ds_val, k):
  diff = 1
  cluster = np.zeros(ds_val.shape[0])
  centroids = data_selected.sample(n=k).values
  while diff:
    for i, row in enumerate(ds_val):
      distance = float('inf')
      for idx, centroid in enumerate(centroids):
        eu_distance = np.sqrt((centroid[0]-row[0])**2 + (centroid[1]-row[1])**2)
        if distance > eu_distance:
          distance = eu_distance
          cluster[i] = idx
    new_centroids = pd.DataFrame(ds_val).groupby(by=cluster).mean().values
    if np.count_nonzero(centroids-new_centroids) == 0:
      diff = 0
    else:
        centroids = new_centroids
    return centroids, cluster
```

4. **Data visualization after clustering**

   Data visualization after clustering stage with K = 2



5. **Elbow Line & WCSS**

   5.1. WCSS

   WCSS is the sum of squared distance between each point and the centroid in a cluster. Calculate the WCSS based on the formula, and we implement into the code.

$$\text{WCSS} = \sum_{C_k}^{C_n} \left(\sum_{d_i in C_i}^{d_m} distance(d_i, C_k)^2 \right)$$

Where,

C is the cluster centroids and d is the data point in each Cluster.

```
# Calculate WCSS
def calculate_cost(X, centroids, cluster):
  sum = 0
  for i, val in enumerate(X):
    sum += np.sqrt((centroids[int(cluster[i]), 0]-val[0])**2 +(centroids[int(cluster[i]), 1]-val[1])**2)
  return sum
```

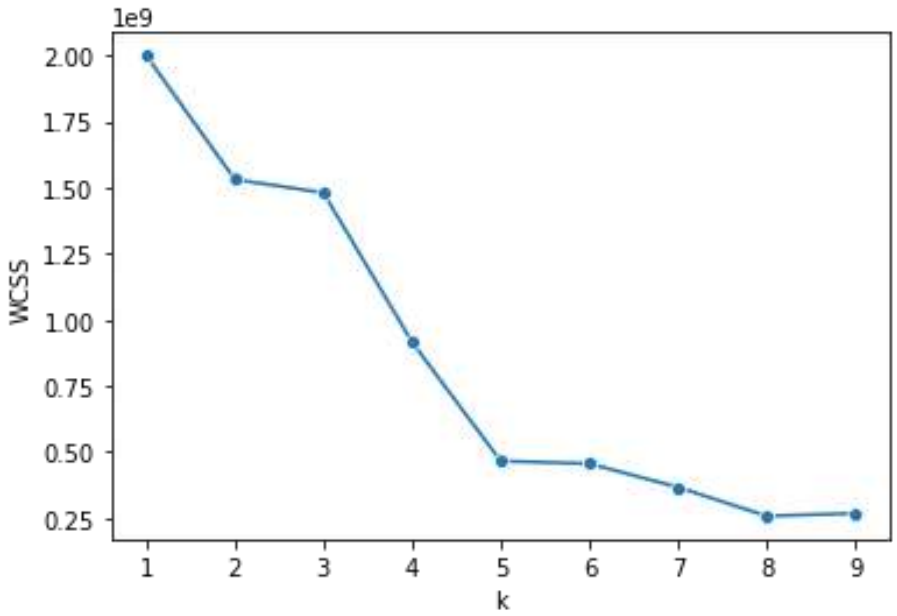
5.2. Elbow Line

Varying the number of clusters (k) with range 1-10. For each value of k, we are calculating WCSS.

```
# Find K value
cost_list = []
for k in range(1,10):
    centroids, cluster = kMeans(ds_val, k)
    cost = calculate_cost(ds_val, centroids, cluster)
    cost_list.append(cost)
```

5.3. Plot

Make the plot with similar range



D. Evaluation

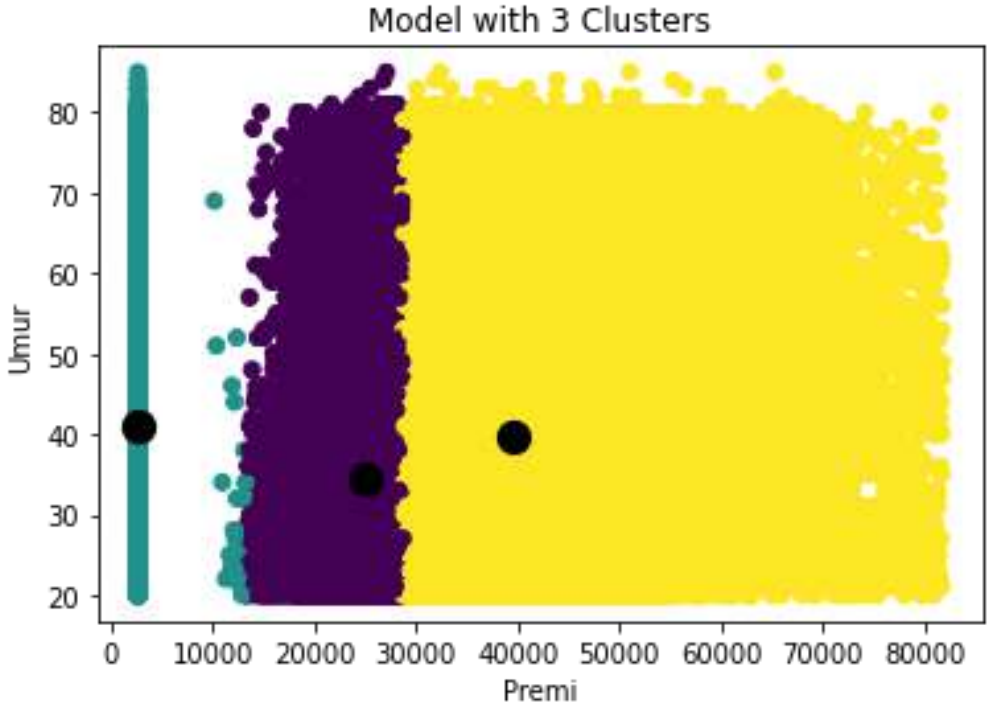
1. Silhouette Value

The evaluation is done by finding Silhouette value. This is the score with K = 2.

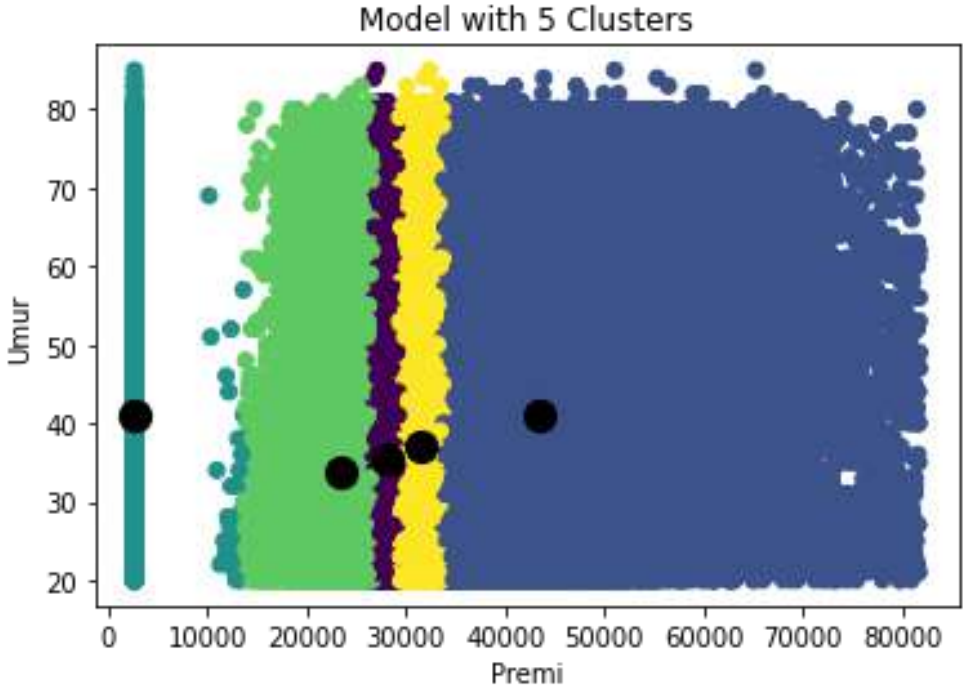
```
0.34022689799573286
```

E. Experiment

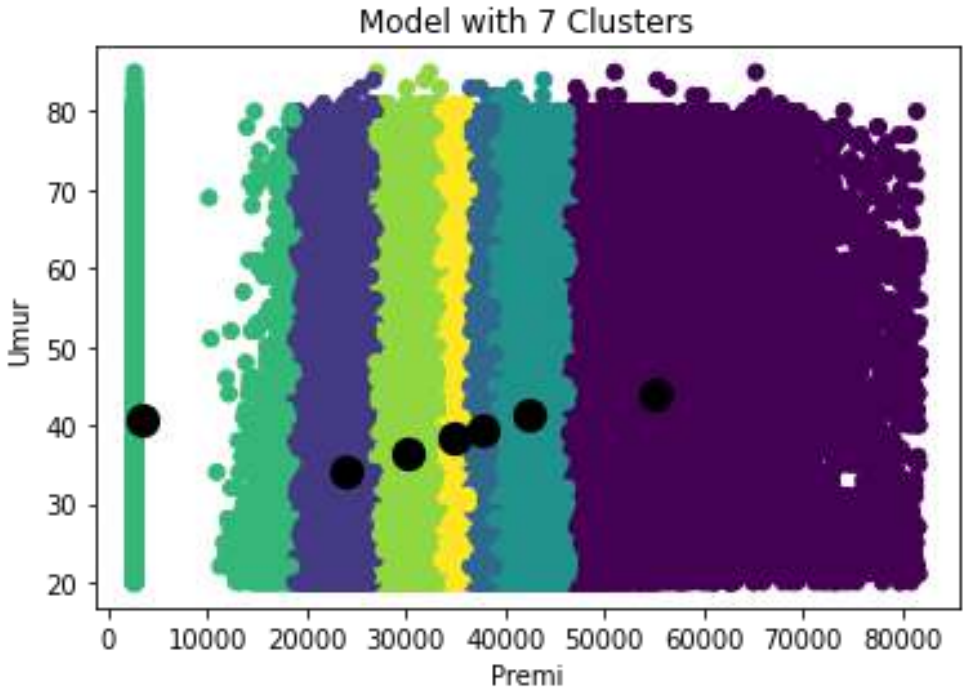
1. Model with 3 Clusters



2. Model with 5 Clusters



3. **Model with 7 Clusters**



F. Conclusion

After doing the experiment, we got the Silhouette values from the selected clusters

- Silhouette value for 2 Clusters

  ```
  0.34022689799573286
  ```

- Silhouette value for 3 Clusters

  ```
  0.40227202909311044
  ```

- Silhouette value for 5 Clusters

  ```
  0.3279705877340818
  ```

- Silhouette value for 7 Clusters

  ```
  0.09479339380624253
  ```

Based on Silhouette method that a value close to 1 is the most optimal. We can conclude that 3 clusters have the most optimal value with K = 3.

YouTube: https://youtu.be/G5h1mDP3JMw