Pengelompokan Tingkat Kejahatan pada Daerah di *United Kingdom* Berdasarkan Data Kejahatan Kepolisian United Kingdom

Pengelolaan Data Besar

1406544135	**Gitta Gracia Evelyn Diva**
1406557535	**Luthfi Abdurrahim**
1406623505	**M Hawari Hilman**
1406575815	**Mahdi Firdaus**
1406579132	**Rohmat Taufik**

PROGRAM SISTEM INFORMASI

FAKULTAS ILMU KOMPUTER

DEPOK

NOVEMBER 2017

ABSTRAK

Laporan ini berisikan hasil analisa kejahatan yang berada di United Kingdom. Tim penulis mendapatkan data dari sumber data pada laman *policeuk-data.s3.amazonaws.com/archive/2017-08.zip.* Metodologi yang digunakan terdiri dari beberapa langkah yaitu pencarian data, persiapan data, persiapan *environment*, eksperimen, dan analisis hasil eksperimen. Pengolahan data dilakukan dengan menggunakan metode CRISP - DM, yang tediri dari enam tahapan yaitu *business understanding, data understanding, data preparation, modelling, evaluation* dan *deployment*. Infrastruktur yang digunakan dalam pengerjaan proyek adalah Spark. Proyek ini dikerjakan dalam lima minggu dengan keluaran berupa *dashboard*. *Dashboard* tersebut akan menampilkan rekomendasi lokasi yang sebaiknya ditingkatkan keamanannya oleh polisi United Kingdom.

Kata Kunci : *Big Data, CRISP-DM, Hadoop, Spark*

PENDAHULUAN

Pengelolaan Data Besar merupakan salah satu mata kuliah pilihan di Fakultas Ilmu Komputer Universitas Indonesia. Mata kuliah tersebut membahas mengenai data beserta hal - hal yang berkaitan dengannya terlebih infrastruktur yang dapat digunakan untuk mengolah data. Seperti tersurat dalam namanya, bahwa data besar merupakan data yang berukuran besar. Perlu penanganan berbeda untuk pengelolaan data besar dibandingkan data biasa. Tujuan yang akan dicapai dari pengelolaan data besar adalah tergambarkannya pengetahuan yang dapat diperoleh dari data yang bersangkutan.

Infrastruktur dibangun dengan karakteristik masing - masing. Terdapat tujuan tertentu dengan dibangunnya infrastruktur tersebut. Sebagai contoh adalah infrastruktur Hadoop yang dibangun dengan tujuan untuk pengelolaan data besar atau infrastruktur GPU yang dibangun untuk mengelola proses yang rumit. Setiap kasus diselesaikan dengan infrastruktur yang cocok dengan karakteristik kasus tersebut. Sangat mungkin apabila satu kasus dengan kasus lain yang karakteristiknya berbeda, ditangani dengan infrastruktur yang berbeda pula.

Proyek akhir mata kuliah memfasilitasi mahasiswa untuk memperdalam pengelolaan data besar. Mahasiswa diminta untuk menganalisis data besar dan mengambil pengetahuan dari data besar tersebut. Dalam menganalisisnya, tentu mahasiswa harus mengetahui tentang data yang dihadapi dan infrastruktur yang tepat dalam mengelola data dan mencari pengetahuan dari data tersebut. Penggunaan infrastruktur yang tepat sangat penting dalam menentukan performa dari proses pengerjaan maupun hasil yang didapatkan.

Laporan ini berisikan hasil analisis data besar mengenai kejahatan yang berada di United Kingdom. Data diperoleh dari *policeuk-data.s3.amazonaws.com/archive/2017-08.zip* yang merupakan data kejahatan yang terjadi di United Kingdom. Dari data tersebut tim penulis melakukan pengelompokkan kejahatan pada berbagai daerah di United Kingdom berdasarkan lokasi kejahatan.

METODOLOGI

Penambangan data memiliki berbagai metodologi yang dapat digunakan untuk mempermudah prosesnya. Akan tetapi, pada perlaksanaannya terdapat dua pilihan metode yang paling sering digunakan, yaitu *Cross-Industry Standard Process for Data Mining* (CRISP-DM) dan *Sample, Explore, Modify, Model, and Assess* (SEMMA). Pada proyek ini tim penulis lebih memilih menggunakan metodologi CRISP-DM karena berdasarkan informasi yang diperolehi, penggunaan SEMMA lebih difokuskan untuk pengembangan model dari proses penambangan data. Selain itu, metode CRISP-DM dapat lebih fleksibel untuk berbagai bisnis, termasuk keamanan.

Metodologi CRISP-DM memiliki enam tahapan dalam melakukan penambangan data, yaitu:

1. *Business Understanding*

   Pada tahapan ini, dilakukan analisa terhadap keamanan yang ada di UK. Setelah mendapatkan hasil analisa tersebut, dilakukan identifikasi masalah yang dapat diselesaikan dengan menggunakan penambangan data.
2. *Data Understanding*

   Pada tahapan ini, dilakukan pengumpulan data terkait kriminalitas yang terjadi di UK. Selain itu, dilakukan juga identifikasi kualitas dan informasi implisit yang mungkin terdapat pada data tersebut.
3. *Data Preparation*

   Tahapan ini adalah tahapan paling riskan dalam penambangan data. Hal ini dikarenakan pada tahapan inilah data diubah menjadi bentuk yang sesuai dengan kebutuhan penambangan data. Adapun tahapan dari *data preparation* adalah sebagai berikut:

   a. *Data Consolidation*

      Integrasi data jika terdapat data yang tersebar ke berbagai *record*/tabel.

   b. *Data Cleaning*

      Pembersihan terhadap *record* data yang terdapat nilai kosong, inkonsistensi, serta *record* data yang *meaningless*.

   c. *Data Transformation*

      Pengubahan data dengan normalisasi, melakukan *discrete*/*aggregate* data, serta membuat atribut baru jika diperlukan.

   d. *Data Reduction*

      Pengurangan data yang tidak berhubungan dengan penambangan data yang akan dilakukan. Hal tersebut dilakukan dengan pengurangan *variables*, maupun dengan *balancing skewed* data.

4. *Model Building*

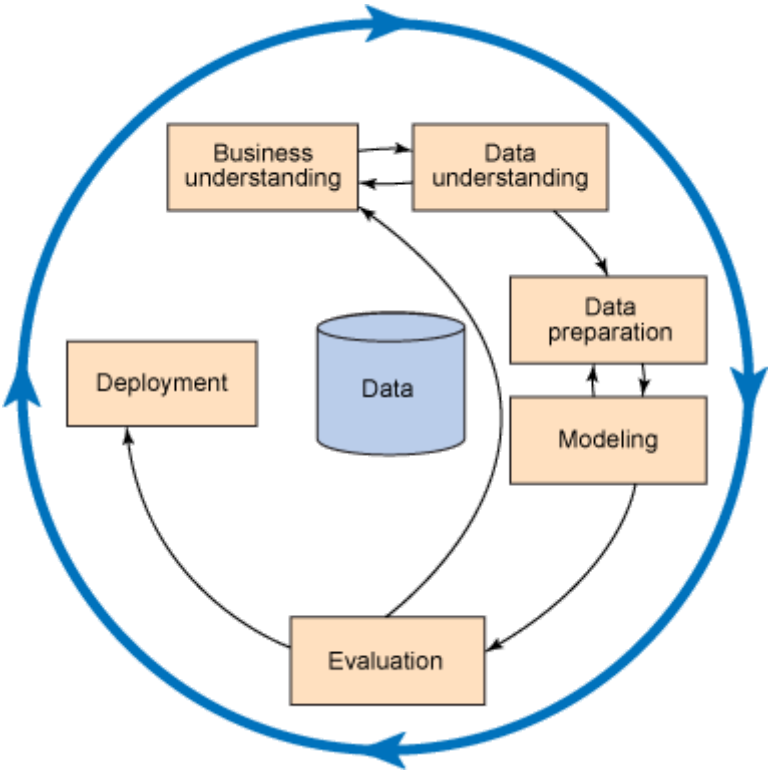
   Pada tahapan ini dilakukan pemilihan teknik modeling, seperti *association, classification, clustering, prediction, sequential patterns, decision tree*, maupun teknik modeling lainnya. Selain itu juga ditentukan parameter dalam penggunaan teknik tersebut dan pembuatan modelnya. Pada proyek ini penulis menggunakan teknik *clustering* untuk mengelompokan lokasi kejadian kriminalitas dalam satuan radius dan kurun waktu tertentu.

5. *Testing and Evaluation*

   Pada tahapan ini dilakukan evaluasi terhadap model yang telah dibuat, apakah model tersebut sudah sesuai untuk menangani permasalahan yang dipaparkan pada tahapan pertama.

6. *Deployment*

   Tahapan ini dilakukan untuk mengorganisasikan pengetahuan/informasi yang telah didapatkan dari model untuk dapat digunakan dalam menyelesaikan permasalahan pada tahapan pertama ataupun sebagai dasar dalam melakukan *decision making*.

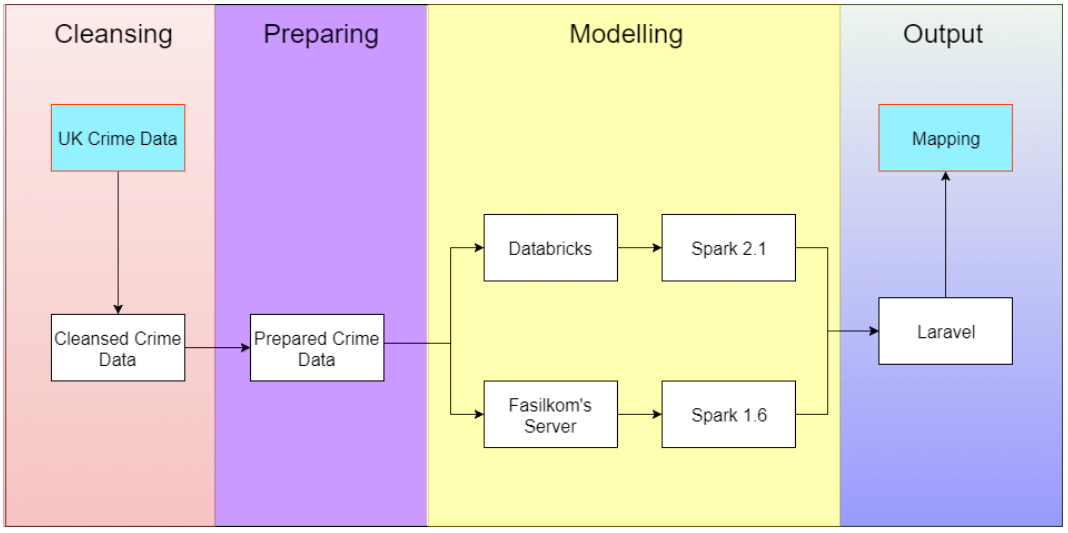


Gambar 1. CRISP-DM Methodology

(sumber:https://www.ibm.com/developerworks/bpm/library/techarticles/1407_chandran/image002.png)

RANCANGAN INFRASTRUKTUR

` Pengelolaan data yang telah kami lakukan memiliki beberapa tahap. Tahap-tahap tesebut diantaranya berupa *cleansing, modelling,* dan *output*. Berikut adalah gambar tentang tahap-tahap yang kami lakukan.



Gambar 2. Rancangan Infrastruktur
(Sumber : Olahan penulis)

Pengelolaan data yang kami lakukan dimulai dari tahap *Cleansing*. Pada tahap ini, data UK Crime Data yang dibersihkan. Pembersihan data yang kami lakukan berupa pembersihan data yang memiliki nilai kosong atau *null*, dan data yang tidak sesuai dengan tipe data yang seharusnya. Contoh value yang tidak sesuai tersebut seperti data *Integer* berisi data yang berisikan *String*. Pembersihan data ini sangat bermanfaat untuk tahap modelling agar tidak terjadi *error* dalam melakukan *modelling data*.

Tahap kedua adalah Preparing. Pada tahap ini, data yang ada ditambahkan Id, Features, dan StandardScaler. Selain itu, data dilakukan join dengan sumber data dan hasil *clustering*. Setelah semua ini selesai maka masuk ke tahap berikutnya.

Tahap ketiga adalah Modelling. Pada tahap ini, data diolah dengan menggunakan Hadoop dan Spark. Hadoop digunakan sebagai *file system* sedangkan Spark untuk memproses data yang ada. Dalam tahap ini, data diolah di dua tempat yang berbeda, yaitu server fasilkom dan Databricks. Data yang diolah pada kedua tempat tersebut merupakan data yang sama. Hal ini kami lakukan karena versi dari *spark* pada server fasilkom berbeda dengan versi pada Databricks sehingga ingin diuji coba apakah ada perbedaannya.

Tahap yang terakhir adalah tahap *Output*. Pada tahap ini, kami membentuk sebuah mapping dengan menggunakan laravel. Mapping ini dilakukan dengan menggunakan API dan *library* google maps.

ANALISIS HASIL

Secara umum, terdapat dua buah percobaan yang dilakukan yaitu dengan menggunakan server fasilkom dengan Spark versi 1.6 dan dengan menggunakan databricks.com dengan Spark versi 2.2. Digunakan pula dua buah algoritma *clustering* yaitu metode *k-means* dan *bisecting k-means*. Kedua metode tersebut telah tersedia pada *spark* di server fasilkom maupun di databricks.com. Jumlah k pada clustering ditentukan sebanyak 9 (k=9), angka 9 tersebut mengacu pada banyaknya negara bagian di United Kingdom.

A. Menggunakan Server Fasilkom

Terdapat tiga buah percobaan yang dilakukan dengan menggunakan server Fasilkom, yaitu k-means dengan spark ml, k-means dengan spark mllib, dan bisectingKMeans dengan mllib. Terdapat dua machine learning yang disediakan oleh spark, yaitu spark ml dan spark mllib. Spark mllib merupkan original API dengan menggunakan RDD, sementara spark ml menyediakan higher-level dengan menggunakan DataFrame.

Input (data) yang digunakan dalam ketiga percobaan tersebut adalah berupa csv file yang berisi longitude dan latitude. Cleansing data dilakukan untuk memperoleh input berupa longitude dan latitude tersebut. Proses cleansing dilakukan dengan menggunakan java dan dijalankan di server fasilkom menghasilkan data dengan besar 426,3 MB dan 22474278 baris. Contoh input data dapat dilihat pada Gambar 3.

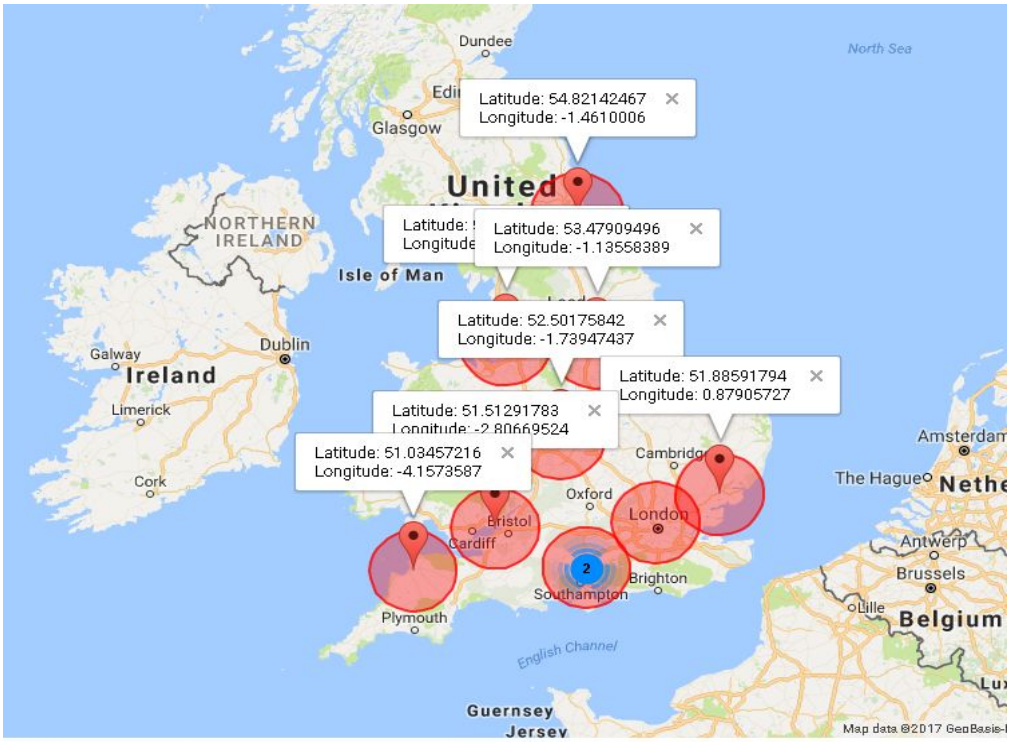
```
-2.442956,50.568566
-2.950523,50.842053
-2.956029,50.845281
-2.956029,50.845281
-2.956029,50.845281
-2.950724,50.845639
-2.902769,50.846263
-2.902769,50.846263
-2.902769,50.846263
-2.95646,50.862903
-2.96373,50.864687
-2.968617,50.868082
-2.956097,50.868734
-2.968212,50.869074
```

Gambar 3. Contoh Input Data
(Sumber : Olahan penulis)

Percobaan pertama menggunakan spark ml dan mengharapkan hasil berupa *center position* berserta *cluster size* dari setiap *cluster*. Namun, kami hanya berhasil mendapatkan *cluster center*, sementara *cluster size* tidak berhasil kami dapatkan. Menurut dokumentasi, *method cluster size* tersedia mulai versi spark 2.1.0. Gambar 4 menunjukkan code spark yang digunakan. Sementara Gambar 5 menunjukkan hasil *center position* yang disajikan dalam maps.

```
from __future__ import print_function

import sys

import numpy as np
from pyspark import SparkContext
from pyspark.mllib.clustering import KMeans


def parseVector(line):
    return np.array([float(x) for x in line.split(',')])


if __name__ == "__main__":
    sc = SparkContext(appName="KMeans")
    lines = sc.textFile("output_positions_v2.csv")
    data = lines.map(parseVector)
    k = 9

    model = KMeans.train(data, k)
    print("Final centers: " + str(model.clusterCenters))
#    print("Count : " + str(model.summary.clusterSize))
#    print("Total data: "+str(model.clusterSizes))
#    print("Total Cost: " + str(model.computeCost(data)))
    sc.stop()
```

Gambar 4. Code Spark ML K-Means
(Sumber :Olahan penulis)

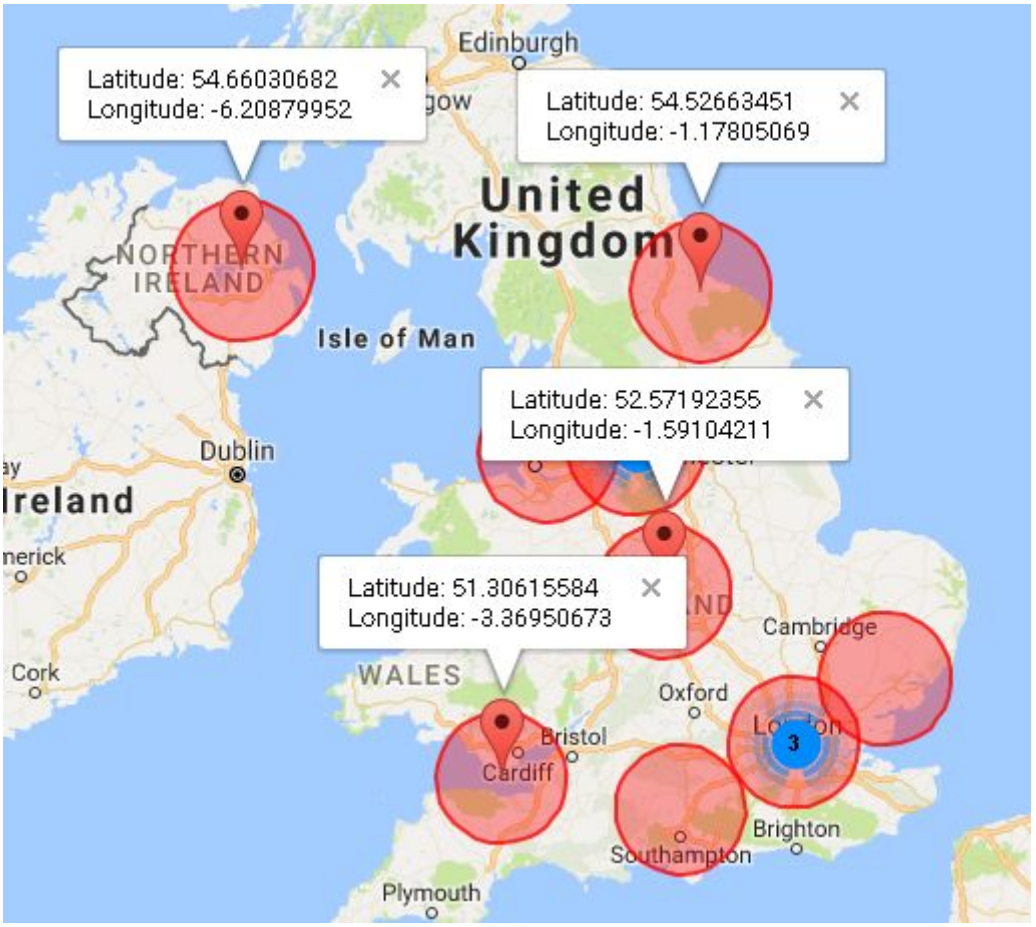


Gambar 5. Hasil Spark ML K-means
(Sumber : Olahan penulis)

Percobaan kedua adalah masih menggunakan *algoritma k-means*, namun dengan *spark mllib*. Seperti pada percobaan sebelumnya, bahwa hasil yang diharapkan adalah berupa *center positions* dan *cluster size* dari setiap *cluster*, namun hasil nyatanya tim hanya mendapatkan *cluster center* saja. *Method* untuk mendapatkan *cluster size* menurut dokumentasi terdapat pada versi spark 2.1.0. Gambar 6 menunjukkan code spark yang digunakan, sementara Gambar 7 menunjukkan hasil yang direpresentasikan dalam *maps*.

```
from numpy import array
from math import sqrt
from pyspark import SparkContext
from pyspark.mllib.clustering import KMeans, KMeansModel

# Load and parse the data
sc = SparkContext(appName="KMeans")
data = sc.textFile("output_positions_v2.csv")
parsedData = data.map(lambda line: array([float(x) for x in line.split(',')]))

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 9, maxIterations=10, initializationMode="random")

# Evaluate clustering by computing Within Set Sum of Squared Errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x + y)
print("Within Set Sum of Squared Error = " + str(WSSSE))

# Save and load model
# clusters.save(sc, "target/org/apache/spark/PythonKMeansExample/KMeansModel")
# sameModel = KMeansModel.load(sc, "target/org/apache/spark/PythonKMeansExample/KMeansModel")
centers = clusters.clusterCenters
print (centers)
```

Gambar 6. Code Spark MLLIB K-Means
(Sumber :Olahan penulis)



Gambar 7. Hasil Spark MLLIB K-means
(Sumber : Olahan penulis)

Percobaan ketiga menggunakan Spark Mllib dengan algoritma BisectingKMeans. Pada percobaan ini tim tidak mendapatkan output baik cluster center maupun cluster size. Menurut dokumentasi, untuk bisecting k means berada pada spark versi 2.0.0. Gambar 8 menunjukkan code spark yang digunakan dalam percobaan.

```
from numpy import array
from math import sqrt
from pyspark import SparkContext
from pyspark.mllib.clustering import BisectingKMeans, BisectingKMeansModel

# Load and parse the data
sc = SparkContext(appName="BisectingKMeans")
data = sc.textFile("output_positions_v2.csv")
parsedData = data.map(lambda line: array([float(x) for x in line.split(',')]))

# Build the model (cluster the data)
model = BisectingKMeans.train(parsedData, 9, maxIterations=5)

#Menncatat center
center = model.clusterCenters
print("Bisecting K-means Center = " + str(center))
```

Gambar 8. Code Bisecting K-mens Spak Mllib
(Sumber : Olahan penulis)

B. Menggunakan Databricks.com

Databricks merupakan suatu platform yang dapat mengelola *Big Data* secara *cloud computing.* Tim kami melakukan tiga percobaan dalam mengelola data kami pada databricks, antara lain:

1. KMeans dengan Standard Scaler
2. Bisecting KMeans atau Hierarchical KMeans
3. KMeans

Ketiga percobaan tersebut diterapkan dengan menggunakan library Machine Learning yang disediakan oleh Spark 2.2.

Input data yang digunakan pada ketiga percobaan tersebut ialah dengan menggunakan data yang sama dengan percobaan yang diterapkan di Server Fasilkom, yaitu berupa csv file yang berisi longitude dan latitude. Cleansing data dilakukan untuk memperoleh input berupa longitude dan latitude tersebut. Proses cleansing dilakukan dengan menggunakan java dan dijalankan di server fasilkom menghasilkan data dengan besar 426,3 MB dan 22474278 baris. Contoh input data dapat dilihat pada Gambar 3.

```
-2.442956,50.568566
-2.950523,50.842053
-2.956029,50.845281
-2.956029,50.845281
-2.956029,50.845281
-2.950724,50.845639
-2.902769,50.846263
-2.902769,50.846263
-2.902769,50.846263
-2.95646,50.862903
-2.96373,50.864687
-2.968617,50.868082
-2.956097,50.868734
-2.968212,50.869074
```

Gambar 9. Contoh Input Data
(Sumber : Olahan penulis)

Percobaan pertama, menggunakan Algoritma *Machine Learning* KMeans dengan menerapkan library Standard Scaler. Pada percobaan ini, diharapkan dapat memberikan hasil dengan inputan Longitude, Latitude, Month, Year hingga Crime Type. Karena secara teori, library Standard Scaler dapat menyamakan standard pengukuran pada masing-masing feature yang ada, sehingga dapat menghasilkan similiarity KMeans yang adil. Namun, pada saat dijalankan, menghasilkan hasil Longitude dan Latitude yang berubah maknanya. Berikut ini pada Gambar 10. merupakan code dalam menjalankan percobaan pertama dengan Standard Scaler.

Cmd 30

```
from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol='features',
                        outputCol='scaledFeatures')

scaler_model = scaler.fit(final_data3)

final_data4 = scaler_model.transform(final_data3)

kmeans99 = KMeans(featuresCol='scaledFeatures',k=9)

model99 = kmeans99.fit(final_data4)

print('WSSSE')
print(model99.computeCost(final_data4))
print('center : '+str(model99.clusterCenters()))

display(model99.transform(final_data4))
```

Gambar 10. Code KMeans dengan StandardScaler
(Sumber: Olahan Penulis)

Berdasarkan hasil code KMeans dengan StandardScaler tersebut, dihasilkan hasil clustering berikut pada Gambar 11.

Cmd 29

```
display(model99.transform(final_data4))
```

(1) Spark Jobs

_c0	_c1	features	scaledFeatures	prediction
-2.474568	51.410258	[1,2,[],[-2.474568,51.410258]]	[1,2,[],[-2.001805209881559,44.89605074937165]]	1
-2.475536	51.412124	[1,2,[],[-2.475536,51.412124]]	[1,2,[],[-2.0025882748218495,44.897680308023126]]	1
-2.486551	51.408096	[1,2,[],[-2.486551,51.408096]]	[1,2,[],[-2.0114988743231947,44.89416269695768]]	1
-2.474568	51.410258	[1,2,[],[-2.474568,51.410258]]	[1,2,[],[-2.001805209881559,44.89605074937165]]	1
-2.474568	51.410258	[1,2,[],[-2.474568,51.410258]]	[1,2,[],[-2.001805209881559,44.89605074937165]]	1
-2.483019	51.412705	[1,2,[],[-2.483019,51.412705]]	[1,2,[],[-2.0086416580327953,44.89818768936102]]	1
-2.478811	51.413145	[1,2,[],[-2.478811,51.413145]]	[1,2,[],[-2.0052375906064075,44.898571936845826]]	1
-2.478811	51.413145	[1,2,[],[-2.478811,51.413145]]	[1,2,[],[-2.0052375906064075,44.898571936845826]]	1
-2.345114	51.401467	[1,2,[],[-2.345114,51.401467]]	[1,2,[],[-1.8970832173398275,44.888373659283175]]	3
-2.345019	51.400362	[1,2,[],[-2.345019,51.400362]]	[1,2,[],[-1.8970063669582906,44.887408674122476]]	3
-2.345114	51.401467	[1,2,[],[-2.345114,51.401467]]	[1,2,[],[-1.8970832173398275,44.888373659283175]]	3
-2.349164	51.399153	[1,2,[],[-2.349164,51.399153]]	[1,2,[],[-1.9003594704474487,44.88635286682899]]	3
-2.342488	51.400225	[1,2,[],[-2.342488,51.400225]]	[1,2,[],[-1.8949589110038734,44.887289033428345]]	3
-2.350626	51.400417	[1,2,[],[-2.350626,51.400417]]	[1,2,[],[-1.9015421573717308,44.887456705057807]]	3
-2.344441	51.396119	[1,2,[],[-2.344441,51.396119]]	[1,2,[],[-1.896538793057993,44.88370330576331]]	3
-2.342774	51.40196	[1,2,[],[-2.342774,51.40196]]	[1,2,[],[-1.8951902710998683,44.888041911241116]]	3
-2.33901	51.402222	[1,2,[],[-2.33901,51.402222]]	[1,2,[],[-1.892145378088242,44.88903299303552]]	3
-2.341424	51.400219	[1,2,[],[-2.341424,51.400219]]	[1,2,[],[-1.8940981867306612,44.88728379368991]]	3

Gambar 10. Hasil Modeling KMeans dengan StandardScaler

Dan berikut ini merupakan hasil cluster centernya:

```
center :
[array([ -0.90921043,  46.69782862]),
array([ -2.77633637,  44.79043881]),
array([ -0.14561878,  45.00805172]),
array([ -1.17157262,  44.60281741]),
array([ -2.06914001,  46.69964788]),
array([ -1.41645044,  45.81523298]),
array([ -1.21409195,  47.87568   ]),
array([  0.65642843,  45.35243381]),
array([ -4.90693776,  47.62696034])]
```

Gambar 11. Hasil Cluster Center dengan KMeans dengan StandardScaler
(Sumber: Olahan Penulis)

Terlihat bahwa _c0 merupakan Longitude, _c1 merupakan Latitude, prediction merupakan hasil labeling dari clustering, lalu scaledFeatures merupakan hasil dari standard scaler, lalu ada pula cluster center yang dihasilkan. Pada cluster center, diharapkan dihasilkan Longitude dan Latitude, namun hasilnya tidak sesuai dengan Longitude dan Latitude yang ada. Sehingga, hasil dari percobaan ini tidak digunakan dan melanjutkan ke percobaan kedua pada databricks.

Percobaan kedua, yaitu menggunakan algoritma *Machine Learning* Bisecting KMeans atau istilah lainnya yaitu Hierarchical KMeans. Disebut Bisecting, karena istilah yang ada pada library spark seperti itu, namun berfungsi sebagai Hierarchical KMeans. Pada percobaan kedua ini, diharapkan hasil yang dapat mengolah inputan Longitude, Latitude, Month, Year hingga Crime Type. Karena, algoritma Bisecting KMeans ini secara teori, dapat menerapkan hal tersebut, kami menginginkan makna hasil yang dimana suatu daerah pada Longitude Latitude tertentu, terjadi Crime Type apa saja dan pada bulan atau tahun kapan saja. Namun, hasilnya tidak sesuai yang diharapkan karena menghasilkan makna cluster center pada Longitude dan Latitude yang aneh.

Berikut ini pada Gambar 12, merupakan code yang digunakan untuk menerapkan Bisecting KMeans, pada mulanya, diterapkan hanya pada Longitude dan Latitude saja.

Cmd 116

```
from numpy import array
from math import sqrt
from pyspark import SparkContext
from pyspark.mllib.clustering import BisectingKMeans, BisectingKMeansModel

# Load and parse the data
sc = SparkContext(appName="KMeans")
data = sc.textFile("/FileStore/tables/output_positions_v2.csv")
parsedData = data.map(lambda line: array([float(x) for x in line.split(',')]))

# Build the model (cluster the data)
model = BisectingKMeans.train(parsedData, 9, maxIterations=5)

#Menncatat center dll
center = model.clusterCenters
cost = model.computeCost(parsedData)
print("Bisecting K-means Center = " + str(center))
print("Bisecting K-means Computer Cost = " + str(cost))
```

Gambar 12. Code Bisecting KMeans di databricks

Code tersebut, menghasilkan cluster center pada Gambar 13 dan menghasilkan longitude dan latitude yang tidak memiliki makna yang seharusnya.

```
Bisecting K-means Center =
[array([ -3.7441027 ,  51.54782719]),
 array([ -2.14693686,  52.38839324]),
 array([ -2.5089698 ,  53.52744447]),
 array([ -1.39759671,  53.92771066]),
 array([ -1.24958541,  51.03615209]),
 array([ -0.97142952,  52.12028742]),
 array([ -0.15613262,  51.49169413]),
 array([  0.63354896,  51.56637895]),
 array([  1.97843159e-02,   5.28580027e+01])]
```

Gambar 13. Hasil Cluster Center Bisecting KMeans.

Berdasarkan hasil tersebut, kami tidak menggunakan Bisecting KMeans sebagai hasil akhir visualisasi clustering kami.

Percobaan ketiga di databricks, yaitu menggunakan algoritma Machine Learning KMeans yang disediadakan oleh Spark versi 2.2 di databricks. Pada percobaan kali ini, diharapkan mendapat hasil yang dapat mengelompokkan Longitude dan Latitude berdasarkan kemiripannya, sehinga didapatkan lokasi-lokasi yang dikelompokkan sejumlah k yang diinput pada parameter KMeans. parameter k yang diinput yang sejumlah sembilan, yang dimana angka tersebut didapat berdasarkan banyaknya negara bagian yang ada di United

Kingdom, sehingga diharapkan, persebaran kemiripan Longitude Latitude menghasilkan sembilan daerah yang tersebar pada kesembilan negara bagian tersebut. Berikut ini pada Gambar 14 merupakan Code KMeans yang dijalankan di databricks.

Cmd 7

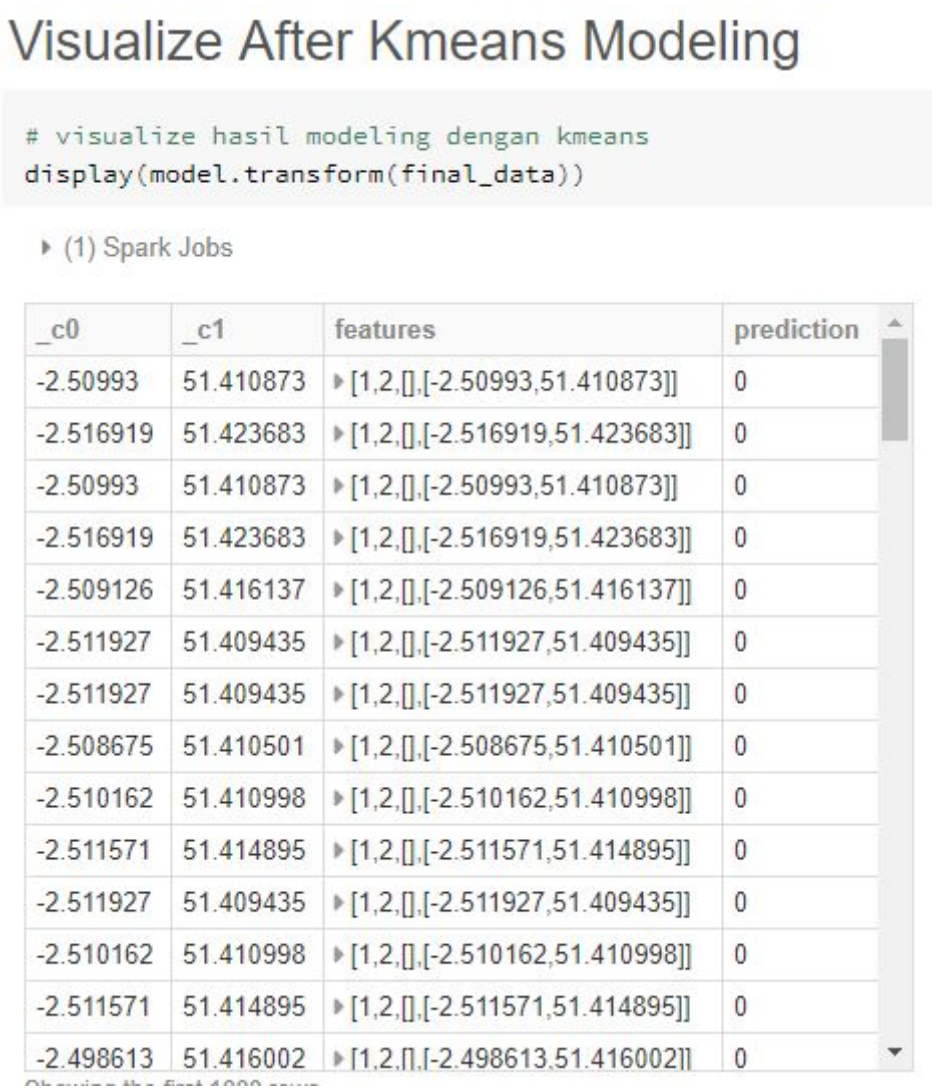
```
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler

# inisiasi appname
spark = SparkSession.builder.appName('cluster').getOrCreate()

# mengambil data csv yang merupakan dataset berupa lat, lng,
# membuat jadi bertipe data double
dataset = spark.read.format('csv').option("header", "false")
          .option("inferSchema","true")
          .load('FileStore/tables/output_positions_v2.csv')

# set up menjadi 9 cluster dengan kmeans
kmeans = KMeans().setK(9).setSeed(1)

# inisiasi menambahkan atribut features yang dimana atribut features
# yaitu memiliki value array of lat,lng
assembler = VectorAssembler(inputCols=dataset.columns,
                            outputCol = 'features')

# jadi punya kolom features
final_data = assembler.transform(dataset)

# inisiasi kmeans dengan mengeksekusi pada kolom bernama features
kmeans = KMeans(featuresCol='features',k=9)

# eksekusi kmeans terhadap dataset yang telah dibuat
model = kmeans.fit(final_data)

# visualize hasil modeling dengan kmeans
display(model.transform(final_data))

centers = model.clusterCenters()
size_summary = model.summary.cluster
total_data = model.summary.clusterSizes
total_cost = model.computeCost(final_data)

print("center "+ str(centers) +"\n")
print("total_data "+ str(total_data) + "\n")
print("total_cost "+ str(total_cost) + "\n")
print("size_summary " + str(size_summary) + "\n")
```

Gambar 14. Code KMeans di databricks
(Sumber: Olahan Penulis)

Proses modeling dengan algoritma machine learning KMeans di Spark databricks diterapkan dengan cara:

1. Dimulai dengan load data csv
2. Menentukan banyak k=9
3. Menambahkan kolom dan value ‘features’ pada tabel csv tersebut, value berupa vector yang menggabungkan atribut yang diinginkan dari inputan data csv kita
4. menginisasi kmeans agar membaca kolom ‘features’ dengan k=9 saat modeling dengan KMeans
5. lalu dieksekusi dengan perintah kmeans.fit(dataset)
6. setelah proses selesai, maka kita akan mendapatkan dataset baru berupa kolom prediction, cluster center, hingga cluster size

Berikut ini pada Gambar 15 merupakan hasil dari KMeans di databricks.



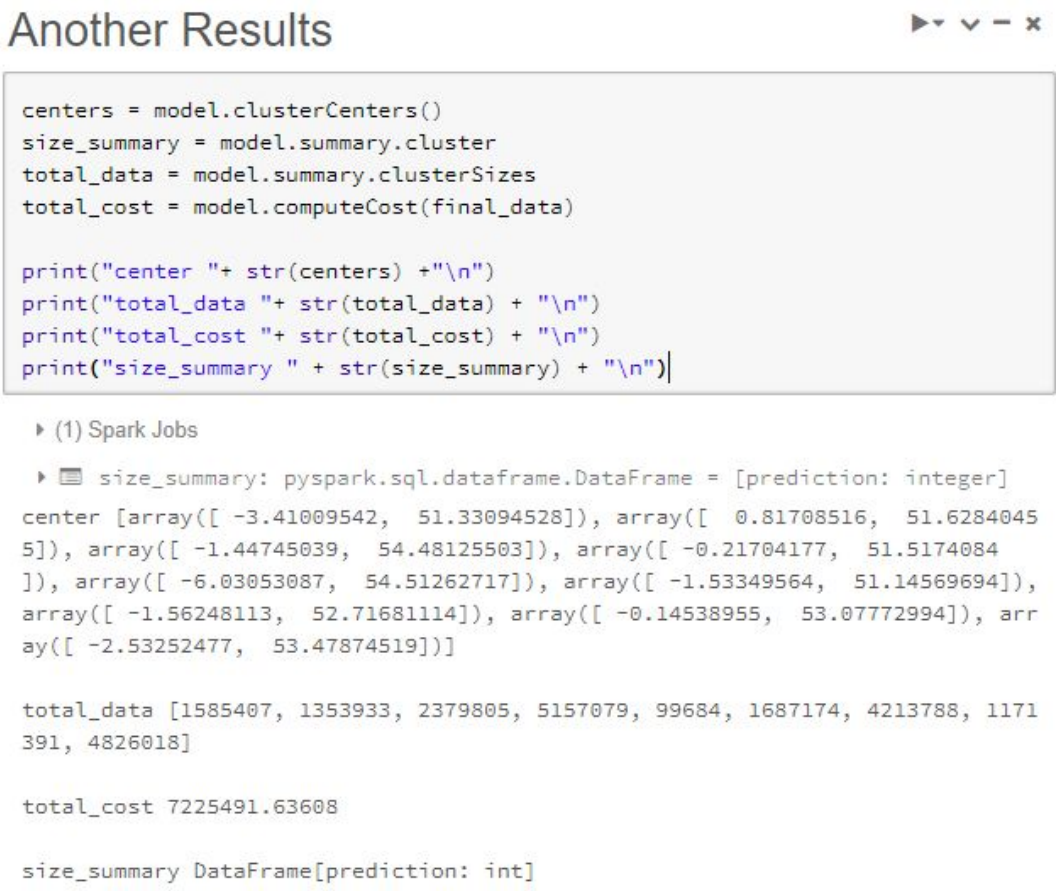
_c0	_c1	features	prediction
-2.50993	51.410873	[1,2,[],[-2.50993,51.410873]]	0
-2.516919	51.423683	[1,2,[],[-2.516919,51.423683]]	0
-2.50993	51.410873	[1,2,[],[-2.50993,51.410873]]	0
-2.516919	51.423683	[1,2,[],[-2.516919,51.423683]]	0
-2.509126	51.416137	[1,2,[],[-2.509126,51.416137]]	0
-2.511927	51.409435	[1,2,[],[-2.511927,51.409435]]	0
-2.511927	51.409435	[1,2,[],[-2.511927,51.409435]]	0
-2.508675	51.410501	[1,2,[],[-2.508675,51.410501]]	0
-2.510162	51.410998	[1,2,[],[-2.510162,51.410998]]	0
-2.511571	51.414895	[1,2,[],[-2.511571,51.414895]]	0
-2.511927	51.409435	[1,2,[],[-2.511927,51.409435]]	0
-2.510162	51.410998	[1,2,[],[-2.510162,51.410998]]	0
-2.511571	51.414895	[1,2,[],[-2.511571,51.414895]]	0
-2.498613	51.416002	[1,2,[],[-2.498613,51.416002]]	0

Gambar 15. Hasil KMeans di databricks.
(Sumber: Olahan Penulis)

Berikut ini merupakan penjelasakn dari hasil KMeans pada Gambar 15 di atas:

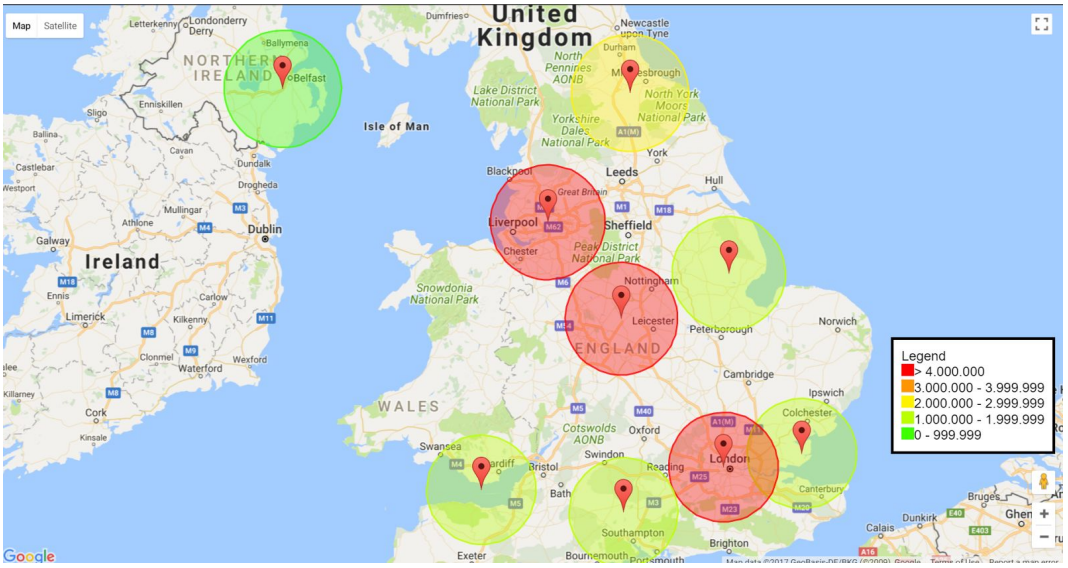
- _c0 merupakan Longitude
- _c1 merupakan Latitude
- features merupakan vector gabungan dari Longitude dan Latitude
- prediction merupakan labeling dari hasil clustering KMeans yang menandakan daerah cluster-nya, k=9, maka prediction memiliki value 0-8.

Berdasarkan data tersebut didapatkan hasil pada Gambar 16, yaitu Cluster size, cluster center, hingga computer cost.



Gambar 16. Hasil KMeans di databricks (2)
(Sumber: Olahan Penulis)

Berdasarkan hasil yang ada, maka didapatkan tujuan yang diinginkan, yaitu mendapatkan lokasi dengna Latitude dan Longitude dari cluster center, lalu, mendapatkan berapa banyak data yang ada pada suatu cluster dari cluster size sehingga diketahui ada berapa banyak kejahatan yang terjadi pada suatu cluster. Berikut ini pada Gambar 17 merupakan hasil visualisasi pada google maps.



Gambar 17. Hasil visualisasi KMeans databricks.

Namun, tim kami menginginkan pula ada jenis kriminal apa saja pada suatu daerah cluster tersebut. Sehingga, diperlukan untuk cleansing data lagi pada feature selain Latitude, Longitude dan melakukan join tabel terhadap tabel hasil modeling KMeans dengan sumber data. Sehingga akan didapatkan crime type pada suatu lokasi hasil clustering. Maka dari itu, tim kami melakukan percobaan terakhir / final ini.

Pada percobaan final ini, diawali dengan melakukan cleansing dari data csv sebanyak -+ 22 juta baris di databricks, dengan code cleansing berikut pada Gambar 18.

Cmd 3

```
# Cleansing (2) : Validate Datatype Value
setdata = spark.read.format('csv').option("header", "true")
          .option("inferSchema","true").load('FileStore/tables/crime/')
print("total baris sumber data: "+ str(setdata.count()))

cleaningset = setdata.withColumn("Latitude", setdata["Latitude"]
                                 .cast("double")).na.drop(subset=["Latitude"])

cleaningset2a = cleaningset.withColumn("Longitude", cleaningset["Longitude"]
                                       .cast("double")).na.drop(subset=["Longitude"])

cleaningset2b = cleaningset2a.withColumn("Month", cleaningset2a["Month"]
                                         .cast("integer")).na.drop(subset=["Month"])

cleaningset2 = cleaningset2a.na.drop(subset=[' Crime ID','Month','Year',
                                             'Reported by', 'Falls within',
                                             'Longitude', 'Latitude',
                                             'Location', 'LSOA code',
                                             'LSOA name', 'Crime type',
                                             'Last outcome category'])

valid = setdata.where(setdata["Latitude"].cast("double").isNotNull())
invalid = setdata.where(setdata["Latitude"].cast("double").isNull())

print("total baris aftercleansing: "+ str(cleaningset2.count()))

# melakukan cleansing, yaitu dengan pemilihan data yang memiliki value
# berdasarkan value crime type yang benar sesuai deskripsi pada data kepolisian UK
cleaningset2 = cleaningset2.where(
  (cleaningset2["Crime type"] == "Criminal damage and arson") |
  (cleaningset2["Crime type"] == "Theft from the person") |
  (cleaningset2["Crime type"] == "Public order") |
  (cleaningset2["Crime type"] == "Vehicle crime") |
  (cleaningset2["Crime type"] == "Violence and sexual offences") |
  (cleaningset2["Crime type"] == "Shoplifting") |
  (cleaningset2["Crime type"] == "Burglary") |
  (cleaningset2["Crime type"] == "Other theft") |
  (cleaningset2["Crime type"] == "Bicycle theft") |
  (cleaningset2["Crime type"] == "Robbery") |
  (cleaningset2["Crime type"] == "Other crime") |
  (cleaningset2["Crime type"] == "Drugs") |
  (cleaningset2["Crime type"] == "Possession of weapons") |
  (cleaningset2["Crime type"] == "social behaviour")
).drop()

# Visualize hasil cleansing
display(cleaningset2.orderBy("Crime type").groupBy("Crime type").count())
```

Gambar 18. Cleansing tahap 2 di databricks

Hasil cleansing tahap 2 di databrciks tersebut menghasilkan sekitar -+ 11 juta baris dari -+ 22 juta baris, tepatnya menjadi 11.949.434 baris. Hal ini dikarenakan, pada 22 juta baris tadi, hanya mengambil Latitude dan Longitude yang tidak null dan tidak memperdulikan atribut lain. Namun, saat dicek pada atribut lain seperti mulai dari Crime ID hingga Crime type, didapatkan ada sekitar 11 juta baris yang bernilai null dan ada yang memiliki value yang tidak sesuai tipe data pada suatu atribut, sehingga menjadi -+ 11 juta baris.

Setelah cleansing tahap 2 di databricks dilakukan, lalu tahap berikutnya ialah menjalankan algoritma machine learning KMeans pada data yang telah di cleansing tersebut. Berikut ini pada Gambar 19 merupakan code penerapan KMeans. Pada dasarnya, code-nya sama saja dengan percobaan sebelumnya.

Cmd 6

```
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler

# inisiasi appname
spark = SparkSession.builder.appName('cluster').getOrCreate()

# mengambil data csv yang merupakan dataset berupa lat, lng,
# membuat jadi bertipe data double
dataclean = cleaningset2.select("Longitude","Latitude")

# set up menjadi 9 cluster dengan kmeans
kmeans = KMeans().setK(9).setSeed(1)

# inisiasi menambahkan atribut features yang dimana
# atribut features yaitu memiliki value array of lat,lng
assembler = VectorAssembler(inputCols=dataclean.columns,
                            outputCol = 'features')

# jadi punya kolom features
final_data_clean = assembler.transform(dataclean)

# inisiasi kmeans dengan mengeksekusi
# pada kolom bernama features
kmeans = KMeans(featuresCol='features',k=9)

model = kmeans.fit(final_data_clean)

display(model.transform(final_data_clean))

centers = model.clusterCenters()
size_summary = model.summary.cluster
total_data = model.summary.clusterSizes
total_cost = model.computeCost(final_data_clean)

print("center "+ str(centers) +"\n")
print("total_data "+ str(total_data) + "\n")
print("total_cost "+ str(total_cost) + "\n")
print("size_summary " + str(size_summary) + "\n")
print("total baris setelah modeling with kmeans: "+
      str(final_data_clean.count()))
```

Gambar 19. Code KMeans databricks pada data cleansing tahap 2
(Sumber: Olahan Penulis)

Berdasarkan code tersebut, maka didapatkan tabel hasil KMeans, cluster center, cluster size, compute cost hingga total baris pada tabel data. Pada Gambar 20 dan Gambar 21 merupakan hasilnya.

Gambar 20. Hasil KMeans pada data cleansing tahap 2
(Sumber: Olahan Penulis)

```
display(model.transform(final_data_clean))
```

▸ (28) Spark Jobs

▸ dataclean: pyspark.sql.dataframe.DataFrame = [Longitude: double, Latitude: doub

▸ final_data_clean: pyspark.sql.dataframe.DataFrame = [Longitude: double, Latituc

Longitude	Latitude	features	prediction
-2.511571	51.414895	▸ [1,2,[],[-2.511571,51.414895]]	3
-2.515816	51.408717	▸ [1,2,[],[-2.515816,51.408717]]	3
-2.515816	51.408717	▸ [1,2,[],[-2.515816,51.408717]]	3
-2.494715	51.419948	▸ [1,2,[],[-2.494715,51.419948]]	3
-2.49793	51.417966	▸ [1,2,[],[-2.49793,51.417966]]	3
-2.497767	51.420232	▸ [1,2,[],[-2.497767,51.420232]]	3
-2.502805	51.414033	▸ [1,2,[],[-2.502805,51.414033]]	3
-2.500214	51.416481	▸ [1,2,[],[-2.500214,51.416481]]	3
-2.501425	51.416692	▸ [1,2,[],[-2.501425,51.416692]]	3
-2.501425	51.416692	▸ [1,2,[],[-2.501425,51.416692]]	3
-2.497799	51.415233	▸ [1,2,[],[-2.497799,51.415233]]	3
-2.49854	51.414618	▸ [1,2,[],[-2.49854,51.414618]]	3
-2.497799	51.415233	▸ [1,2,[],[-2.497799,51.415233]]	3
-2.49854	51.414618	▸ [1,2,[],[-2.49854,51.414618]]	3
-2.498613	51.416002	▸ [1,2,[],[-2.498613,51.416002]]	3
-2.498613	51.416002	▸ [1,2,[],[-2.498613,51.416002]]	3

Gambar 21. Tabel Hasil KMeans pada data cleansing tahap 2
(Sumber: Olahan Penulis)

Setelah mendapatkan hasil dari modeling dengan KMeans, maka tahap berikutnya ialah menambahkan kolom / atribut id unik pada tabel sumber data yang telah di cleansing dan pada tabel hasil KMeans. Hal ini dilakukan agar dapat melakukan join dengan relasi one-to-one. Sehingga akan didapatkan atribut prediction, crime type, longitude, latitude dan lainnya pada satu tabel yang sesuai pada tiap barisnya dan total baris akan tetap -+ 11 juta baris. Berikut ini pada Gambar 22 merupakan code untuk melakukan penambahan id pada tabel sumber data dan tabel hasil KMeans.

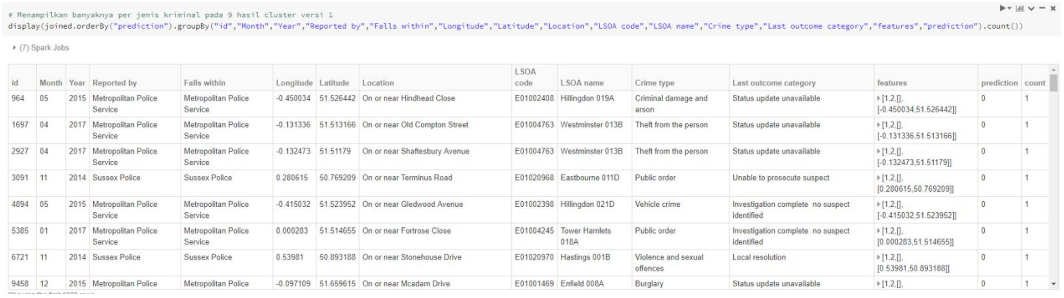
```
# inisiasi variable dataset
dataset1 = cleaningset2
dataset2 = model.transform(final_data_clean)

# start code for dataset2
from pyspark.sql import Row
from pyspark.sql.types import StructField, StructType, LongType

row = Row("foo", "bar")
row_with_index = Row(*["id"] + dataset2.columns)

def make_row(columns):
    def _make_row(row, uid):
        row_dict = row.asDict()
        return row_with_index(*[uid] + [row_dict.get(c) for c in columns])
    return _make_row

f = make_row(dataset2.columns)

df_with_pk2 = (dataset2.rdd
    .zipWithUniqueId()
    .map(lambda x: f(*x))
    .toDF(StructType([StructField("id", LongType(), False)]
                     + dataset2.schema.fields)))

################################################################################

# start code for dataset1
from pyspark.sql import Row
from pyspark.sql.types import StructField, StructType, LongType

row = Row("foo", "bar")
row_with_index = Row(*["id"] + dataset1.columns)

def make_row(columns):
    def _make_row(row, uid):
        row_dict = row.asDict()
        return row_with_index(*[uid] + [row_dict.get(c) for c in columns])
    return _make_row

f2 = make_row(dataset1.columns)

df_with_pk1 = (dataset1.rdd
    .zipWithUniqueId()
    .map(lambda x: f2(*x))
    .toDF(StructType([StructField("id", LongType(), False)]
                     + dataset1.schema.fields)))
```

Gambar 22. Code penambahan atribut ID unik
(Sumber: Olahan Penulis)

Setelah dilakukan penambahan atribut id untuk dua buah dataset, maka berikutnya dilakukan join pada dua dataset tersebut. Berikut ini pada Gambar 23 merupakan code untuk melakukan join pada dataset 1: sumber data setelah cleansing dan dataset 2: tabel hasil KMeans.

Gambar 23. Code Join Dataset 1 dan 2
(Sumber: Olahan Penulis)

Setelah dilakukan join pada dua dataset tersebut, maka berikut ini Gambar 24 merupakan hasil join tabelnya.



id	Month	Year	Reported by	Falls within	Longitude	Latitude	Location	LSOA code	LSOA name	Crime type	Last outcome category	features	prediction	count
964	05	2015	Metropolitan Police Service	Metropolitan Police Service	-0.450034	51.526442	On or near Hindhead Close	E01002408	Hillingdon 019A	Criminal damage and arson	Status update unavailable	▸[1,2,[],[-0.450034,51.526442]]	0	1
1697	04	2017	Metropolitan Police Service	Metropolitan Police Service	-0.131336	51.513166	On or near Old Compton Street	E01004763	Westminster 013B	Theft from the person	Status update unavailable	▸[1,2,[],[-0.131336,51.513166]]	0	1
2927	04	2017	Metropolitan Police Service	Metropolitan Police Service	-0.132473	51.51179	On or near Shaftesbury Avenue	E01004763	Westminster 013B	Theft from the person	Status update unavailable	▸[1,2,[],[-0.132473,51.51179]]	0	1
3091	11	2014	Sussex Police	Sussex Police	0.280615	50.769209	On or near Terminus Road	E01020968	Eastbourne 011D	Public order	Unable to prosecute suspect	▸[1,2,[],[0.280615,50.769209]]	0	1
4894	05	2015	Metropolitan Police Service	Metropolitan Police Service	-0.415032	51.523952	On or near Gledwood Avenue	E01002398	Hillingdon 021D	Vehicle crime	Investigation complete no suspect identified	▸[1,2,[],[-0.415032,51.523952]]	0	1
5385	01	2017	Metropolitan Police Service	Metropolitan Police Service	0.000283	51.514655	On or near Fortrose Close	E01004245	Tower Hamlets 018A	Public order	Investigation complete no suspect identified	▸[1,2,[],[0.000283,51.514655]]	0	1
6721	11	2014	Sussex Police	Sussex Police	0.53981	50.893188	On or near Stonehouse Drive	E01020970	Hastings 001B	Violence and sexual offences	Local resolution	▸[1,2,[],[0.53981,50.893188]]	0	1
9458	12	2015	Metropolitan Police	Metropolitan Police	-0.097109	51.659615	On or near Mcadam Drive	E01001469	Enfield 008A	Burglary	Status update unavailable	▸[1,2,[],	0	1

Gambar 24. Tabel Hasil Join
(Sumber: Olahan Penulis)

Berdasarkan hasil pada Gambar 24 tersebut, maka didapatkan 14 atribut, antara lain: id, Month, Year, Reported by, Falls within, Longitude, Latitude, Location, LSOA code, LSOA name, Crime type, Last outcome, category, features, dan prediction. Lalu, atribut yang akan digunakan yaitu, prediction, latitude, longitude, month, year. Atribut lain bisa saja digunakan jika diperlukan selanjutnya untuk tujuan lain. Berikut ini pada Gambar 25 merupakan hasil agregasi yang menggambarkan suatu daerah cluster memiliki jenis kriminal apa saja dan berapa banyak.

prediction	Bicycle theft	Burglary	Criminal damage and arson	Drugs	Other crime	Other theft	Possession of weapons	Public order	Robbery	Shoplifting
0	89114	335622	361111	145343	51583	490633	29798	211319	82373	251223
1	19403	112513	180501	40734	18291	117059	6424	65800	8344	97131
2	23146	101912	133611	27693	16753	103313	6720	38001	8514	96292
3	18456	90861	155619	41378	18796	124178	5670	82141	5955	100000
4	23643	196344	243951	38724	32371	194163	11324	116339	22024	133176
5	10712	57738	113930	17686	12153	75544	4372	44574	3544	69292
6	28607	165147	190088	39937	23184	160670	10648	57998	25145	136197
7	17353	75836	117885	22686	16513	88411	5234	48974	6765	69126
8	27556	76704	114002	26039	13531	91981	5921	56340	4917	78972

Gambar 25. Hasil Lokasi dan Crime Type
(Sumber: Olahan Penulis)

Berdasarkan hasil yang didapat dari percobaan final menggunakan KMeans di databricks, melakukan cleansing tahap dua, dan hingga melakukan join tabel dengan databricks tersebut. Maka, kami memutuskan bahwa percobaan final ini yang akan digunakan untuk visualisasi final dan selanjutnya. Namun, tetap kami lakukan evaluasi analisis akurasi terhadap pemilihan ini pada bagian Evaluasi Analisis Akurasi untuk setiap percobaan yang telah kami lakukan. Percobaan final ini dapat diakses pada *notebook databricks* yang ada pada link berikut ini: bit.ly/result3final yang berisi code beserta hasil visualisasinya.

C. Evaluasi Analisis Akurasi

Percobaan ini menggunakan metode *clustering*. Metode *clustering* merupakan metode untuk mengelompokkan data berdasarkan sifat - sifat tertentu yang ada pada dataset. Sehingga pada dasarnya, dengan demikian, tidak terdapat akurasi pada percobaan ini seperti halnya klasifikasi yang memiliki metode khusus dalam mengevaluasi hasilnya.

Namun, kami mencoba mengevaluasi hasil clustering agar menjawab apakah data hasil clustering ini sudah baik, maka kami menggunakan kriteria time execution, data Within Set Sum of Squared Error (WSSSE) compute cost dari fungsi yang ada pada Spark. Lalu, WSSSE compute cost tersebut kami terapkan pada metode evaluasi *Elbow Method.* Evaluasi ini digunakan untuk mendapatkan nilai k yang terbaik dengan mencoba beberapa nilai k sehingga mendapatkan hasil k yang terbaik dengan menerapkan *Elbow Method.* Dengan adanya data compute cost, maka setiap percobaan nilai k dicatat compute cost nya, lalu dibuat visualisasi agar nilai WSSSE Error compute cost nya terlihat dan dapat ditentukan nilai k yang cocok dari hasil WSSSE tersebut. Pada dasarnya, nilai WSSSE tersebut akan selalu berkurang saat nilai k semakin membesar, hal ini dikarenakan angka dari cluster terus meningkat sehingga pengelompokkan per cluster semakin kecil, sehingga distorsi juga akan semakin mengecil. Ide dari *Elbow Method* yaitu untuk memilih k yang dimana nilai WSSSE berkurang

secara drastis. Hal ini menghasilkan *Elbow Effect* pada grafik yang akan ditampilkan pada evaluasi kedua dan ketiga di bawah.

Berikut ini pada Tabel 1 merupakan evaluasi tahap pertama, yaitu dengan 6 percobaan dengan k=9 dan input data sebanyak 22.474.278 baris. Keenam percobaan pada Tabel 1 ini telah dijelaskan sebelumnya, yaitu pada server fasilkom dan databricks.

Tabel 1. Evaluasi Tahap Pertama

No	Percobaan	Time of Execution (minutes)	Compute Cost	Keterangan
1	Databricks - KMeans Scaler standard	1.42	-	Results of Lat,Lng are missed value
2	Databricks - Bisecting KMeans	47.82	-	Results of Lat,Lng are missed value, compute cost not supported
3	Databricks - KMeans	1.48	7,225,491.64	Good results, clusterSize supported
4	Server Fasilkom - KMeans ML	14	6,765,665.56	Spark 1.6 not supported for clusterSize
5	Server Fasilkom - KMeans MLlib	38	9,173,528.92	Spark 1.6 not supported for clusterSize
6	Server Fasilkom - Bisecting KMeans	-	-	Error, Spark 1.6 Not Supported Bisecting KMeans

Berdasarkan hasil tersebut, didapat bahwa Databricks KMeans Scaler Standard memiliki time of execution paling kecil, dikarenakan seluruh atribut yang dipiilh untuk dilakukan proses KMeans telah distandarkan dan dimasukkan pada satu atribut bernama ScaledStandard. Sehingga, saat proses eksekusi menjadi lebih efisien waktu yang dijalankan.

Namun, pada Server Fasilkom paling lama yaitu ada pada KMeans MLlib, selama 38 menit karena Spark mllib merupkan original API dengan menggunakan RDD, sementara spark ml menyediakan higher-level dengan menggunakan DataFrame. Hal lainnya, yaitu karena proses eksekusi dengan KMeans langsung membaca file csv, sedangkan di databricks dilakukan preparation data yaitu dengan cara menambahkan satua atribut bernama ‘features’ yang menggabungkan atribut-atribut yang akan dilakukan proses KMeans, sehingga lebih efisien saat melakukan eksekusi KMeans.

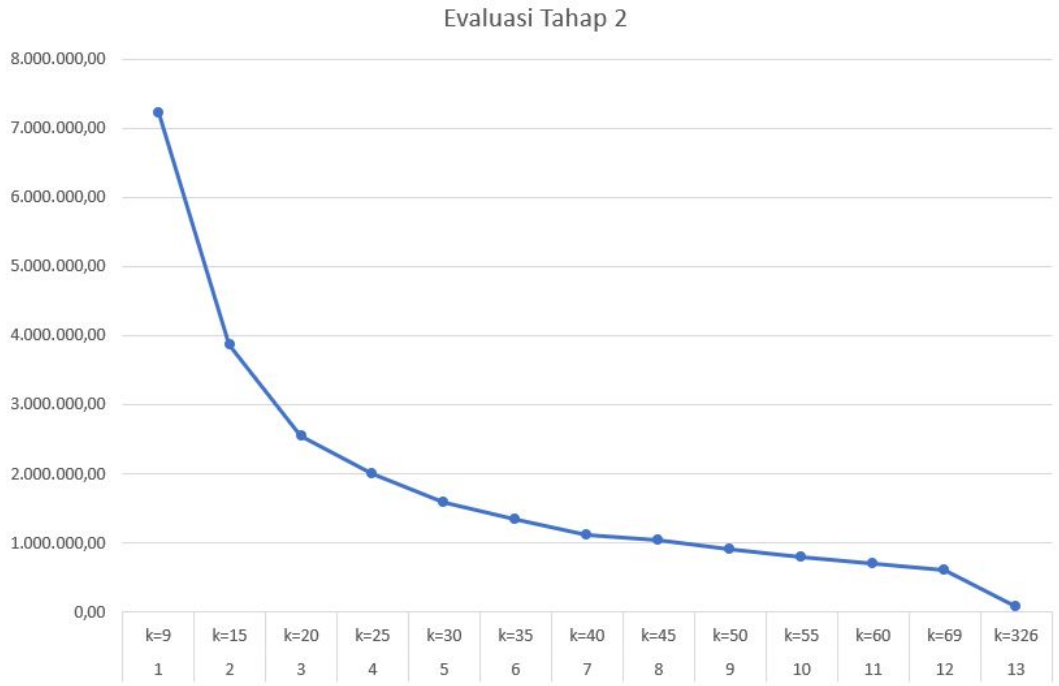
Dengan demikian, maka dilanjutkan evaluasi dengan menggunakan percobaan nomor 3, yaitu dengan databricks KMeans, karena memiliki waktu eksekusi yang efisien, compute cost yang bagus serta menghasilkan hasil yang diinginkan dan sesuai. Dengan percobaan nomor 3 tersebut, maka akan diterapkan evaluasi tahap kedua.

Berikut ini pada Tabel 2 merupakan evaluasi tahap kedua, yang dimana menggunakan input data sebesar 22.474.278 baris dengan nilai k yang berbeda dan dilakukan di databricks. Nilai k dimulai dari 9, lalu increment 5, lalu ada nilai k = 69 yang merupakan banyak kota yang ada di United Kingdom dan nilai k=326 yang merupakan banyaknya distrik di United Kingdom.

Tabel 2. Evaluasi Tahap Kedua

No	Percobaan	Kriteria	
		Time	Compute Cost
1	k=9	1.48	7,225,491.64
2	k=15	1.73	3,863,472.57
3	k=20	1.75	2,545,130.74
4	k=25	1.85	2,008,765.81
5	k=30	2.02	1,594,961.67
6	k=35	2.07	1,342,563.30
7	k=40	2.12	1,120,203.37
8	k=45	2.25	1,033,636.70
9	k=50	2.36	904,262.09
10	k=55	2.78	792,891.74
11	k=60	2.68	700,201.56
12	k=69	2.87	610,027.98
13	k=326	13.68	81,251.82

Berdasarkan hasil dari Tabel 2 pada evaluasi tahap kedua, maka dilakukan visualisasi grafik agar mendapatkan Elbow Effect, sehingga akan mendapatkan nilai k yang sesuai. Grafik divisualisasikan pada Gambar 26 berikut.



Gambar 26. Grafik Evaluasi TahapKedua
(Sumber: Olahan Penulis)

Berdasarkan Grafik pada Gambar 26 tersebut, maka didapatkan nilai k = 20 memiliki Elbow Effect, yang dimana nilai k=15 ke k=20 menghasilkan penurunan nilai WSSSE secara drastis jika dibandingkan nilai k=20 ke k=25 dan seterusnya tidak sedrastis saat k=15 ke 20.

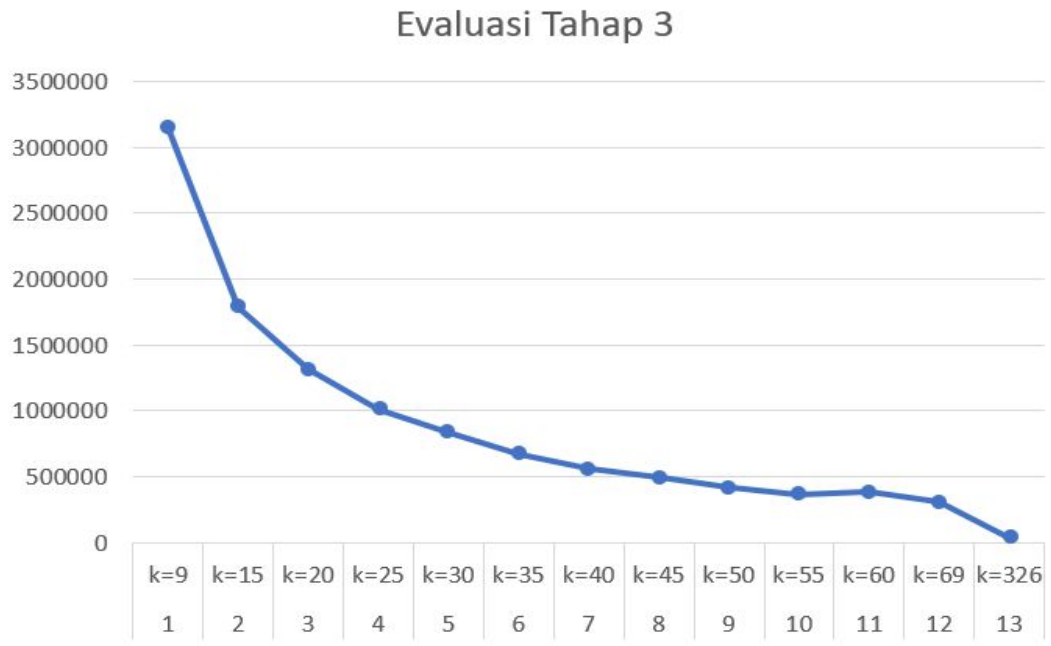
Pada evaluasi tahap ketiga, dilakukan evaluasi pada percobaan final yaitu menggunakan dataset yang telah di-cleansing di databricks. Berikut ini pada Tabel 3 merupakan evaluasi tahap ketiga, yang dimana menggunakan input data sebesar 11.949.434 baris, sama dengan percobaan kedua, yaitu dengan nilai k yang berbeda dan dilakukan di databricks. Nilai k dimulai dari 9, lalu increment 5, lalu ada nilai k = 69 yang merupakan banyak kota yang ada di United Kingdom dan nilai k=326 yang merupakan banyaknya distrik di United Kingdom.

Tabel 3. Evaluasi Tahap Ketiga

No	Percobaan	Kriteria	
		Time	Compute Cost
1	k=9	2	3151794.96
2	k=15	2.18	1788284.333
3	k=20	2.3	1312148.093
4	k=25	2.33	1011639.479
5	k=30	2.22	836118.2553
6	k=35	2.2	675598.1969
7	k=40	2.42	560196.1955

8	k=45	7.92	495901.9537
9	k=50	9.16	418962.9819
10	k=55	3.79	368188.1978
11	k=60	2.8	382038.3566
12	k=69	2.68	305258.7229
13	k=326	5.24	41121.45144

Berdasarkan hasil dari Tabel 3 pada evaluasi tahap ketiga, maka dilakukan visualisasi grafik agar mendapatkan Elbow Effect, sehingga akan mendapatkan nilai k yang sesuai. Grafik divisualisasikan pada Gambar 27 berikut.



Gambar 27. Grafik Evaluasi Tahap Ketiga
(Sumber: Olahan Penulis)

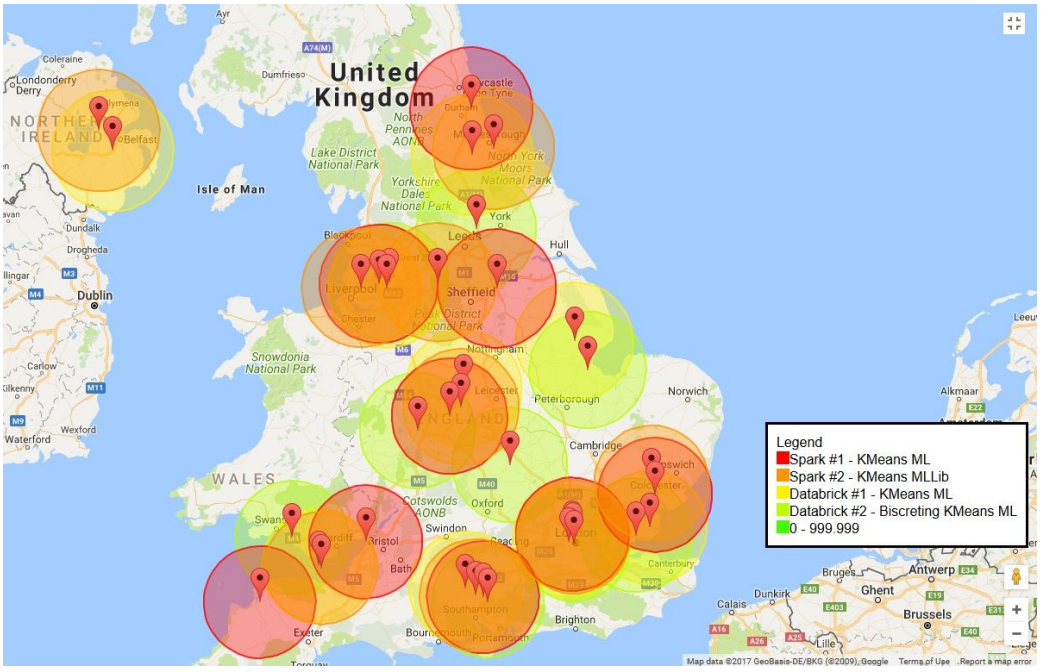
Berdasarkan Grafik pada Gambar 27 tersebut, maka didapatkan nilai k = 20 memiliki Elbow Effect, yang dimana nilai k=15 ke k=20 menghasilkan penurunan nilai WSSSE secara drastis jika dibandingkan nilai k=20 ke k=25 dan seterusnya tidak sedrastis saat k=15 ke 20. Sehingga, dapat disimpulkan bahwa, Elbow Effect yang dihasilkan pada evaluasi tahap kedua dan tahap ketiga memiliki nilai k yang sama, yaitu k=20. Maka dari itu, nilai k=9 yang telah dilakukan diawal merupakan asumsi agar dipetakan sesuai dengan banyaknya negara bagian di United Kingdom tidak sesuai dengan data yang ada. Namun, nilai k=20 beradasarkan penerapan *Elbow Method,* didapatkan penyebaran yang merata karena mendapatkan *Elbow Effect*. Oleh karena itu, kami akan memvisualisasikan ke peta dengan 9 daerah cluster dan 20 daerah cluster Latitude dan Longitude dengan data final yaitu sebanyak 11.949.434 baris.

OUTPUT SOFTWARE

Aplikasi dapat diakses pada domain: http://kawe.pe.hu/

Pengolahan data kriminalitas yang kami lakukan memberikan hasil berupa sembilan buah *cluster* yang merupakan representasi persebaran tindak kriminalitas di United Kingdom(UK). Hasil *clustering* tersebut ditampilkan secara visual, menggunakan aplikasi berbasis *website* dengan menggunakan *framework* Laravel dan menghubungkannya dengan Google Maps JavaScript API.

Aplikasi tersebut dinamai dengan Kawe, merupakan aplikasi yang berfungsi untuk membantu masyarakat dan kepolisian United Kingdom untuk mengetahui daerah yang memiliki tingkat kriminalitas yang tinggi. Aplikasi ini akan menampilkan peta lokasi daerah kriminalitas beserta radiusnya dan akan menunjukkan total kriminalitas yang terjadi. Kami membuat visualisasi peta United Kingdom dengan sembilan buah *pivot* beserta radius dari masing-masing *cluster*.



Gambar 28. Screenshot output software
(Sumber : Olahan penulis)

Warna merah manandakan hasil clustering menggunakan Spark ML versi 1.6 dengan algoritma K-Means. Warna oranye, menandakan hasil clustering Spark Mllib versi 1.6 menggunakan K-Means. Warna kuning, menandakan hasil clustering menggunakan databricks, menggunakan Spark versi 2.0.0 dengan algoritma K-Means ML. Warna hijau muda, menandakan hasil clustering menggunakan databricks, dengan algoritma Bisrecting K-Means.

CARA PENGGUNAAN APLIKASI

A. Cara Instalasi

Untuk dapat menggunakan software Kawe, terdapat beberapa tahapan yang harus dilakukan untuk melakukan installasi. Adapun tahapan tersebut adalah sebagai berikut:

1. Menginstall Composer

   Software Kawe dikembangkan merupakan aplikasi berbasis website yang dikembangkan dengan menggunakan framework Laravel dan bahasa pemrograman PHP. Untuk itu diperlukan composer sebagai *Dependency Manager* PHP*.

   a. Linux/Unix/OSX

      Untuk pengguna *operating system* Linux/Unix/OSX, berkas dan petunjuk installasi dapat ditemukan pada tautan berikut: https://getcomposer.org/doc/00-intro.md#installation-linux-unix-osx

   b. Windows

      Untuk pengguna operating system Windows, berkas dan petunjuk installasi dapat ditemukan pada tautan berikut: https://getcomposer.org/doc/00-intro.md#installation-windows

   * Pengguna yang sudah menginstall Composer, silahkan lewati tahap ini.
   * Pengecekan installasi Composer dapat dilakukan dengan menggunakan perintah “composer -v” pada cmd/terminal.

2. Melakukan Clone Project atau Mendownload Berkas Software

   Berkas software bisa didapatkan dengan clone project pada GitHub, atau dengan mendownload berkas pada .zip tugas akhir.

   a. Clone Project pada GitHub

      Berkas project dapat didapatkan dengan clone git pada tautan berikut:
      https://github.com/mahdifr17/PDB-Kawe.git

   b. Download Berkas Tugas Akhir

      Berkas project juga bisa didapatkan dari submission SCeLE CS UI. Dan berada pada folder “Software”.

3. Mendownload file .env

   Project yang menggunakan framework Laravel membutuhkan file .env sebagai *environment* variable untuk menjalankan software. File tersebut bisa didapatkan pada tautan berikut:
   https://drive.google.com/open?id=1Smm8iQG9KaFQxmzGZIRtCzrbImHS3Rtq

   Setelah file tersebut didownload, tempatkan pada folder project.

4. Melakukan Inisiasi Project

   Pada folder project, buka cmd/terminal dan masukan perintah “composer install”. Hal tersebut digunakan untuk mendapatkan file

library-library yang digunakan pada project. Hal ini perlu dilakukan karena file tersebut memiliki ukuran yang cukup besar, dan secara *default* termasuk kedalam .gitignore.

5. Jalankan Project

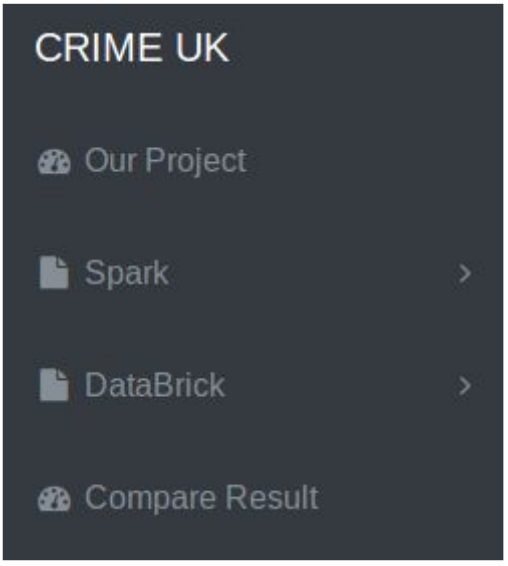
   Untuk menjalankan software, dilakukan perintah pada cmd/terminal. Masukan perintah “php artisan serve”, kemudian software akan berjalan pada localhost dan port 8000.

B. User Manual

Software Kawe memiliki dua buah fitur utama, yaitu menampilkan data masing-masing hasil percobaan berupa *code* untuk *cleaning* data, *sample data*, Spark *code*, serta visualisasi hasil *clustering*, dan fitur kedua adalah memvisualisasikan perbandingan antara seluruh pengujian.

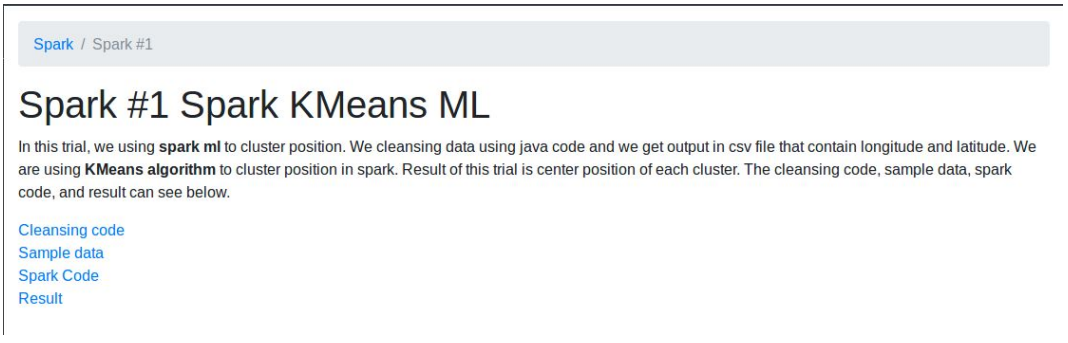
1. Menampilkan Data Masing-masing Percobaan

   Fitur ini dapat diakses melalui sidebar kiri pada software. Terdapat pilihan untuk melihat hasil pengujian pada server Fasilkom, dan pengujian pada DataBricks.



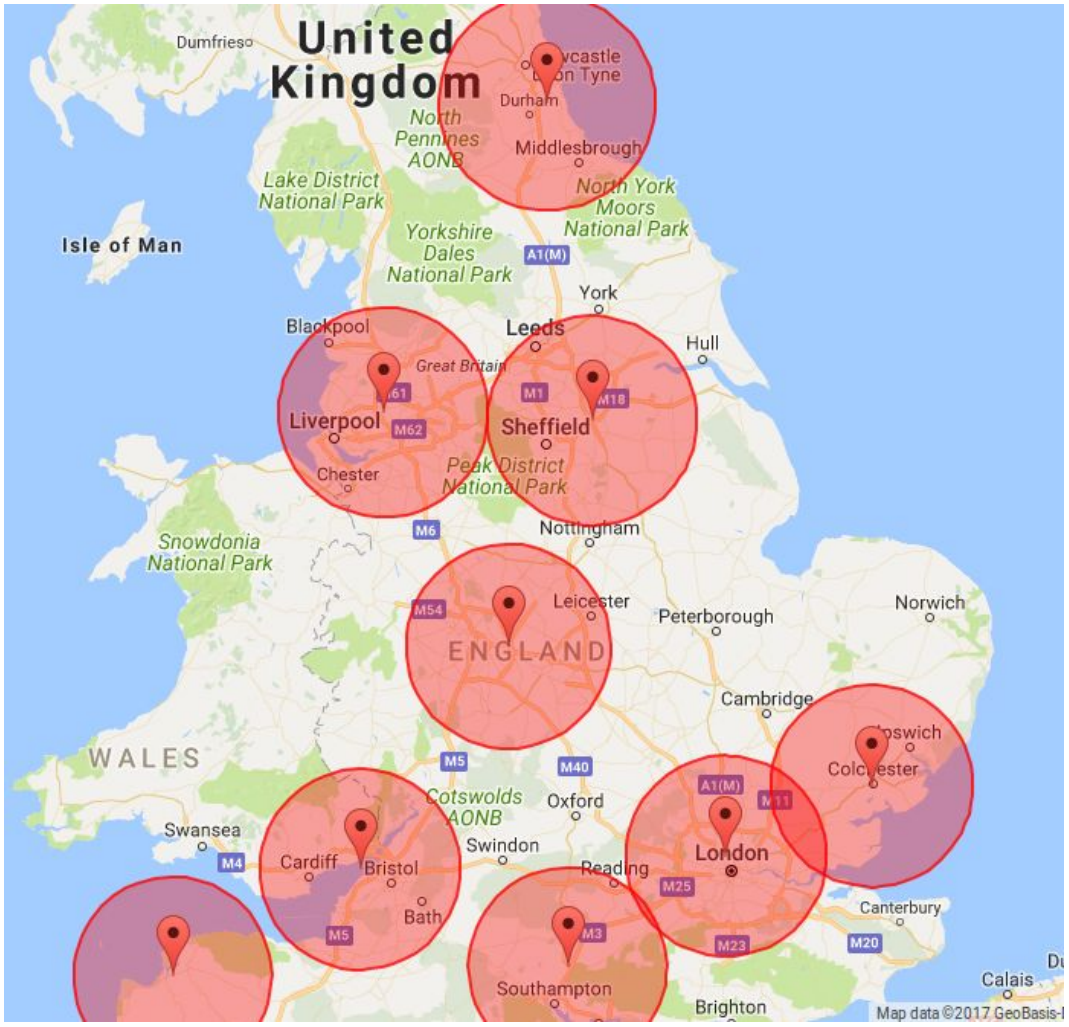
Gambar 29. Screenshot sidebar software
(Sumber : Olahan penulis)

Setelah pengguna mimilih pengujian mana yang ingin dilihat. Pengguna mendapatkan tampilan sebagai berikut:



Gambar 30. Screenshot fitur pertama software
(Sumber : Olahan penulis)

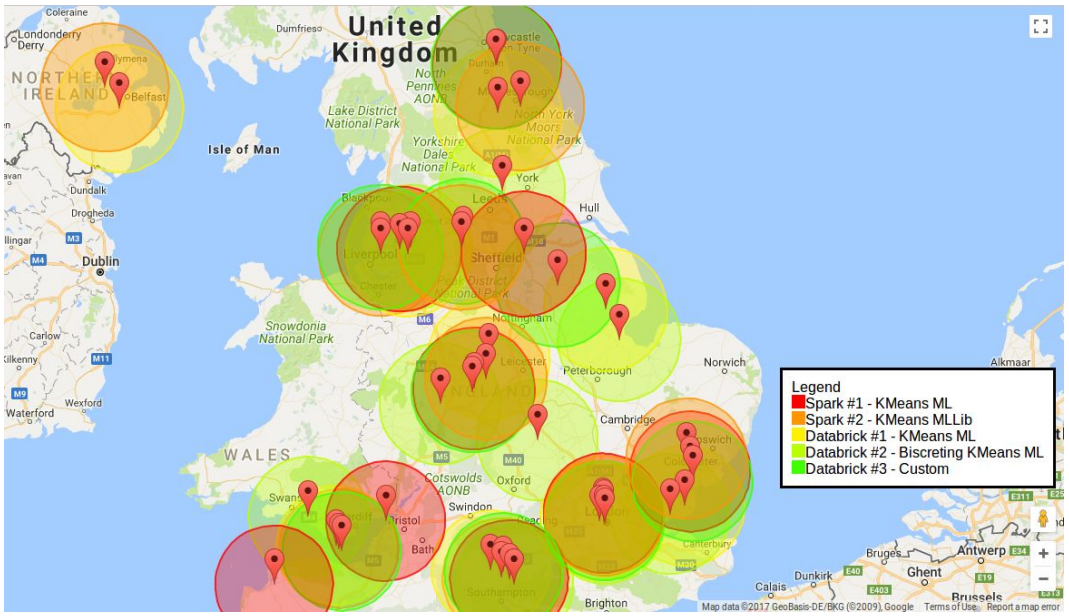
Untuk pilihan “Cleansing code”, pengguna akan diberikan file *cleansing* yang digunakan, begitu pula dengan “Sample data” dan “Spark Code”. Sedangkan untuk “Result”, pengguna akan diberikan visualisasi hasil clustering pada peta UK.



Gambar 31. Screenshot peta pada fitur pertama software
(Sumber : Olahan penulis)

2. Menampilkan Perbandingan Antara Seluruh Percobaan

Fitur ini dapat diakses melalui sidebar kiri pada software, pada menu “Compare Result”. Menu tersebut akan menampilkan peta yang menunjukan komparasi perbandingan masing-masing metode clustering.



Gambar 32. Screenshot peta pada fitur kedua software
(Sumber : Olahan penulis)

Daftar Pustaka

CRISP-DM 1 - The Modeling Agency. (n.d.). Retrieved November 9, 2017, from https://www.the-modeling-agency.com/crisp-dm.pdf

Data Mining Techniques. (n.d.). Retrieved November 09, 2017, from http://www.zentut.com/data-mining/data-mining-techniques/

Districts of England. (2017, December 21). Retrieved December 31, 2017, from https://en.wikipedia.org/wiki/Districts_of_England

KDnuggets. (n.d.). Retrieved November 09, 2017, from https://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html

Nava, J., & Hernández, P. (n.d.). Optimization of a Hybrid Methodology (CRISP-DM). Data Mining, 1998-2020.

Hadoop and the Modern Data Architecture. (n.d.). Retrieved November 09, 2017, from https://hortonworks.com/event/hadoop-modern-data-architecture -9/

How can we choose a "good" K for K-means clustering? (n.d.). Retrieved December 31, 2017, from https://www.quora.com/How-can-we-choose-a-good-K-for-K-means-clustering

List of cities in the United Kingdom. (2017, December 30). Retrieved December 31, 2017, from https://en.wikipedia.org/wiki/List_of_cities_in_the_United_Kingdom

Sklearn: calculating accuracy score of k-means on the test data set. (n.d.). Retrieved December 31, 2017, from https://stackoverflow.com/questions/37842165/sklearn-calculating-accuracy-score-of-k-means-on-the-test-data-set