LAPORAN TUGAS BESAR MACHINE LEARNING

CLARISA HASYA YUTIKA | 1301174256 | IF 41 02

1. PROBLEM & SOLUTION

1.1. Articulate Problem

- **CLUSTERING**

  Algoritma yang akan digunakan pada clustering adalah K-Means. Atribut yang akan digunakan adalah ‘latitude’ dan ‘longitude’, dari proses clustering ini akan menghasilkan lokasi yang terdekat yang dikelompokkan sebanyak 4 kelompok. Proses yang akan dilakukan ada preprocessing data, clustering dengan 2 model, dan evaluasi.

- **CLASSIFICATION**

  Algoritma yang akan digunakan pada classification adalah Naïve Bayes, k-Nearest Neighbor, dan Decision Tree. Atribut yang akan digunakan adalah ‘latitude’, ‘longitude’, ‘neighbourhood’, dan kelas yang akan digunakan adalah ‘neighbourhood_group’. Proses yang akan dilakukan ada preprocessing data, classification dengan 2 model, dan evaluasi.

1.2. Identify Data Sources

Data yang digunakan adalah air_bnb, memiliki 22552 record data, 16 atribut.

1.3. Identify Potential Learning Problems

Data yang diperoleh, tidak semuanya berhubungan. Hanya beberapa atribut yang berhubungan. Contohnya ‘latitude’, ‘longitude’, ‘neighbourhood’, dan ‘neighbourhood_group’. Dari atribut yang digunakan tidak terdapat missing value, terdapat outlier pada atribut ‘latitude’, dan ‘longitude’.

1.4. Potential Bias and Ethics

Dalam data ini tidak ada data yang dapat menyinggung suatu kelompok.

2. PREPROCESSING

2.1. Data Exploration

Pada tahap data exploration akan dilakukan pengecekan data, apakah terdapat missing value, outlier, dsb. Data yang digunakan adalah airbnb. Jumlah record airbnb adalah sebanyak 22552 record data.

- Mengambil data dari file 'air_bnb.csv' dan ditampung oleh variable 'df'

```
df = pd.read_csv('air_bnb.csv')
```

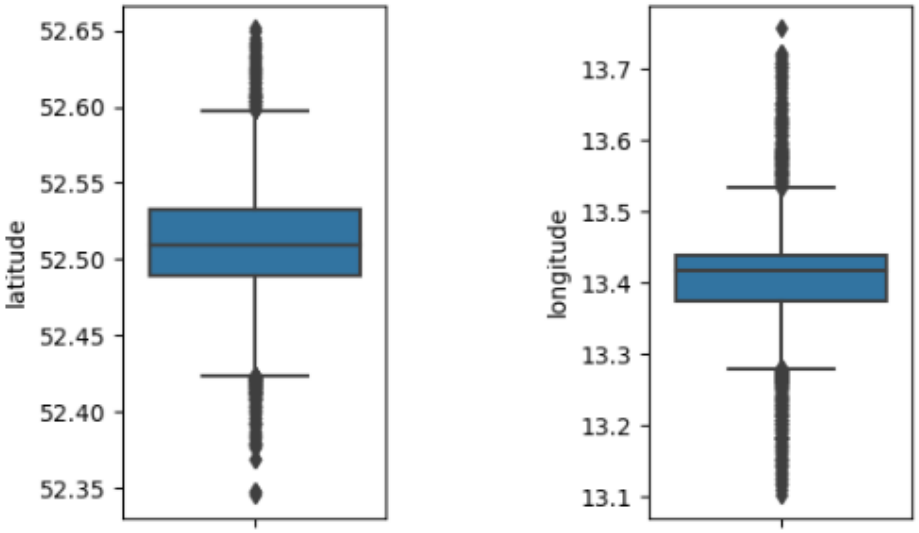
- Cek informasi data untuk melihat ada missing value atau tidak. Berdasarkan atribut yang akan dipilih, yaitu 'latitude', 'longitude', 'neighbourhood', dan 'neighbourhood_group' tidak ada missing value

```
#check missing value
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22552 entries, 0 to 22551
Data columns (total 16 columns):
id                                22552 non-null int64
name                              22493 non-null object
host_id                           22552 non-null int64
host_name                         22526 non-null object
neighbourhood_group               22552 non-null object
neighbourhood                     22552 non-null object
latitude                          22552 non-null float64
longitude                         22552 non-null float64
room_type                         22552 non-null object
price                             22552 non-null int64
minimum_nights                    22552 non-null int64
number_of_reviews                 22552 non-null int64
last_review                       18644 non-null object
reviews_per_month                 18638 non-null float64
calculated_host_listings_count    22552 non-null int64
availability_365                  22552 non-null int64
dtypes: float64(3), int64(7), object(6)
memory usage: 2.8+ MB
```

- Kemudian buat boxplot untuk melihat ada outlier atau tidak. Hanya melihat untuk atribut yang akan dipakai saat clustering dan classification, yaitu ‘latitude’, dan ‘longitude’. Untuk ‘neighbourhood’, dan ‘neighbourhood_group’ tipe data nya adalah ordinal, sehingga perlu proses encode terlebih dahulu

```
#check outlier
f, axes = plt.subplots(1, 2)
sns.boxplot(y=df["latitude"], ax=axes[0])
sns.boxplot(y=df["longitude"], ax=axes[1])
plt.subplots_adjust(wspace=1)
```



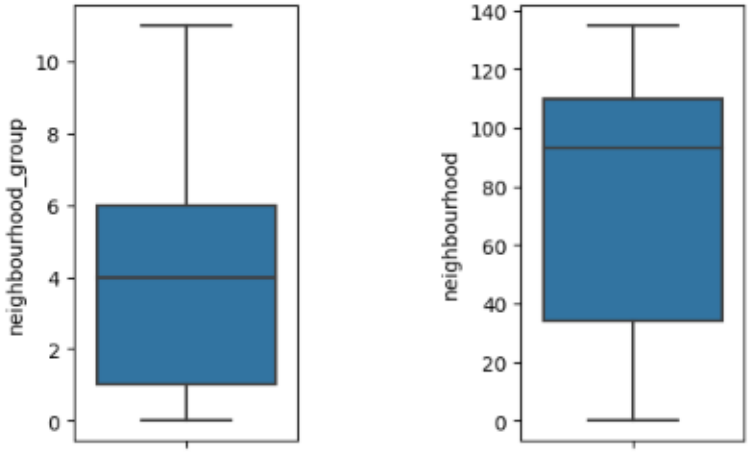
- Proses encode atribut neighbourhood’, dan ‘neighbourhood_group’ dari tipe data ordinal menjadi numerical

```
#encode from ordinal type to numerical type
from sklearn import preprocessing

encode = preprocessing.LabelEncoder()
clas['neighbourhood_group'] = encode.fit_transform(clas['neighbourhood_group'])
clas['neighbourhood'] = encode.fit_transform(clas['neighbourhood'])

clas.head()
```

	neighbourhood_group	neighbourhood	latitude	longitude
0	4	18	52.534537	13.402557
1	6	95	52.548513	13.404553
2	6	98	52.534996	13.417579
3	10	110	52.498855	13.349065
4	6	49	52.543157	13.415091

- Kemudian cek apakah ada outlier dari atribut ‘neighbourhood’, dan ‘neighbourhood_group’

```
#check outlier after encode
f, axes = plt.subplots(1, 2)
sns.boxplot(y=clas['neighbourhood_group'], ax=axes[0])
sns.boxplot(y=clas['neighbourhood'], ax=axes[1])
plt.subplots_adjust(wspace=1)
```



2.2. Data Cleansing

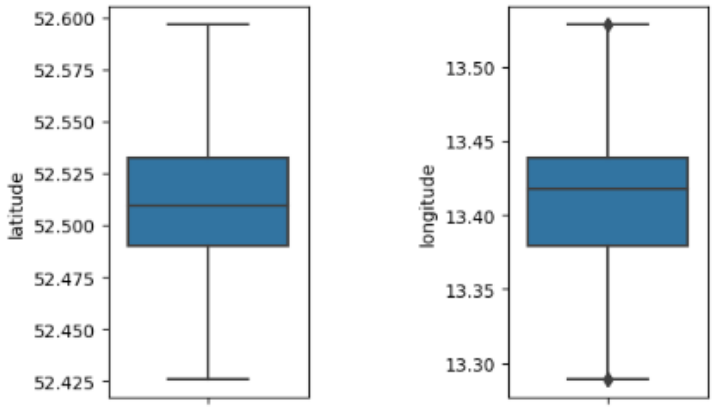
Pada tahap data cleansing akan menghandle data yang outlier dengan cara menghapus data

- Handle outlier dengan menghapus data atau diisi dengan nilai median sampai tidak ada lagi data yang outlier. Dipilihnya teknik menghapus data karena performansi nya lebih bagus dibandingkan dengan diisi dengan nilai median. Kemudian ketika diisi dengan nilai median semakin banyak data yang terkena outlier di setiap perulangannya.

```
#handle outlier
#longitude
#for model 2 clustering

while True:
  qlo1, qlo3 = np.percentile(clusterr['longitude'],[25,75])
  iqrlo = qlo3 - qlo1
  lowerlo = qlo1 - (1.5 * iqrlo)
  upperlo = qlo3 + (1.5 * iqrlo)
  outlierlo = clusterr[(clusterr['longitude'] < (lowerlo)) | (clusterr['longitude'] > (upperlo))]
  print('amount of outlier data',outlierlo.shape[0]) #amount of outlier data
  idxlo = outlierlo.index
  # midlo = np.median(clusterr['longitude'])
  # clusterr.loc[idxlo,'longitude'] = midlo #impute with median
  clusterr.drop(idxlo, inplace=True) #drop outlier data
  if (outlierlo.shape[0] <= 0):
    break
```

- Hasil boxplot setelah handle outlier

```
#check outlier after handle it
f, axes = plt.subplots(1, 2)
sns.boxplot(y=clusterr["latitude"], ax=axes[0])
sns.boxplot(y=clusterr["longitude"], ax=axes[1])
plt.subplots_adjust(wspace=1)
```



2.3. Feature Engineering (Scalling)

Pada tahap scalling akan menggunakan min max normalization dan z score normalization. Scalling digunakan hanya untuk task clustering.

- Scalling data 'cluster' untuk model 1 menggunakan min max normalization. Min max normalization akan merubah data menjadi range (0, 1). Dipilihnya teknik scalling dengan min max normalization karena dari hasil beberapa penelitian menghasilkan bahwa min max normalization lebih bagus dibandingkan teknik normalisasi yang lainnya.

```
#min max normalization

from sklearn.preprocessing import MinMaxScaler

minmax = MinMaxScaler()
cluster = minmax.fit_transform(cluster)
cluster = pd.DataFrame(cluster, columns=['latitude','longitude'])
cluster.head()
```

	latitude	longitude
0	0.617046	0.457127
1	0.662737	0.460179
2	0.618546	0.480093
3	0.500386	0.375345
4	0.645228	0.476290

- Scalling data ‘clusterr’ untuk model 2 menggunakan zscore normalization. Zscore normalization akan merubah data menjadi range (-1.5, 1.5). Dipilihnya teknik scalling dengan zscore normalization dikarenakan ingin membandingkan nilai performansi dengan min max normalization, apakah min max normalization lebih baik dibandingkan dengan zscore normalization.

```
#z score normalization
from scipy import stats

clusterr = stats.zscore(clusterr)

clusterr = pd.DataFrame(clusterr, columns=['latitude','longitude'])
clusterr.head()
```

	latitude	longitude
0	0.848461	-0.106518
1	1.349018	-0.063560
2	0.864896	0.216797
3	-0.429570	-1.257845
4	1.157200	0.163257

2.4. Data Splitting

Pada tahap data splitting, akan mengambil atribut yang digunakan untuk proses clustering dan classification. Kemudian akan membagi data menjadi data train sebesar 80% dan data test sebesar 20% dan digunakan untuk proses classification.

- Mengambil atribut ‘latitude’, dan ‘longitude’ untuk proses clustering. Data ‘cluster’ akan digunakan untuk clustering model 1, dan data ‘clusterr’ akan digunakan untuk clustering model 2. Dipilihnya atribut ‘latitude, dan ‘longitude’ pada proses clustering dikarenakan atribut tersebut menunjukkan tata letak dari hotel, sehingga akan di clustering sebanyak 4 kelompok.

```
#pick 2 attribute for clustering
cluster = df.loc[:,'latitude':'longitude'] #model 1
clusterr = df.loc[:,'latitude':'longitude'] #model2
cluster.head()
clusterr.head()
```

	latitude	longitude
0	52.534537	13.402557
1	52.548513	13.404553
2	52.534996	13.417579
3	52.498855	13.349065
4	52.543157	13.415091

- Mengambil atribut ‘latitude’, ‘longitude’, ‘neighbourhood’, dan ‘neighbourhood_group’ untuk proses classification. Dipilihnya atribut ‘latitude’, ‘longitude’, ‘neighbourhood’, dan ‘neighbourhood_group’, dikarenakan atribut ‘neighbourhood_group’ merupakan nama kota dari hotel-hotel yang termasuk dalam suatu daerah, sehingga akan diklasifikasikan berdasarkan atribut yang dipilih.

```
#pick 4 attribute for classification
clas = df.loc[:,'neighbourhood_group':'longitude']
clas.head()
```

	neighbourhood_group	neighbourhood	latitude	longitude
0	Mitte	Brunnenstr. Süd	52.534537	13.402557
1	Pankow	Prenzlauer Berg Nordwest	52.548513	13.404553
2	Pankow	Prenzlauer Berg Südwest	52.534996	13.417579
3	Tempelhof - Schöneberg	Schöneberg-Nord	52.498855	13.349065
4	Pankow	Helmholtzplatz	52.543157	13.415091

- Membagi data ‘clas’ menjadi data train sebesar 80% dan data test sebesar 20% dan digunakan untuk proses classification.

```
#variable selection, splitting into features and class
features = clas.drop(["neighbourhood_group"], axis = 1)
claas = clas["neighbourhood_group"]
```

```
#splitting into training and testing
f_train, f_test, c_train, c_test = tr(features, claas, test_size=0.2, random_state=1) # 80% data train, 20% data test
```

3. CLUSTERING

Pada tahap clustering menggunakan algoritma K-Means. Dikarenakan algoritma K-Means salah satu algoritma unsupervised yang paling sederhana dan popular. Tujuan dari algoritma ini adalah untuk menemukan grup/kluster dalam data, dengan jumlah grup yang diwakili oleh variable K. Variabel K dapat kita tentukan sendiri. Tetapi ada cara untuk menemukan nilai K yang optimal salah satu caranya dengan Elbow Method. Kemudian cara kerja algoritma ini adalah dengan mencari titik centroid sebanyak K yang sudah ditentukan, kemudian melakukan perhitungan menggunakan Euclidean distance dan rata-rata dari setiap data untuk mengoptimalkan posisi centroid. Proses ini berhenti ketika centroid sebelumnya sudah sama atau stabil dengan centroid sesudahnya.

3.1. Pemodelan

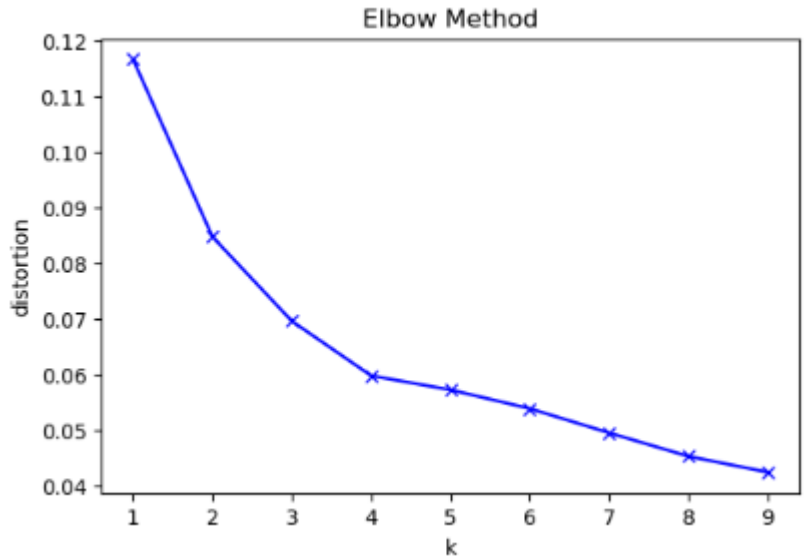
Berikut adalah langkah-langkah secara detailnya :

- Menentukan K, dengan teknik Elbow Method. Dari hasil elbow method, didapatkan K=4

```
#Search the best K using elbow method
from sklearn.cluster import KMeans #only used for elbow method
from scipy.spatial.distance import cdist

K = range(1,10)
distortions = []
for k in range(1,10):
    kmeanModel = KMeans(n_clusters=k).fit(cluster)
    kmeanModel.fit(cluster)
    distortions.append(sum(np.min(cdist(cluster, kmeanModel.cluster_centers_, 'euclidean'), axis=1)) / cluster.shape[0])
```

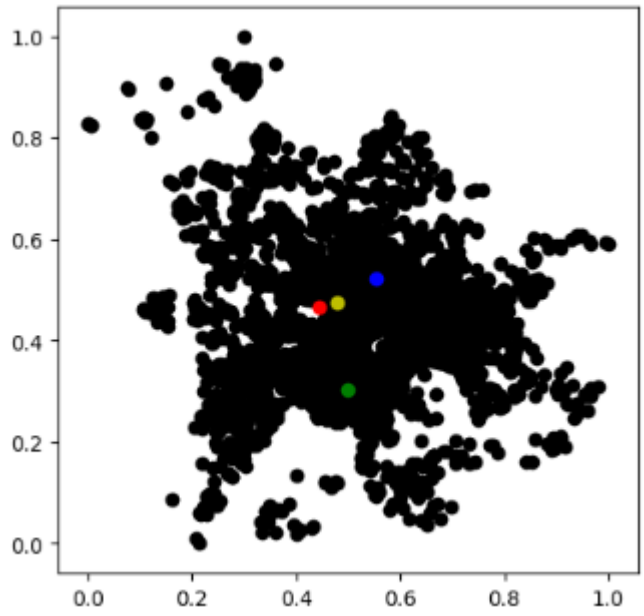
```
#plot elbow method
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('distortion')
plt.title('Elbow Method')
fig = plt.figure(figsize=(5, 5))
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



- Menentukan posisi centroid

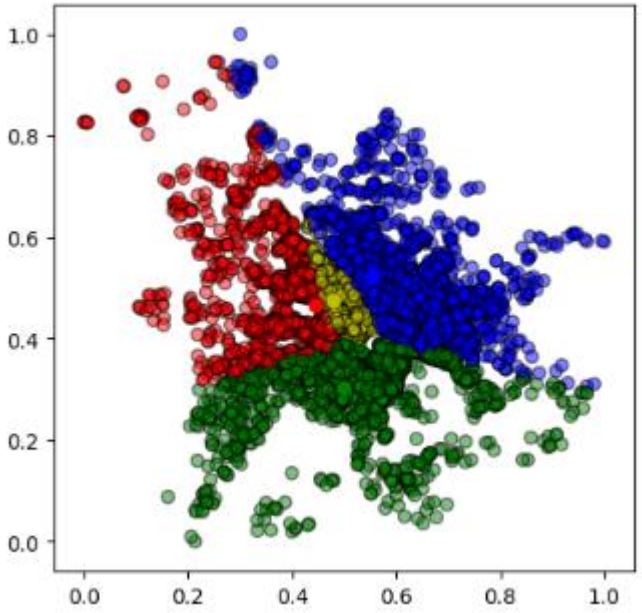
```
# Find Centroids

import random

def find_centroids(cluster):
    k = 4   #from elbow method, the best k is 4
    centroids = {
        i + 1 : [random.choices(cluster['latitude']),random.choices(cluster['longitude'])]
        for i in range(k)
    }
    return centroids
```



- Kemudian hitung jarak terdekat dari centroid terhadapt data menggunakan Euclidean distance

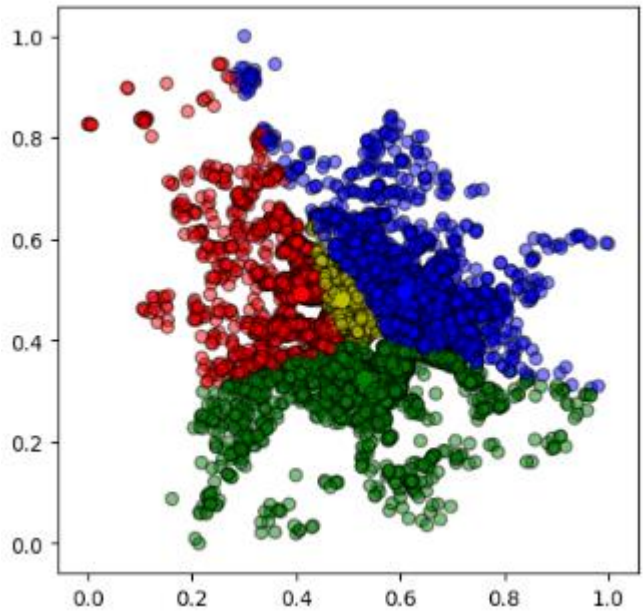
```
## Assignment Stage


def assignment(cluster, centroids):
    for i in centroids.keys():
        # sqrt((x1 - x2)^2 - (y1 - y2)^2) euclidean distance
        cluster['distance_from_{}'.format(i)] = (
            np.sqrt(
                (cluster['latitude'] - centroids[i][0]) ** 2
                + (cluster['longitude'] - centroids[i][1]) ** 2
            )
        )
    centroid_distance_cols = ['distance_from_{}'.format(i) for i in centroids.keys()]
    cluster['closest'] = cluster.loc[:, centroid_distance_cols].idxmin(axis=1)
    cluster['closest'] = cluster['closest'].map(lambda x: int(x.lstrip('distance_from_')))
    cluster['color'] = cluster['closest'].map(lambda x: colmap[x])
    return cluster

cluster = assignment(cluster, centroids)
cluster.head()
```

	latitude	longitude	distance_from_1	distance_from_2	distance_from_3	distance_from_4	closest	color
0	0.617046	0.457127	0.173016	0.194630	0.090780	0.139270	3	b
1	0.662737	0.460179	0.218587	0.227714	0.125523	0.184381	3	b
2	0.618546	0.480093	0.175021	0.214053	0.077375	0.139626	3	b
3	0.500386	0.375345	0.105600	0.071748	0.155849	0.102431	2	g
4	0.645228	0.476290	0.201360	0.227313	0.102498	0.166234	3	b



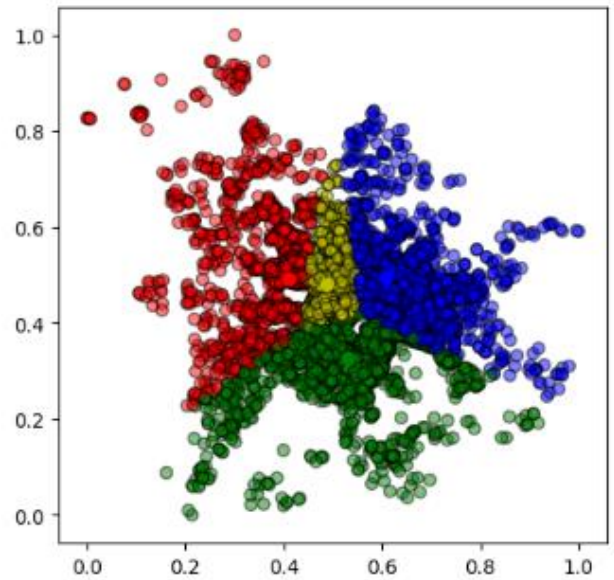
- Kemudian update titik centroid dengan merata-ratakan jarak dari data yang termasuk wilayah centroid tersebut

```
def update(k):
    for i in centroids.keys():
        centroids[i][0] = np.mean(cluster[cluster['closest'] == i]['latitude'])
        centroids[i][1] = np.mean(cluster[cluster['closest'] == i]['longitude'])
    return k

centroids = update(centroids)
```



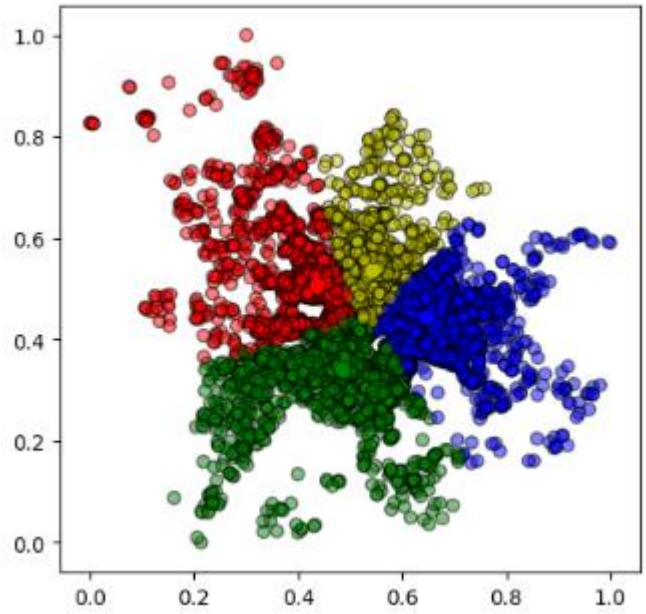
- Kemudian hitung jarak terdekat lagi dari titik centroid terbaru terhadap data

```
## Repeat Assigment Stage

cluster = assignment(cluster, centroids)
```



- Kemudian ulangi proses update titik centroid dan menyesuaikan titik centroid baru dengan datanya, sampai posisi centroid stabil, posisi centroid sebelumnya sama dengan posisi centroid sesudahnya

```
# Continue until all assigned categories don't change any more

while True:
    closest_centroids = cluster['closest'].copy(deep=True)
    centroids = update(centroids)
    cluster = assignment(cluster, centroids)
    if closest_centroids.equals(cluster['closest']):
        break
```



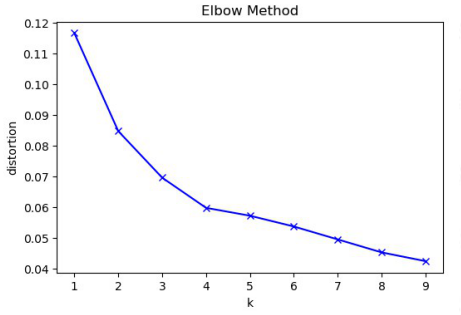
- Hasil evaluasi berdasarkan nilai SSE (Sum-of-Squared Error). Semakin kecil nilai SSE tersebut maka lebih bagus hasilnya. Dipilihnya teknik evaluasi dengan SSE dikarenakan sering digunakan oleh beberapa penelitian, sehingga hasilnya akan akurat.

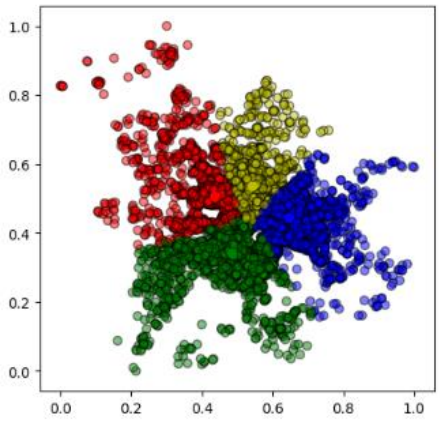
```
#evaluation based on SSE

print('k=4',distortions[3])
print('k=5',distortions[4])
```

```
k=4 0.0597194541915578
k=5 0.05643011892853079
```

3.2. Eksperimen

- **Not handle outlier, Min Max Normalization**



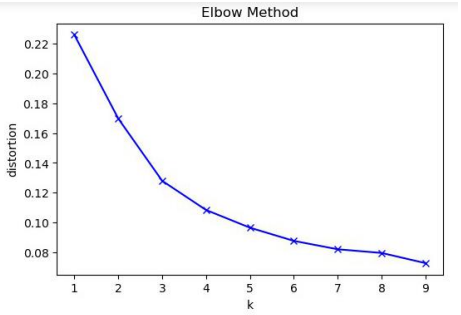


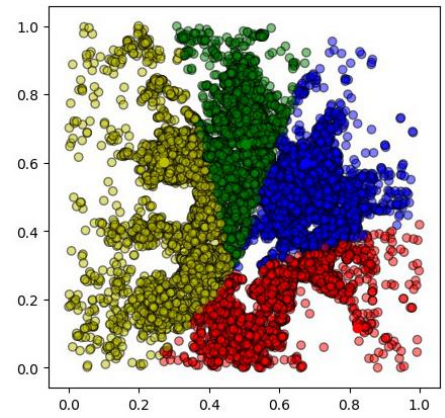
```
<Figure size 500x500 with 0 Axes>
```

```
print('k=4',distortions[3])
print('k=5',distortions[4])
```

```
k=4 0.059741315556201484
k=5 0.05718266963657246
```

- **Handle outlier by drop data, Min Max Normalization**



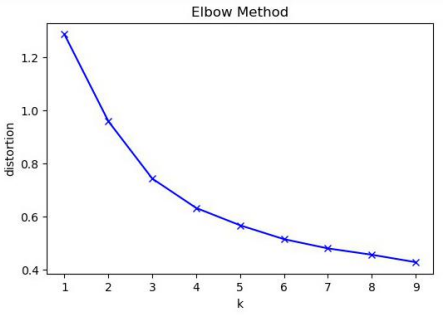


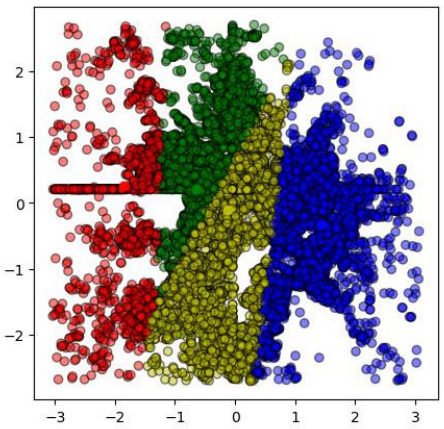
```
<Figure size 500x500 with 0 Axes>
```

```
print('k=4',distortionss[3])
print('k=5',distortionss[4])
```

```
k=4 0.10842164856485587
k=5 0.09655512949888066
```

- **Handle outlier by impute with median, Z Score Normalization**





```
<Figure size 500x500 with 0 Axes>
```

```
print('k=4',distortionss[3])
print('k=5',distortionss[4])
```

```
k=4 0.6316627602325613
k=5 0.5663011786256165
```

3.3. Kesimpulan

Dari hasil 3 eksperimen yang dilakukan, berdasarkan nilai SSE (Sum-of-Squared Error) di K=4 sebesar 5.9%, eksperimen yang bagus adalah tidak di handle outlier, menggunakan min max normalization.

4. CLASSIFICATION

Pada tahap classification, akan membandingkan performansi algoritma Naïve Bayes, Decision Tree, dan k-Nearest Neighbor, dengan teknik preprocessing tidak di handle outlier, encode atribut 'neighbourhood', dan 'neighbourhood_group' dari tipe data ordinal menjadi tipe data numerical.

4.1. Pemodelan

- Import library sklearn untuk split data, klasifikasi Naïve Bayes, Decision Tree, dan k-Nearest Neighbor, dan untuk evaluasi menggunakan F1-Score, Accuracy, Precision, Recall.

```
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split as tr
from sklearn.metrics import f1_score as f1
from sklearn.metrics import accuracy_score as acc
from sklearn.metrics import precision_score as pr
from sklearn.metrics import recall_score as rec
```

- Memisahkan data menjadi data fitur, dan data kelas.

```
#variable selection, splitting into features and class
features = clas.drop(["neighbourhood_group"], axis = 1)
claas = clas["neighbourhood_group"]
```

```
features.head()
```

	neighbourhood	latitude	longitude
0	18	52.534537	13.402557
1	95	52.548513	13.404553
2	98	52.534996	13.417579
3	110	52.498855	13.349065
4	49	52.543157	13.415091

```
claas.head()
```

```
0     4
1     6
2     6
3    10
4     6
Name: neighbourhood_group, dtype: int64
```

- Memisahkan data menjadi data train sebesar 80%, dan data test sebesar 20%.

```
#splitting into training and testing
f_train, f_test, c_train, c_test = tr(features, claas, test_size=0.2, random_state=1) # 80% data train, 20% data test
```

```
f_train.head()
```

	neighbourhood	latitude	longitude
13916	73	52.535735	13.357130
14076	135	52.499286	13.445806
9956	35	52.499444	13.510699
9612	100	52.511179	13.383790
16401	91	52.561302	13.399581

```
f_test.head()
```

	neighbourhood	latitude	longitude
16455	51	52.508308	13.312476
21435	110	52.492695	13.360330
8155	87	52.573421	13.356127
9009	135	52.493702	13.431392
10801	115	52.503983	13.414387

```
c_train.head()
```

```
13916    4
14076    1
9956     2
9612     4
16401    6
Name: neighbourhood_group,
```

```
c_test.head()
```

```
16455     0
21435    10
8155      7
9009      1
10801     1
Name: neighbourhood_group,
```

- Membuat pemodelan dengan ‘Gaussian Classifier’, ‘Decision Tree Classifier’, dan ‘KNeighbors Classifier’.

```
#Create a Gaussian Classifier
naive = GaussianNB()
```

```
# Create Decision Tree classifer object
dt = DecisionTreeClassifier(criterion="entropy", max_depth=4)
```

```
#Create a KNeighbors Classifier
knn = KNeighborsClassifier(n_neighbors=5)
```

- Latih model yang sudah dibuat dengan data train.

```
# Train the model using the training sets
naive.fit(f_train, c_train)
```

```
# Train Decision Tree Classifer
dt = dt.fit(f_train,c_train)
```

```
# Train the model using the training sets
knn.fit(f_train,c_train)
```

- Implementasikan model yang sudah dilatih dengan data test.

```
#Predict Output
cn_predict = naive.predict(f_test)
print("Class Prediction\n")
for i in range(len(cn_predict)):
    print(cn_predict[i])
```

```
4
6
0
```

```
#Predict the response for test dataset
cd_predict = dt.predict(f_test)
print("Class Prediction\n")
for i in range(len(cd_predict)):
    print(cd_predict[i])
```

```
5
5
5
```

```
#Predict Output
ck_predict = knn.predict(f_test)
print("Class Prediction\n")
for i in range(len(ck_predict)):
    print(ck_predict[i])
```

```
4
6
0
```

- Evaluasi hasil implementasi dengan F1-Score, Accuracy, Precision, dan Recall. Dipilihnya teknik evaluasi ini dikarenakan salah satu teknik evaluasi yang sering digunakan di berbagai penelitian, dan mudah dipahami.

```
# Accuracy score
print("NAIVE BAYES\n")

print("F1-SCORE ",f1(c_test,cn_predict,average='macro'))

print("ACCURACY ",acc(c_test,cn_predict))

print("PRECISION ",pr(c_test,cn_predict,average='macro'))

print("RECALL",rec(c_test,cn_predict,average='macro'))
```

```
NAIVE BAYES

F1-SCORE  0.8436087528171469
ACCURACY  0.8940367989359344
PRECISION  0.8753170302119303
RECALL 0.8253868233628489
```

```
# Accuracy score
print("DECISION TREE\n")

print("F1-SCORE ",f1(c_test,cd_predict,average='macro'))

print("ACCURACY ",acc(c_test,cd_predict))

print("PRECISION ",pr(c_test,cd_predict,average='macro'))

print("RECALL",rec(c_test,cd_predict,average='macro'))
```

```
DECISION TREE

F1-SCORE  0.784082517135457
ACCURACY  0.8618931500775882
PRECISION  0.8449413312062246
RECALL 0.7627049805707324
```

```
# Accuracy score
print("K NEAREST NEIGHBOR\n")

print("F1-SCORE ",f1(c_test,ck_predict,average='macro'))

print("ACCURACY ",acc(c_test,ck_predict))

print("PRECISION ",pr(c_test,ck_predict,average='macro'))

print("RECALL",rec(c_test,ck_predict,average='macro'))
```

```
K NEAREST NEIGHBOR

F1-SCORE  0.9991527808363307
ACCURACY  0.9995566393260917
PRECISION  0.9997678737233056
RECALL 0.9985507246376811
```

4.2. Eksperimen

- **Naïve Bayes**

```
# Accuracy score
print("NAIVE BAYES\n")

print("F1-SCORE ",f1(c_test,cn_predict,average='macro'))

print("ACCURACY ",acc(c_test,cn_predict))

print("PRECISION ",pr(c_test,cn_predict,average='macro'))

print("RECALL",rec(c_test,cn_predict,average='macro'))
```

```
NAIVE BAYES

F1-SCORE  0.8436087528171469
ACCURACY  0.8940367989359344
PRECISION  0.8753170302119303
RECALL 0.8253868233628489
```

- **Decision Tree**

```
# Accuracy score
print("DECISION TREE\n")

print("F1-SCORE ",f1(c_test,cd_predict,average='macro'))

print("ACCURACY ",acc(c_test,cd_predict))

print("PRECISION ",pr(c_test,cd_predict,average='macro'))

print("RECALL",rec(c_test,cd_predict,average='macro'))
```

```
DECISION TREE

F1-SCORE  0.784082517135457
ACCURACY  0.8618931500775882
PRECISION  0.8449413312062246
RECALL 0.7627049805707324
```

- **K-Nearest Neighbor**

```
# Accuracy score
print("K NEAREST NEIGHBOR\n")

print("F1-SCORE ",f1(c_test,ck_predict,average='macro'))

print("ACCURACY ",acc(c_test,ck_predict))

print("PRECISION ",pr(c_test,ck_predict,average='macro'))

print("RECALL",rec(c_test,ck_predict,average='macro'))
```

```
K NEAREST NEIGHBOR

F1-SCORE  0.9991527808363307
ACCURACY  0.9995566393260917
PRECISION  0.9997678737233056
RECALL 0.9985507246376811
```

4.3. Kesimpulan

Dari hasil 3 eksperimen yang dilakukan, berdasarkan nilai F1-Score, Accuracy, Precision, dan Recall didapatkan algoritma terbaik yaitu k-Nearest Neighbor dengan nilai F1-Score sebesar 99.9%, Accuracy sebesar 99.9%, Precision sebesar 99.9%, dan Recall sebesar 99.9%. Dikarenakan k-Nearest Neighbor mencari data yang terdekat disekitarnya, dan tujuan dari klasifikasi ini adalah mencari kota-kota berdasarkan tata letak suatu hotel dan kelompok daerahnya. Sehingga k-Nearest Neighbor menghasilkan nilai performansi tertinggi.

REFERENSI

https://youtu.be/1XqG0kaJVHY

https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn

https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

https://www.datacamp.com/community/tutorials/decision-tree-classification-python